Price Setting

Accelerator

Jul 2024

Accelerate Price Setting Package

With the help of Price Setting Package you can manage Price Lists / Live Price Grids and set up your pricing processes.

In this documentation:

- Business User Overview (Price Setting)
- Business User Reference (Price Setting)
- Admin User Reference (Price Setting)
- Technical User Reference (Price Setting)
- Glossary (Price Setting)
- Release Notes (Price Setting)
- Archive of Previous Versions (Price Setting)

Business User Overview (Price Setting)

The Price Setting Accelerator generates global and local list prices, integrates adjustments and on-invoice discounts, and automates recalculation for static/dynamic price lists.

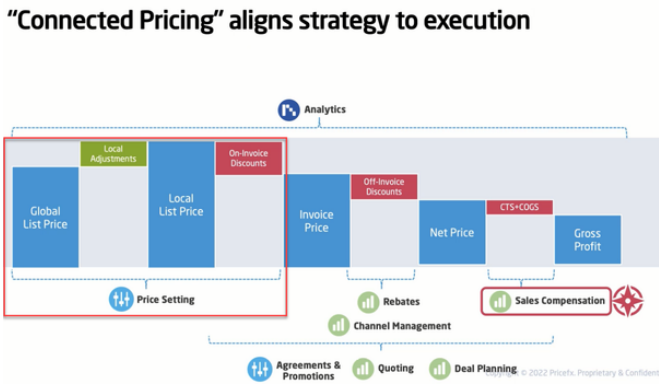
It is designed to address a frequent issue in many businesses: the processes for the creation and generation of price lists are too cumbersome and inefficient and they slow down reactions to pricing opportunities in the market. Also, the pricing strategies utilized are inflexible to market changes and the approval process for price changes is too time-consuming.

That is why Price Setting Package provides a framework that focuses on accuracy, consistency, efficiency, dependability, and responsiveness to provide a single source of truth for product pricing.

You can also watch a video introducing Price Setting and its usage.

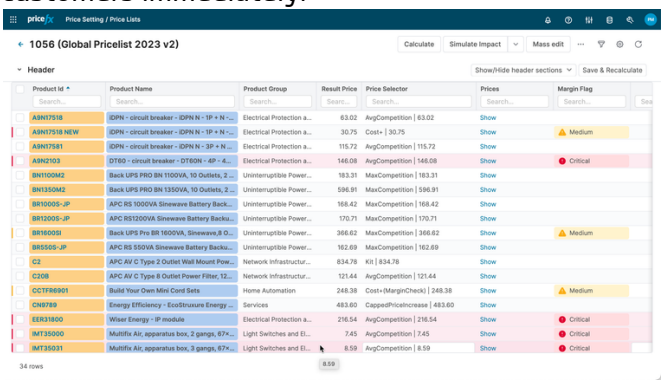
Pricefx Key Accelerators

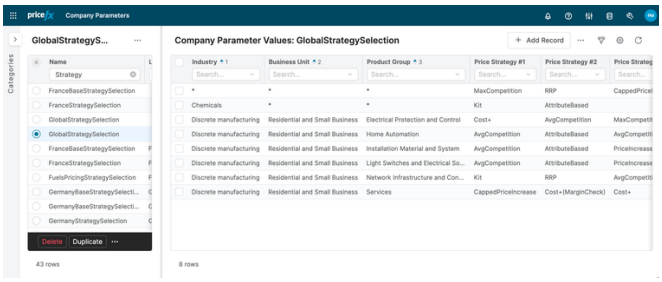
This accelerator provides aggregation at the following points in the waterfall:



Price Setting Package Capabilities

Price Setting Package (in close cooperation with Price Flexibility Package) provides a price setting and management framework powered by several out-of-the-box pricing strategies that pricing managers can use to create list prices, applying pricing strategies to different parts of their product portfolio. Once new list prices are created, pricing managers can communicate them to their sales teams or customers immediately.





Key Functionalities

Trade Level Scenarios

Price Setting allows you to define if prices are calculated as **List Prices or Gross List Prices**. Moreover, it can calculate **Net Prices** after discounts are applied.

It is worth mentioning that in B2C environments, prices are usually set as List Prices. In B2B ones, Net Prices are more commonly discounted List Prices. You can calculate all these trade levels with the Price Setting accelerator.

Price Levels

You can define several price levels and dependencies between them.

- **Independent** level – Prices on this level are not derived from other levels. You can think about it as "top" pricing level.
- **Dependent** level – Prices of this level are derived from another parent level. You can think about them as "lower" levels of the dependency hierarchy.
- **Complex pricing** level structure – You can build a complex structure. You can define several Parent Price Lists calculating prices independently. You can arrange various Dependent Price Lists around them. You can also create a multi-level tree where a Parent Price List depends on another Parent Price List. With this you can have for example multiple Parent Price Lists for different regions (EMEA, AMER,

	APAC). Dependent on them you can have Price Lists for the different countries, and dependent on them different channels, shops etc. • **Standalone Price List** – Covers price setting for just one price list.
Dynamic Pricing	Price are updated automatically based on an underlying product or competitor changes.
Pricing Strategies	You can select **out-of-the-box** pricing strategies such as calculations based on anchor, competition, BoM data or attribute-based pricing. You can also **configure the strategies** to your needs by using the additional setup options i.e. you can define their hierarchy, on which level they are valid, whether they can be overridden, how they work with exceptions.
Product Segmentation	In Price Setting you can set up **dimensions of your product master data** th at are used **to diversify your pricing strategy**. The resulting product segmentation helps you define all your pricing decisions on every level of granularity. The dimensions are looked up hierarchically, so that you can define some high-level pricing strategy and specify special product segments on a more granular level where needed.
Approval Workflow	Done through the Approval Workflow Package.

Configuration Wizard

A comprehensive wizard is provided to help you with the configuration. It is used after installation. It can be used to:

- Set up the modules/features
- Enable/disable modules
- Configure modules

Modularization and Customizations

Price Setting allows for simple customization by activating or deactivating existing functionalities.

- **Modularization** – Individual features of the accelerator are prepared as modules which can be **turned on or off**. This separation allows the package to stay fairly simple for small installations, while also allowing for more complex feature rich configurations.
- **Customizations** – Besides vast configuration options, you can also customize the accelerator, i.e. you can add additional items which were not there out-of-the-box. Generally, this means you will have to add some programming code, typically for:
  - Custom pricing strategy
  - Custom pricing logic, i.e. adjustments to the provided logics

Business User Reference (Price Setting)

- Pricing Strategy (Price Setting)
- Pricing Levels (Price Setting)
- Product Segmentation (Price Setting)

- Trade Levels (Price Setting)
- Prices Calculated in In/Dependent Price List
- Waterfall - Calculation Process (Price Setting)
- Fields Provided in Price List/Grid (Price Setting)
- Create Live Price Grid (Price Setting)
- Update Cost Plus Adjustment (Price Setting)
- Update Relevant Competition Data (Price Setting)
- Update Dependent Price List Level Adjustment (Price Setting)
- Troubleshoot Price List/Grid (Price Setting)

Pricing Strategy (Price Setting)

When calculating prices, you use some rules how to do it, a sequence of steps. For example, when calculating a price using the Cost Plus pricing strategy, you commonly need to retrieve the cost of the product, then find the margin uplift relevant to that product and then calculate the list price.

In Price Setting Accelerator, the Pricing Strategy represents the technical implementation of your pricing rules. When a price strategy is executed, it results in a price proposal for your product. When you change the strategy (update it), it results in a new price proposal.

You can have different price strategies for different Pricing Levels and Product Segments. The accelerator will calculate prices based on all of them and the one with the highest priority will end up as the final price proposal for your product.

Out-of-the-box Pricing Strategies

The accelerator contains several configurable pricing strategies out of the box:

- Competition Based – Min, Avg and Max
- Recommended Retail Price
- Cost Plus
- Price Increase
- Kit Pricing
- Anchor Pricing

Custom Pricing Strategies

In the case of unique, complex pricing rules that are not covered by the provided out-of-the-box strategies, the accelerator lets you define your own strategy and use it in the calculation process.

Cost Plus (Price Setting)

The Cost Plus pricing strategy calculates a product's selling price by adding a markup to its production cost, covering materials, labor, and overhead. This method ensures all costs are covered and a profit margin is achieved, making it popular in manufacturing, construction, and services for its simplicity and predictability.

Calculations

The Cost Plus strategy is calculated in one of the following ways:

- `Price = Cost + Plus` // absolute margin based – adding fixed amount of money to the cost
- `Price = Cost * ( 1 + Plus% )` // percentage based – adding percentage amount (based on the **cost**) to the cost. The Cost represents 100% and we add on top of it.

- `Price = Cost / ( 1 - Plus% )` // selling price based - adding percentage amount (based on the **price**) to the cost. The Price represents 100% and the Plus % is a percentage of that Price.

where:

- **Cost** (of the product) - Money value including currency.
  - Will be available in the price list/grid in the *Cost* field.
  - Is stored in a Product Extension table.
  - Is part of the Core PSP module (see PSP Cost Element).
- **Plus** (for the product) - Can be either percentage or absolute value.
  - Is stored in a Company Parameter tables "CostPlus" per Pricing Level and defined for a combination of Product Segments (see also Cost Plus Lookup).
- **Price**
  - The calculated result will be available in the price list/grid in the *Final Price* field.

Use Case Examples

Here are a few use cases for Cost Plus pricing strategy:

- **Traditional wholesale distribution model** - The Cost Plus pricing strategy can be used here to add a fixed or variable markup to the product cost to determine the selling price. This pricing strategy works well when the distributor wants to maintain a fixed profit margin regardless of the price changes in the market.
- **Manufacturing industry** - The Cost Plus pricing strategy is commonly used when products are manufactured to order, and the costs are variable based on the production quantity. The Cost Plus markup can be used to calculate the final selling price and ensure the desired profit margin. This pricing strategy can also help to manage inventory costs, as it considers the variable production costs.
- **Service-based businesses** - In service-based businesses such as consulting, legal, or accounting firms, the Cost Plus pricing strategy can be used to calculate the price of services based on the direct and indirect costs incurred in providing the service. The markup could be a fixed percentage or a fixed dollar amount.
- **Retail industry** - In a highly competitive retail industry, the Cost Plus pricing strategy can be used to calculate the selling price based on the direct and indirect costs, and the retailer's desired profit margin. This pricing strategy allows retailers to remain competitive in the market while maintaining a healthy profit margin.

Overall, the Cost Plus pricing strategy is suitable for businesses that require a straightforward pricing approach that ensures a desired profit margin while accounting for the product cost. It is a flexible pricing strategy that can be customized to suit various industries and business models.

Price Increase (Price Setting)

Price Increase strategy is used to (periodically) increase price over time - the increase factor is applied to the old price to get a new increased price.

Calculation

The Price Increase strategy is calculated in this way:

```
Price = Base Price + Price Increase
```

where:

- **Base Price** - Actual Price

    - Can be found in the price list/grid in the field *Actual Price.*
    - Is stored either in a price list / price grid or Product Extension table.
    - Actual Price is part of the Core PSP module (see PSP Actual Price Element).
- **Price Increase** – Can be either percentage or absolute value.
    - The *Increase* is stored in a Company Parameter tables "Price Increase" per Pricing Level and defined for a combination of Product Segments (see also Price Increase Lookup).
- **Price**
    - Can be found in the price list/grid in the field *Final Price.*

Use Case Examples

The Price Increase strategy, involving periodic adjustments to prices over time through either percentage-based or absolute value increases, is a common approach across various industries. This strategy is often used to address inflation, cost of production increases, market demand changes, or to reflect the added value of product improvements. Here are several common use cases for a Price Increase pricing strategy:

- **Consumer Goods and Retail** – Retailers and manufacturers of consumer goods frequently adjust their prices to reflect changes in production costs, such as raw materials and labor, or to keep up with inflation. For example, during periods of high inflation, businesses will periodically increase prices to maintain their profit margins.
- **Automotive** – Car manufacturers may increase the prices of new vehicles annually or with new model releases to reflect advancements in technology, safety features, and to cover increased costs of materials and labor.
- **Inflation-driven price increase** – Finally, you may want to increase prices based on inflation rates. In this case, you can use a custom pricing strategy that takes the inflation rate into consideration. You'll need to calculate the new price based on the current price and the inflation rate. The new price can then be passed on to the consumer to cover the additional costs due to inflation.

Anchor Pricing (Price Setting)

Anchor Pricing means that a product price is based on another product's price (Anchor) and an anchor factor by which we multiply this price. Note that this means that the price of the anchor product must be calculated using some other strategy.

Calculation

The formula for anchor pricing is:

Price = AnchorPrice * (1 + AnchorFactor%)

where:

- **Anchor Price** – Price of an "anchor" product. It is available in a price list/grid in either in the Final Price or Final List Price fields.
- **Anchor Factor** – Percentage value.
    - Stored in the Company Parameter "AnchorData" for each SKU, or in another table (the table is configurable).
- **Anchor Product** – The information which product is an anchor of which product is stored together with the information about Anchor Factor.

Limitations

- You cannot specify an anchor for an anchor, etc. Only one level of connection is possible.
- All connected products must be added to the same price list/grid.

- Because of technical limitations, Anchor Pricing should not be used in the same price list/grid with Kit Pricing.

References

- Engine used to calculate the price: Attribute Based Engine

Use Case Examples

Anchor pricing is a pricing strategy that uses the price of a related product as a reference point to set the price of another product. Here are some use cases for anchor pricing strategy:

1. **Premium Pricing** – If your company offers both premium and standard products in the same category, you can use anchor pricing to set premium pricing by setting the premium product price higher than the standard product price. The anchor factor for premium products could be a percentage value over the standard product price.
2. **Compatibility Pricing** – If your company has a product line that includes multiple versions with different features and prices, you can use anchor pricing to set the prices for the versions with fewer features. The anchor product would be the version with all the features, and the anchor factor for the version with fewer features would be a percentage value below that of the anchor product.

Recommended Retail Price (Price Setting)

In this strategy, the price is actually not calculated, but rather read from a table.

Calculations

The price:

- Will be available in the price list/grid in the Final (List) Price.
- Can be read from either a Product Extension or Company Parameter table.

References

- Prices in this strategy are retrieved using Lookup Engine.

Use Case Examples

Here are a few use cases for the Recommended Retail Price (RRP) pricing strategy:

1. **Distribution and wholesale pricing** – Many distributors and wholesalers have agreements with manufacturers and suppliers to set the RRP for their products. This pricing strategy allows them to automatically calculate and apply the RRP to their product offerings. By setting the RRP at the supplier or manufacturer level, distributors and wholesalers can ensure consistent pricing across their entire product catalog.
2. **New product introductions** – When introducing new products to the market, manufacturers and suppliers may want to base their initial pricing strategy on RRP. This allows them to set a starting point for their pricing and adjust it over time based on feedback from the market and competitors. The RRP pricing strategy can help manufacturers and suppliers ensure they are staying competitive while still maintaining consistent profit margins.

Overall, the RRP pricing strategy is a useful tool for businesses looking to automate their pricing processes based on external recommendations or agreements. It can be particularly effective in industries where distribution, wholesale, and retail pricing is common, allowing businesses to streamline their pricing processes while still maintaining consistency and competitiveness.

Competition Based (Price Setting)

Competition-based pricing strategy involves setting the price of a product or service based on what the competition is charging. This strategy is particularly useful in markets with high competition and similar products.

Calculations

When a price is based on competition, it is calculated in 3 steps:

1. Find the competitor's price (in the competition table) or calculate it.
2. Perform a margin check.
3. Reposition the price (i.e. modify the competition price by a percentage or absolute value).

1. Find Competitor's Price

Competitor's price can be selected based on one of two approaches (only one of them can be used at a time):

- **Competitor Position** – Selects one competition record to align the price with.
  - **min / max** – Selects a minimum/maximum available competitor price.
  - **min + X / max - Y** – Selects minimum/maximum competitor price position and adjusts it by a given value. For example "min + 1" means the 2nd lowest competition price.
  - **10%, 50%, 70%** – Select the target competitor based on the provided percentage. The formula for the calculation is: `TargetCompetitorPosition = NumberOfCompetitors * Percentage`. For example, 10% means when you would take all the competition prices, and put them to a row sorted by size, you would pick the one which is on the 10% position.
- **Price Position** – Select a price at the given percentage point. A value of 0% matches the lowest competition price and value of 100% matches the highest competition price. The formula for calculation is: `Price = CompetitorMinPrice + (CompetitorMaxPrice - CompetitorMinPrice) * Percentage`.
  - Note that the calculated price does not need to exist in the competition data, it is only derived from them.
  - Limitation: The Competitor Name is not known in this case, as the price is derived, and not directly retrieved from a competitor record.

The price list/grid will contain the Competition Data and Relevant Competition Data fields showing the competition data used for the row.

2. Margin Check

The engine supports Force Margin Check to verify that the selected competitor price is affordable. You can set values "Yes/No" in the table to turn the functionality on or off.

If the new competitor price is affordable after applying Force Margin Check, the engine will find the corresponding competitor name again and calculate the total of skipped competitors counting from the old competitor price to the new one. Finding the corresponding competitor name is not relevant for the "Price Position" mode. In such case, the lowest affordable price will override the current price if in range of available prices.

3. Reposition the Price

Then you can additionally "reposition" the price by using:

- **Percentage** - Modifies the price by a provided percentage. To make the price 5% cheaper, you use `-5%`; the same applies for a positive adjustment.
- **Absolute value** - Modifies the price by an absolute value. To make the price 10 units cheaper, you use `-10`; the same applies for a positive adjustment.

The result price will be in the price list/grid in Final (List) Price.

Reference

- This strategy requires the PSP Product Competition Module.
- This strategy is calculated using Competition Engine.

Use Case Examples

Use cases for implementing a competition-based pricing strategy can include:

E-commerce Retail - An e-commerce retailer selling consumer electronics uses competition-based pricing to adjust the prices of their products in real-time. By monitoring the prices of identical or similar products on competitor websites, the retailer can adjust their prices to be slightly lower, matching, or slightly higher depending on their value proposition and brand positioning. This strategy helps the retailer stay competitive, attract price-sensitive customers, and increase market share in a highly competitive online marketplace.

The key to success with competition-based pricing is not just to set prices in relation to competitors but also to continuously monitor market dynamics and adjust pricing strategies as needed to maintain competitive advantage and meet business objectives.

Kit Pricing (Price Setting)

Kit Pricing is a pricing strategy that allows companies to accurately calculate the price of complex products based on their individual components.

Calculation

Kit Pricing means that a price for a given product is based on subcomponents and quantities defined in a Bill of Material (BoM) data table.

A Kit Price is a sum of all of its subcomponent prices multiplied by provided quantities. There is no limit on how many levels of "subcomponent of subcomponent" are defined. If more than one level is present, we sum all the prices "at the bottom of the tree" using proper quantity factors.

Limitations

- All connected products must be added to the same price list/grid.
- Because of technical limitations, Kit Pricing should not be used in the same price list/grid with Anchor Pricing.

References

- Engine used to calculate the prices: Kit Engine

Use Case Examples

Here are a few use cases for Kit Pricing pricing strategy:

- **Electronics Components Manufacturing** – In this scenario, a company manufacturing electronic components has a Bill of Material (BoM) for each of its products, which lists all the components required to assemble that particular product. Using Kit Pricing, the company can calculate the price of the final product by summing up the prices of all its subcomponents, multiplied by their respective quantities. This not only simplifies the process of calculating the price of complex products but also allows the company to offer discounts and promotions on certain components, thus reducing their overall cost of production.
- **Furniture Manufacturing** – A furniture manufacturer sells various products such as beds, sofas, chairs, etc., each with its own set of components. The price of these products is determined by calculating the total cost of all its subcomponents, including the frame, cushion, and covers. By using Kit Pricing, the company can easily adjust the prices of its products based on factors like the material used, the size of the furniture item, and other such parameters.
- **Automotive Industry** – In the automotive industry, Kit Pricing is commonly used to calculate the price of a car by summing up the costs of all its constituent parts, such as the engine, transmission, body panels, and other components. By breaking down the price in this manner, the manufacturer can easily adjust the final price based on factors like material costs, production quantities, and demand.

Overall, Kit Pricing is a powerful pricing strategy that allows companies to accurately calculate the price of complex products based on their individual components, while also providing flexibility to adjust pricing based on various factors.

Pricing Levels (Price Setting)

You may need to calculate prices for scenarios such as:

- Standalone price list or independent/dependent price lists
- National & state prices
- Global & local/country prices
- Regional & country & channels prices
- Global & regions & countries & shop prices

Using the Price Setting Accelerator terms, you need to calculate prices on different Pricing Levels. Prices for various Pricing Levels are then calculated in separate price grids/lists.

Company Parameter Values: USER: 900 Price List Dependencies (PS)

Dependency Level Name ▲ 2	Depends On ▼ 1	Source Type	Source ID	Dimension	Currency	Is complete	ISO Code	SalesOrg
Search...	Search...	Sele...	Searc...	Searc...	Searc...	Sele...	Searc...	Searc...
Global	Independent	*	*		EUR	No	GL	SO20
France	Global	PL	1056	Country	EUR	No		SO25
Germany	Global	PL	1056	Country	EUR	No	DE	SO24
Italy	Global	PL	1056	Country	EUR	No	IT	SO28
Spain	Global	PL	1056	Country	EUR	No	ES	SO27
United Kingdom	Global	PL	1056	Country	GBP	No	UK	SO30
United States of America	Global	PL	1056	Country	USD	No	US	SO26

The prices on lower levels can depend on prices from higher levels, i.e. the lower level price can be derived from the higher level price.

- **Independent level** - Prices on this level are not derived from other levels. You can think of it as the "top" pricing level.
- **Dependent level** - Prices of this level are derived from another parent level. You can think of them as "lower" levels of the dependency hierarchy.

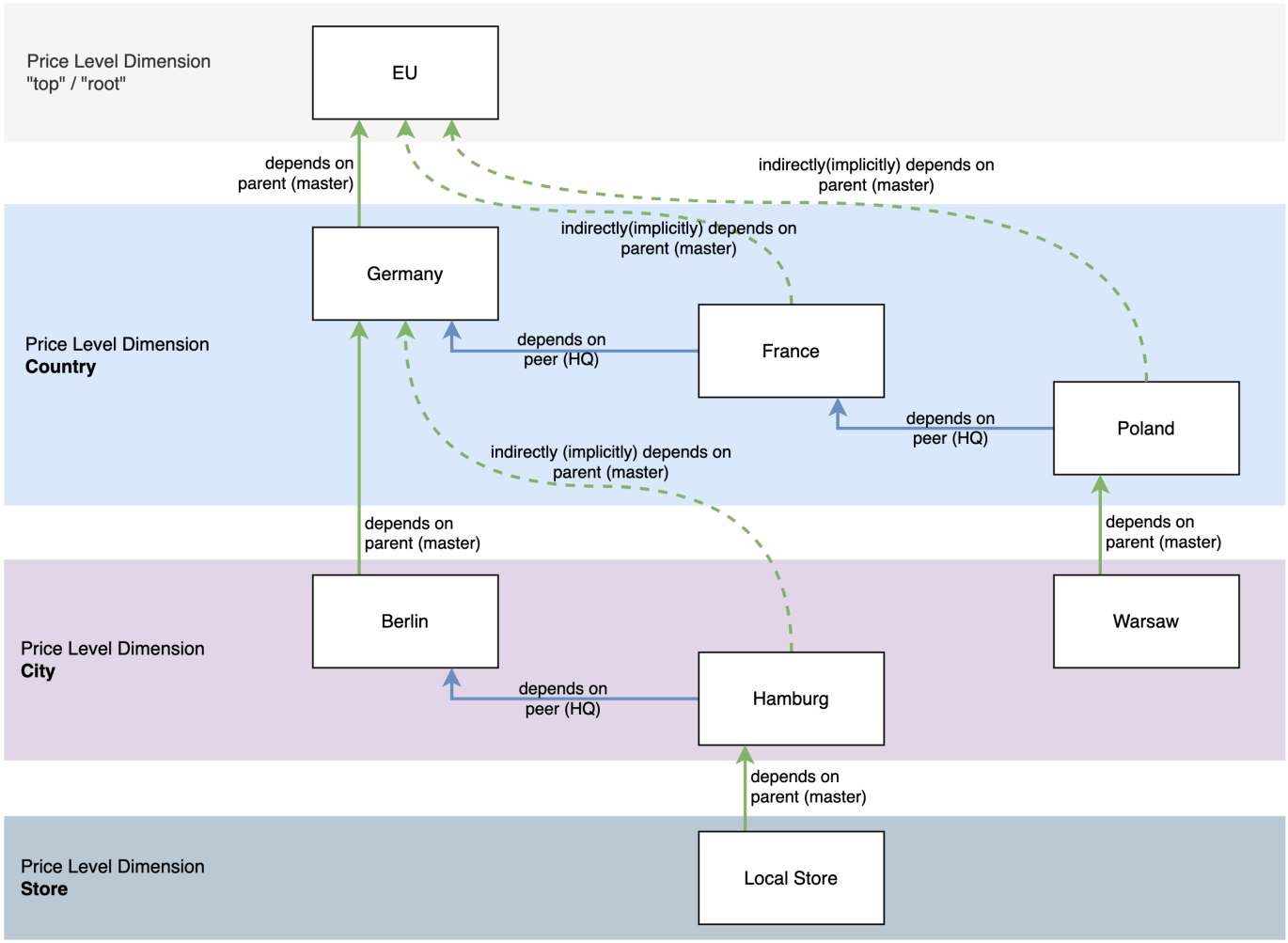
> ⚠ The business decision which pricing levels to use and how they depend on each other is very important, and must be done before installation of the accelerator because it has direct impact on how data tables are created and named.
>
> However, during installation you need to upload only the fields Dependency Level Name, Depends On and Dimension. Other fields can be resolved later during configuration.

The word "Dependency" is sometimes used in the meaning of "Level", e.g. in the term "Dependency Mapping".

Pricing Levels and their dependencies are also sometimes referred to as Pricing Structure, Dependency Structure or Dependency Configuration.

Pricing Level Dependencies

There are several types of dependencies between the pricing levels. These dependencies are used during prices calculation to find proper data for each product.



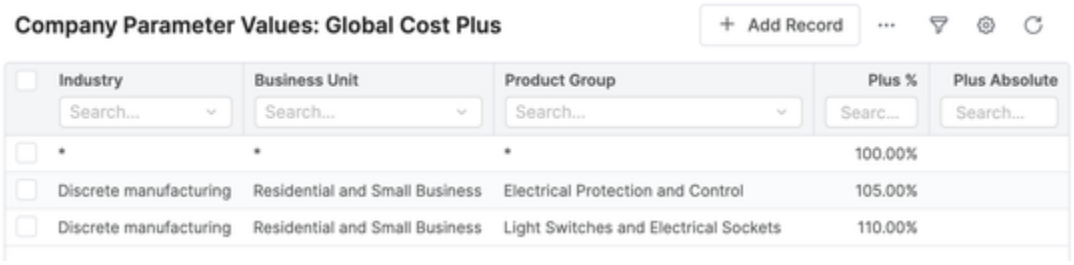
- **Independent** - "top"/"root" - Does not depend on anything.

- **Master/parent dependency** – Such dependency goes across dimensions. It influences data lookups and configuration – e.g. lower dependent levels can define only some of its specific data (different from parent), and then find the rest of the data on the parent level.
- **HQ (Head Quarter) / peer dependency** – Dependency between pricing levels within the same dimension.
- **Implicit/indirect dependency on master/parent** – Any pricing level which depends on the HQ pricing level also indirectly/implicitly depends on the master of that HQ level. This dependency is not explicitly configured in the pricing level definition table, but it influences the data and configuration lookups in the same way as the master dependency (with the exception of a product master price lookup).

Product Segmentation (Price Setting)

Product Segmentation is used to diversify your pricing decisions, for example:

- Pricing adjustments/factors (e.g. margin uplift in Cost+ method) can depend not only on the Pricing Levels, but also on a combination of several product attributes, e.g. Industry, Business Unit, Product Group.

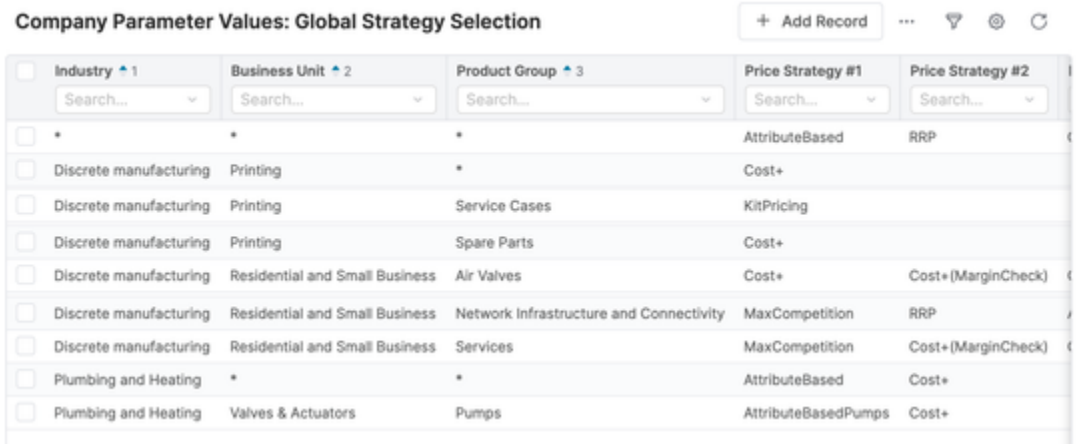


Company Parameter Values: Global Cost Plus

+ Add Record

Industry	Business Unit	Product Group	Plus %	Plus Absolute
Search...	Search...	Search...	Searc...	Search...
*	*	*	100.00%	
Discrete manufacturing	Residential and Small Business	Electrical Protection and Control	105.00%	
Discrete manufacturing	Residential and Small Business	Light Switches and Electrical Sockets	110.00%	

- Pricing Strategies can be defined differently for different product segments. You can define some high-level pricing strategy and specify special product segments on a more granular level where needed.



Company Parameter Values: Global Strategy Selection

+ Add Record

Industry 1	Business Unit 2	Product Group 3	Price Strategy #1	Price Strategy #2
Search...	Search...	Search...	Search...	Search...
*	*	*	AttributeBased	RRP
Discrete manufacturing	Printing	*	Cost+	
Discrete manufacturing	Printing	Service Cases	KitPricing	
Discrete manufacturing	Printing	Spare Parts	Cost+	
Discrete manufacturing	Residential and Small Business	Air Valves	Cost+	Cost+(MarginCheck)
Discrete manufacturing	Residential and Small Business	Network Infrastructure and Connectivity	MaxCompetition	RRP
Discrete manufacturing	Residential and Small Business	Services	MaxCompetition	Cost+(MarginCheck)
Plumbing and Heating	*	*	AttributeBased	Cost+
Plumbing and Heating	Valves & Actuators	Pumps	AttributeBasedPumps	Cost+

Product Segment is described by combination of values of several product attributes, e.g. Industry, Business Unit, Product Group. Those product attributes are not necessarily hierarchical, but when defining the product segments, we needed to have them in certain order, e.g. Industry > Business Unit > Product Group.

A product segment can be either very narrow (e.g. products with industry A, business unit B, product group C), but also more broad (e.g. products with industry A, business unit “any”, product group “any”).

> The product segmentation must be decided very early in the project because you need to know it already during installation of the accelerator. It directly impacts fields of data tables, so any later change requires a special process.

Product segmentation is also known as: segmented product portfolio, product attribute hierarchy.

Trade Levels (Price Setting)

Prices can be set in two ways: either as the List Price (Gross price) alone or as both Gross and Net prices. The choice depends on the industry's standard.

When both Gross and Net prices are used, businesses can apply an additional discount to the Gross price to determine the Net price. This discount is customizable at various pricing levels.

For a detailed guide on how to calculate these prices, visit the documentation page: Price Calculation Process.

Note, that the documentation refers to "Trade Levels" also as "Price Setting Level".

For instructions on setting up Trade Levels, refer to the PSP Net Price Module documentation.

Prices Calculated in In/Dependent Price List

The in/dependent price list calculates several prices and places them to the Prices field on the price list line item. The Final Price is then selected from the list available in the Prices field.

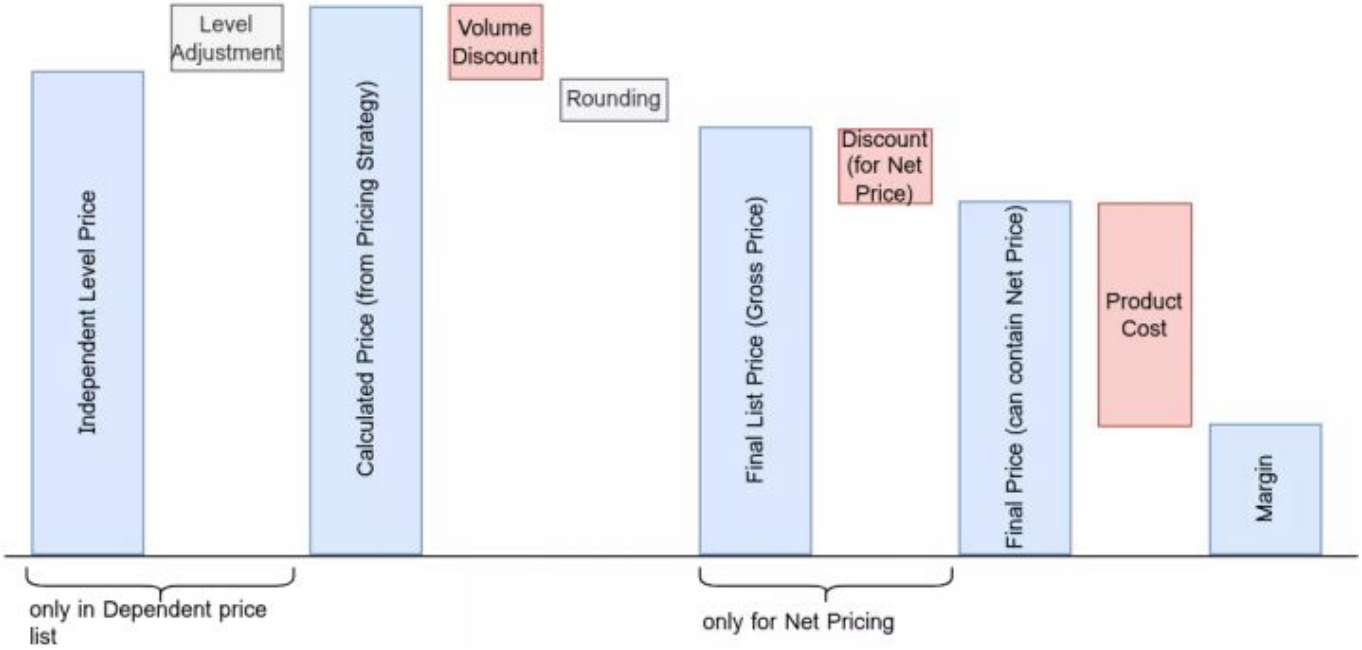
- **Exception Price** - If you need to "override" a price (set a price regardless of the regular calculation) of a specific product, you can set up an Exception.
- **Adjusted Final Price** - Final Price from the parent level; an adjustment is applied to it.
- (Base) **Pricing Strategy #N** - You can have several strategies set up for the price calculations. The system will calculate the price from all those strategies and apply to them waterfall calculations to get to the Final Price.
- (Base) **Pricing Strategy #N from parent level** - Copy of prices calculated by strategies on the parent level price list.

Some of those prices are available only in Dependent price list. Here is the comparison:

Prices \ Price List Level	Parent	Dependent
Exception Price	Yes	Yes
Adjusted Final Price from parent level		Yes
Base Strategy #N	Yes	Yes
Strategy #N	Yes	Yes
Adjusted Base Strategy #N from parent level		Yes
Adjusted Strategy #N from parent level		Yes

Waterfall - Calculation Process (Price Setting)

The Price Setting Package calculation process consists of several steps.



1. Parent Level Price (only in Dependent price lists) – A price coming from the "parent" price list is copied.
2. Level Adjustment – An adjustment is applied to the price coming from the parent price list.
3. Calculated Price – The Pricing Strategy (either the out-of-the-box or custom one) calculates the price.
   - In the dependent price list, you will also find the prices from the parent price list adjusted by an level adjustment.
4. Volume discount is applied (only when you use Volume Discounting).
5. Rounding is applied (either standard rounding to 2 decimal places or special one configured in Price Setting Package).
6. Final List Price is calculated. When using Net Pricing, this price represents the Gross Price.
7. Discount is applied to the Final List Price. This is used only for Net Pricing.
8. Final Price is calculated. When using Net Pricing, this price represents the Net Price.
9. Product cost is deducted.
10. Margin is calculated.

Fields Provided in Price List/Grid (Price Setting)

When you open the detail of the (in)dependent price list/grid calculated by Price Setting Accelerator, you can see many fields – some of them out-of-the-box from Pricefx and many of them calculated by the Price Setting Accelerator.

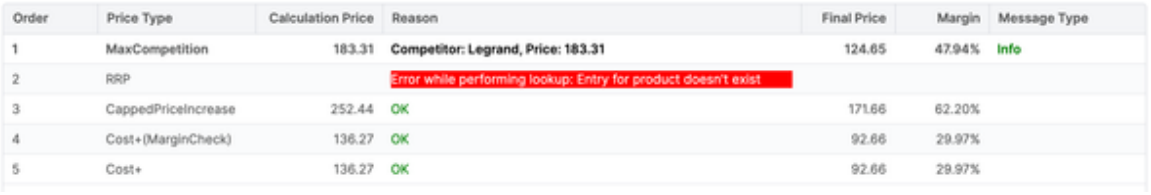
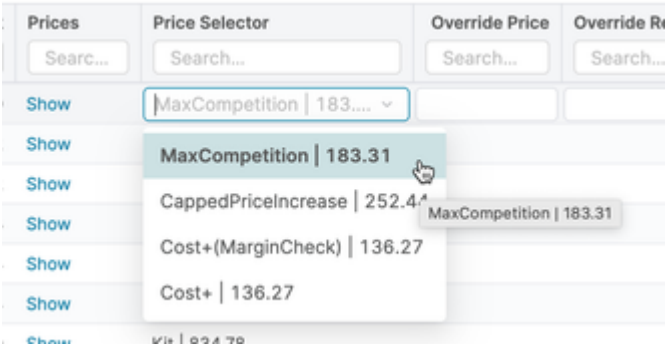
Description of the out of the box fields in LPG is available at LPG Detail.

Here you can find a list of fields specific for the Price Setting Accelerator. Note that some of them are present/calculated only when a certain PSP module is switched on, or if some other condition applies. The order is based on their appearance from left to right (in case it was not manipulated by Preferences).

			Description

Field Label	Type	PSP Module	
Changes		Price Flexibility	Changes from Monitor. Displayed only when the SKU is a target of Price Flexibility.
Actual Price Lookup	$		Actual/current price of the product – either the last approved price, or the new price (if no approved price is available yet).
Dependency Level Currency	String		Currency of Dependency Level. Used as a base currency for calculations of the money fields of this item.
Parent Level Decision	String		(Available only in Dependent price lists. This value is taken from the Parent price list.) Reason for the final price value. If the price was overridden by user, this decision is written by the user. Otherwise, it will be the name of the price strategy used for calculation of the result.
Parent Level Price	$		(Available only in Dependent price lists.) Final Price of the product as calculated in the Parent price list.

Dependent Level Adjustment	%		(Available only in Dependent price lists.) How much % will be added to the Parent level price to get the Dependent price.
Advanced Costs		Advanced Costs	
Cost	$		Cost of the product.
Competition Data	Matrix	Product Competition	The list of competition data obtained from the Competition Data master table. The list contains details of the competitors, their prices, the types of prices and when the price is valid from. It will be displayed for products where competition data is available.
Relevant Competition Data	Matrix	Product Competition	The list of (only) relevant/target competitors data obtained from the Competition Data master table.
Discount	%	Net Price	Discount (%) to be applied when transitioning from a Gross to Net price.
Volume Disc	%		

ount			
Minimum Margin Price	$	Price Checks	Price calculated based on Minimum Margin (%), Cost and Discount (%), depending if the pricing mode is Gross or Net.
Stock	#		Number of items available on Stock. This value is also used to calculate Stock Cover Days.
Prices	Matrix		Shows a list of all prices calculated for the product, including the info about pricing strategy used, and the error messages raised during the calculation. Unlike the Price Selector, this list shows also the strategies which failed to calculate the price. This field is not editable for the user.  Order / Price Type / Calculation Price / Reason / Final Price / Margin / Message Type 1 / MaxCompetition / 183.31 / Competitor: Legrand, Price: 183.31 / 124.65 / 47.94% / Info 2 / RRP / Error while performing lookup: Entry for product doesn't exist 3 / CappedPriceIncrease / 252.44 / OK / 171.66 / 62.20% 4 / Cost+(MarginCheck) / 136.27 / OK / 92.66 / 29.97% 5 / Cost+ / 136.27 / OK / 92.66 / 29.97%
Price Selector		Override	Drop-down menu showing successfully calculated prices for the product, including the strategy used to calculate those prices. The price with the highest priority (based on strategies) is pre-selected, but the user can select another price from the list.  Prices / Price Selector / Override Price / Override R Search... / Search... / Search... / Search... Show / MaxCompetition | 183.... MaxCompetition | 183.31 CappedPriceIncrease | 252.44 Cost+(MarginCheck) | 136.27 Cost+ | 136.27 MaxCompetition | 183.31
Override Price	$	Override	Field to enter the price manually, in case the user needs to override the price.
Override Reason	String	Override	Field to enter the comment/reason for override. Done by the reviewer who does the price override.

Exceptions	String	Override	Information about the override values (if any).
Final List Price	$	Net Price	Strategy calculated price. This is used as Result Price in case of Net Price calculations.
Net Price	$	Net Price	Final List Price with a Discount applied. When “Net pricing” mode is used, this price represents the Final price.
Final Price	$		Returns either Net or Gross price, depending on the Net Price Level. When calculating Gross Prices, this is used as Result Price of the product. In case of Net Prices, this contains a copy of the Net Price value.
Price Decision	String		Reason for the final price value. If the price was overridden by user, this decision is written by user. Otherwise it will be the name of the price strategy used for calculation of the result.
Old Margin	%		
New Margin	%		Margin % based on the current Final Price and Product Cost.
Margin Flag	String	Price Checks	Shows either Critical, High or Medium, depending on what margin falls into what range.
Minimum Margin	%	Price Checks	The minimum margin specified for the product.

Margin Breakeven Volume	#		"Margin Breakeven Volume" is the level of sales or production volume at which a business covers all of its costs and reaches a point where it neither makes a profit nor incurs a loss. In other words, it is the point at which a company's total revenue equals its total costs, including both fixed and variable costs, resulting in a breakeven situation where there is no net profit or loss. See also article about Volume Breakdown.
Revenue Breakeven Volume	#		"Revenue Breakeven Volume" is the level of sales or production volume at which a business generates enough revenue to cover its total costs, including both fixed and variable costs. At this point, the company does not make a profit, but it also does not incur a loss. In other words, it is the point where total revenue equals total costs, resulting in a breakeven situation. See also article about Volume Breakdown.
Adjusted Price Corridor	%	Price Checks	(Available only in Dependent price lists.) Deviation between the parent adjusted price and final list price.
List Price Corridor	%	Price Checks	(Available only in Dependent price lists.) Deviation between the parent price and final list price.
Sales Volume YTD	#	Transaction	Volume from this year, i.e. the sum of sales volume from the beginning of the current year to the calculation date. "Sales Volume YTD" stands for "Sales Volume Year-to-Date." It represents the total quantity or volume of products or services a business has sold or produced from the beginning of the current calendar year up to the present date. It helps track performance and sales trends over the course of the year, providing insights into business operations and progress toward annual targets.

Turnover YTD	$	Transaction	Turnover from this year, i.e. the sum of turnover from the beginning of the current year to the calculation date. "Turnover YTD" stands for "Turnover Year-to-Date." It refers to the total revenue or sales a business has generated from the beginning of the current calendar year up to the present date. This metric is used to assess a company's financial performance and sales trends for the year, helping with year-to-date comparisons and goal tracking.
Last Year Sales Volume	#	Transaction	Volume from last year, i.e. the sum of sales volume from the whole last year. "Last Year Sales Volume" refers to the total quantity or volume of products or services a business sold or produced during the previous calendar year. It is a historical measure used for comparisons and analysis of sales performance between the current year and the preceding year, providing insights into growth or changes in business operations over that time frame.
Last Year Turnover	$	Transaction	The sum of turnover from the whole last year. "Last Year Turnover" represents the total revenue or sales a business generated during the previous calendar year. It serves as a historical reference point for assessing the company's past financial performance and is often used for year-over-year comparisons to analyze changes or growth in revenue over time.
Last Period Volume	#	Transaction	The sum of sales volume in a specified time range in the past. "Last Period Volume" typically refers to the quantity or volume of products or services a business sold or produced during the most recent reporting period, such as a month, quarter, or year. It provides a historical reference point for assessing past production or sales performance and can be used for comparison with current or future volume figures to analyze trends or changes in business operations.
Last Period Turnover	$	Transaction	The sum of turnover in a specified time range in the past. "Last Period Turnover" refers to the total revenue or sales a business generated during the most recent reporting period, typically a month, quarter, or year. It serves as a historical reference point for assessing a company's past performance and can be used for comparison with current or future sales figures to analyze growth or changes in revenue.
Sales Volume Fore	#	Transaction	A "Sales Volume Forecast" is a prediction or estimate of the quantity of products or services a business anticipates selling within a specific future period, such as a month, quarter, or year. It is a crucial tool for planning inventory, production, and resource allocation to meet expected customer demand.

cast			
TurnoverForecast	$	Transaction	A "Turnover Forecast" is a financial projection that estimates the total revenue or sales a business expects to generate within a specified future period, typically a month, quarter, or year. It helps businesses plan and make informed decisions regarding sales strategies, resource allocation, and financial performance expectations.
DaysOfCover	#	Transaction	"Days of cover" is a financial metric used to assess how long a company's current resources or inventory will last at its current rate of consumption or sales. It is primarily used in the context of inventory management and working capital analysis. The formula for calculating days of cover is: Days of Cover = (Current Inventory or Resources) / (Average Daily Consumption or Sales)
PriceChangeEffect	$		Shows how big impact the price change will have. This takes into account the Final Price, Sales Volume Forecast and the previous List Price (from Actual Price).
DataLookups	Matrix		Which PSP data tables were used for calculation. Useful for finding problems during price calculations.
ConfigLookups	Matrix		Which PSP configuration tables were used for calculation. Useful for finding problems during price calculations.
Warningsanderrors	Matrix		List of issues and exceptions found during calculation of this line.

Create Live Price Grid (Price Setting)

This article describes how to create a new in/dependent Live Price Grid using the out-of-the-box Price Setting Package features.

Only the Price Setting Package specific steps are described here. For general steps see How to Create and Configure New LPG.

Steps:

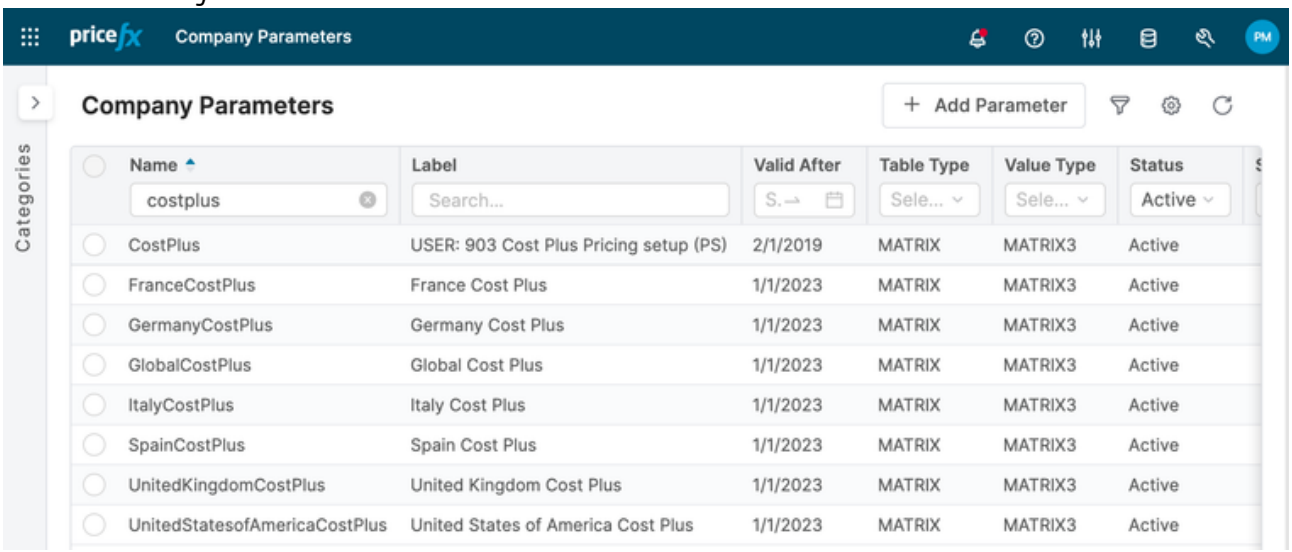
1. Navigate to **App › Price Setting › Live Price Grids**.
2. Click **New Price Grid** and select *(default)* type. Note that in customer projects there can be prepared several types with certain presets.
   a. Define the Live Price Grid:
      - **Label** – Provide a name which will be useful/meaningful for the business user. The name can be based on the products type and Pricing Level for which the prices are calculated. For example, the label "Discrete Manufacturing - Global Pricelist" suggests that the price grid contains prices for "Discrete Manufacturing" industry and "Global" could be a name of an parent Pricing Level. In case of a "dependent" price list, the name could represent a region or a country.
        Note that you will select the actual Pricing Level in the dropdown on the *Set Parameter* tab.
      - **Type**:
        - SIMPLE – If you do not use the Volume break down feature.
        - MATRIX – If you want to use the out-of-the-box Volume Breakdown feature, and calculate the prices not only per SKU, but for combination of SKU and volume tier. Additionally, you also have to specify:
          - Matrix Logic – VolumeBreakdownMatrixLogic
          - Matrix Logic Element – Volumes
   b. Select **Products** for which you need to calculate the prices (for example, only products from a specific industry).
   c. Set **Parameters**:
      - **Allow column type change** – Should be checked, so that the price list columns can accommodate the changes in setup of the accelerator.
      - **Default pricing logic**:
        - For calculation of parent (e.g. "global") price lists – Choose *ParentPriceListLogic* (this logic is provided by Price Setting Package). Note that in your customer project this could have been extended or customized and so it may have a different name.
        - For calculation of dependent (e.g. "country", "region", etc.) price lists – Choose *DependentPriceListLogic*.
      - **Result Price** – Select *Final Price* (or *Final List Price*). This depends on whether you use Net Pricing – see Calculations.
      - **In/Dependent Level Name** – The actual name and available values of this input field depend on whether you set up this price list as parent or dependent. Choose from available in /dependent Pricing Levels. The choice of Pricing Level should match the *Label* you gave to the price list. Available options depend on the Pricing Levels defined for the accelerator.
      - **Output Elements** – Select all elements you prefer to have on each line of the price list. Review the list of fields available from the Price Setting Package and choose those you may need.
      - **Default fields** – Hide those fields you do not need to see. For example, you can hide all of them, if you need to have only the fields provided by the accelerator.
3. **Save** the Live Price Grid definition.
4. **Recalculate** the whole price grid.
   a. In the list of Live Price Grids, select the one you have just created, and click the **Calculate** button.

5. Hide the Currency column.
   The reason is that this field gets the information from Product Master and it is not needed here – the currency of the calculated price is available in the column Dependency Level Currency (see also Fields Provided in Price List/Grid (Price Setting)).
   - To hide it, open the price grid, hide the Currency column and save the Preference as default one for all users.
6. Review the calculated price grid and eventually hide other columns which are not needed to be displayed for your business.

Update Cost Plus Adjustment (Price Setting)

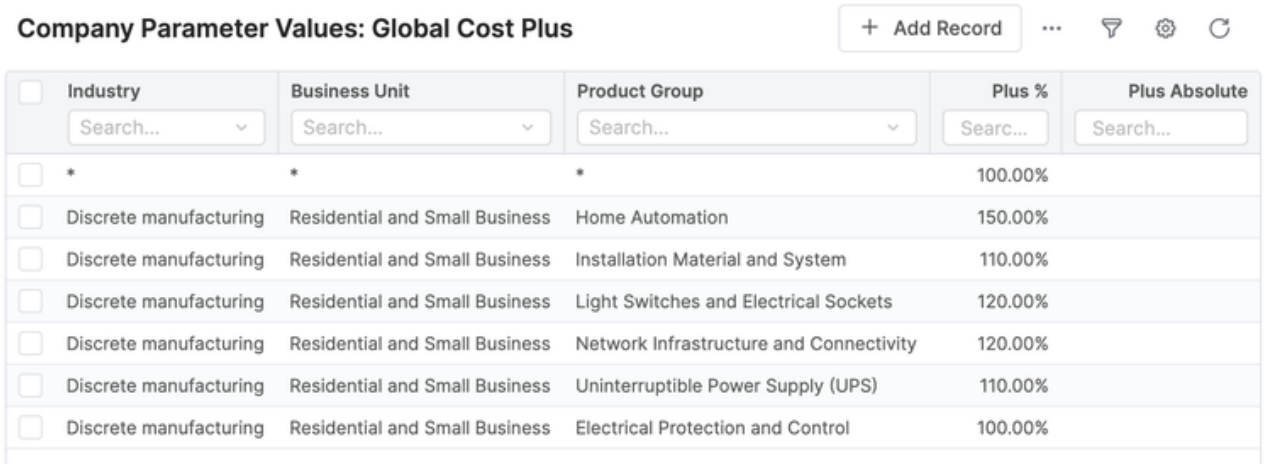
This article describes how to update the adjustment for the out-of-the-box Cost Plus strategy.

1. Navigate to the Company Parameters page, and search for tables with name ending with "CostPlus". They hold the CostPlus factors.



2. Select a table to modify. Decide for which Pricing Level you want to set up the adjustments. You can either set it for:
   - very specific Pricing Level (for example in "GermanyCostPlus),
   - or for some of the parent Pricing Levels (for example in "GlobalCostPlus),
   - or eventually for all in the table "CostPlus" (called also fallback).

   Note: When the system looks for the actual value of the adjustment, it will start from the table specific for the price list/grid Pricing Level (dependency name) and if not found there, it will traverse up the Pricing Levels via their parents, and if not found, it will try to get it from the CostPlus table.
3. Define the Plus adjustment (either % or a particular money value, depending on what is used for your business) for a combination of Product Segmentations. If you are curious how your Cost+ is set up, you can review the article https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/4775149631/Configure+Cost+Plus+Pricing+Strategy+Price+Setting#Select-Formula-for-Calculation.

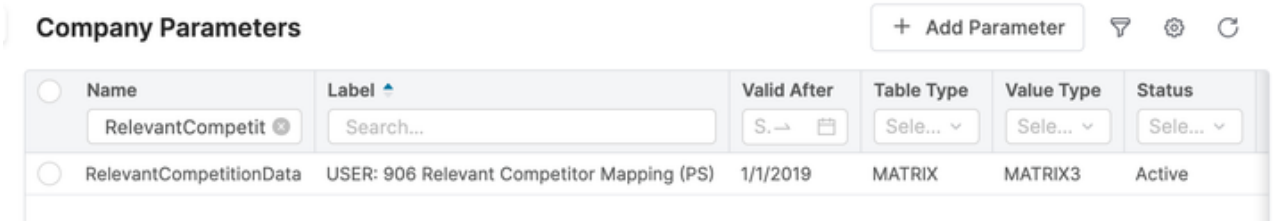


4. Recalculate the price list/grid and ensure your adjustment changes are used.

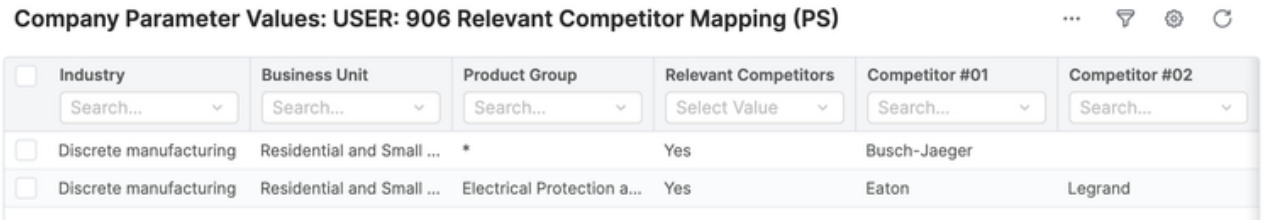
Update Relevant Competition Data (Price Setting)

This article describes how to update the relevant/target competition data for the out-of-the-box Competition Based pricing strategy.

1. Navigate to the Company Parameters page, and search for tables with name ending with "RelevantCompetitionData". They hold the relevant competitors which you want to take into account. Note that it does not have to be set up.



2. Select the table to modify. Decide for which Pricing Level you want to set up the relevant competitor. You can either set it for:
   a. very specific Pricing Level (for example in "GermanyRelevantCompetitionData"),
   b. or for some of the parent Pricing Levels (for example in "GlobalRelevantCompetitionData"),
   c. or eventually for all in the able "RelevantCompetitionData" (called also fallback).
3. Define the relevant competitors for a combination of Product Segmentations.

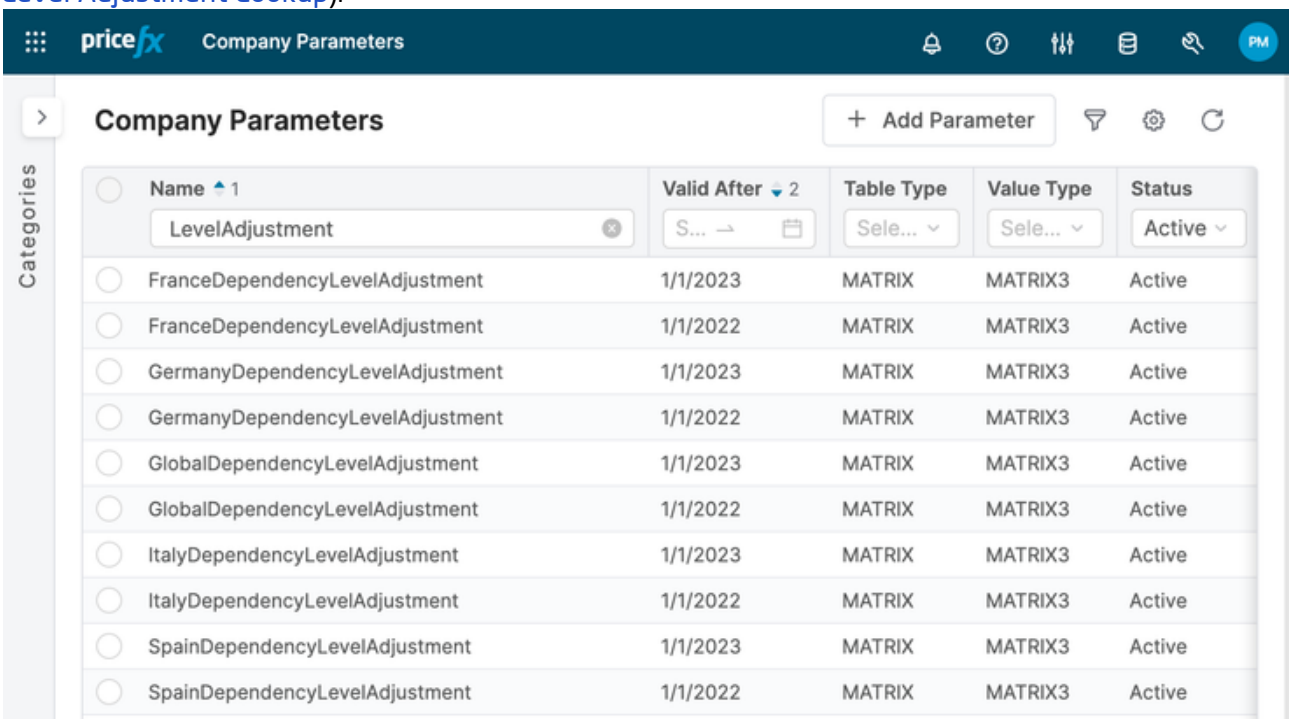


4. Recalculate the price list/grid and ensure your adjustment changes are used.

Update Dependent Price List Level Adjustment (Price Setting)

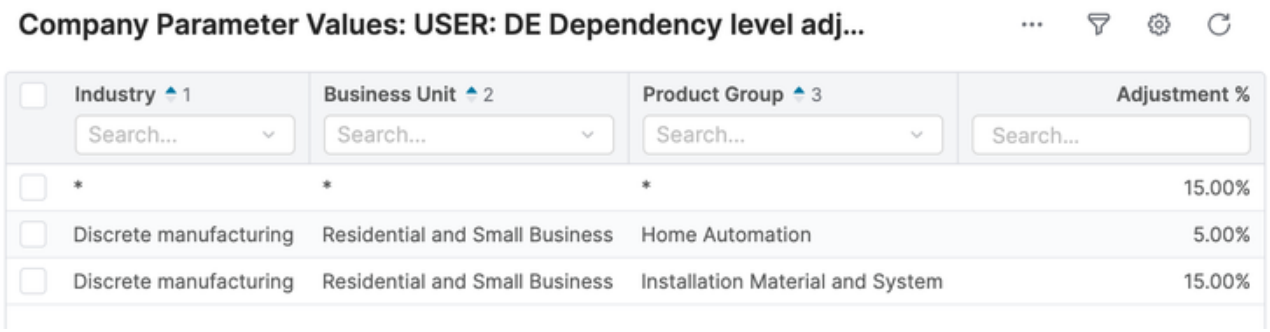
This article describes how to update the level adjustment used for calculating dependent price list/grid prices where the prices coming from the parent price list are adjusted by a % factor.

1. Navigate to the Company Parameters page, and search for tables with a name ending with "DependencyLevelAdjustment". They hold the Level Adjustment factors. (See also Dependency Level Adjustment Lookup).



2. Select a table to modify. Decide for which Pricing Level you want to set up the adjustments. You can either set it for:
   - very specific Pricing Level (for example in "FranceDependencyLevelAdjustment),
   - or for some of the parent Pricing Levels (for example in "GlobalDependencyLevelAdjustment"),
   - or eventually for all in the table "DependencyLevelAdjustment" (called also fallback).

   Note: When the system looks for the actual value of the adjustment, it will start from the table specific for the price list/grid Pricing Level (dependency name) and if not found there, it will traverse up the Pricing Levels via their parents, and if not found, it will try to get it from the "DependencyLevelAdjustment" table.
3. Define the Adjustment % for a combination of Product Segmentations.



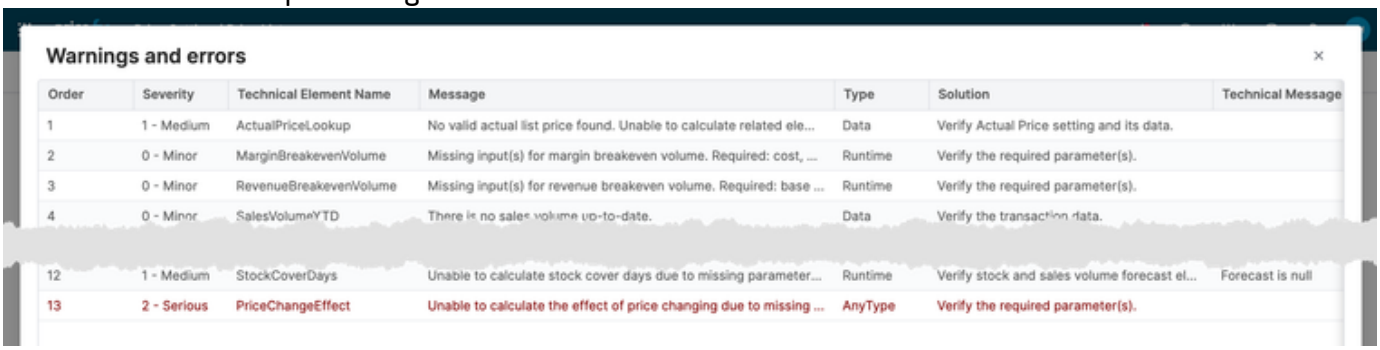
4. Recalculate the price list/grid and ensure your adjustment changes are used.

Troubleshoot Price List/Grid (Price Setting)

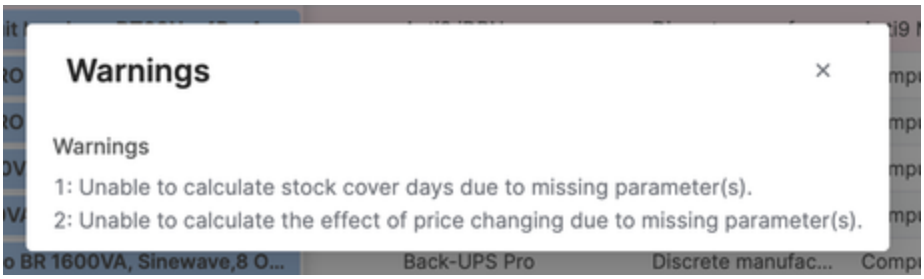
If the Prices field is empty or encountering issues, it is time to troubleshoot. This guide explains how to identify calculation problems in price lists/grids managed by the Price Setting Package.

Review Warnings and Errors

When a problem occurs during calculation of a price list line, all the error messages are placed to *Warnings and errors* field in the price list/grid.



Note that some of them are also duplicated into the price list *Warnings* field, but there they do not have all the details.



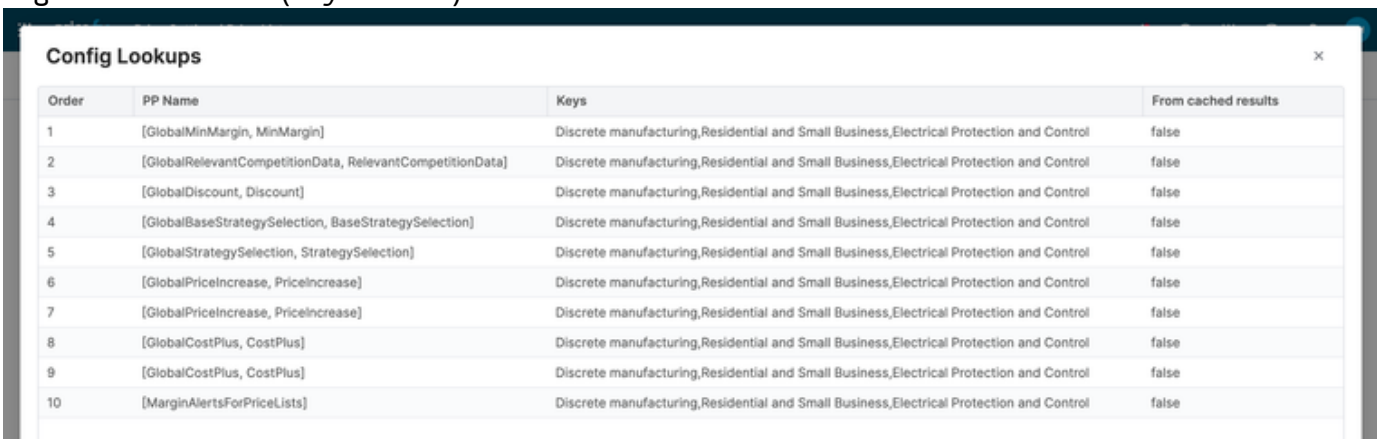
Review the whole message, including the suggested solution, to find out the cause of the problem. Note that some messages do not necessarily mean that there is a real problem, but are more of a notification.

1. **Order** – Review the messages from top to bottom because some minor issues at the beginning could cause bigger issues later and you could fix it by fixing the minor issue from the beginning. For example, a missing Cost value may not be a big problem in the general sense, but it could cause a failure of the Cost+ method calculation.
2. **Severity** – When the Severity is "3 - Fatal", the calculation cannot continue, and the process fails, so you do not even see the
3. **Message and Solution** – Always review the whole message (it can be long, but it is worth it) and possible solution. It can give you a good hint about what to do.
4. **Type** – Based on the Type of the problem, you can investigate further.
   - Data – There is a problem to find data. To find out which tables the accelerator tries to read, review also the "Data Lookups" field on the price list/grid item. It can give you an idea what tables and filters are used to look for data and it can help you to find the problem.
5. **Technical Element Name** – This could give you an indication of when (during calculation of which field) the problem happened. This name does not have to match exactly the label of the field visible in the price list/grid, but it can still give you some idea where the problem occurred.

Review Config Lookups

If you have doubts whether your configuration in Company Parameters is used during the calculation, you should review the contents of the *Config Lookups* field.

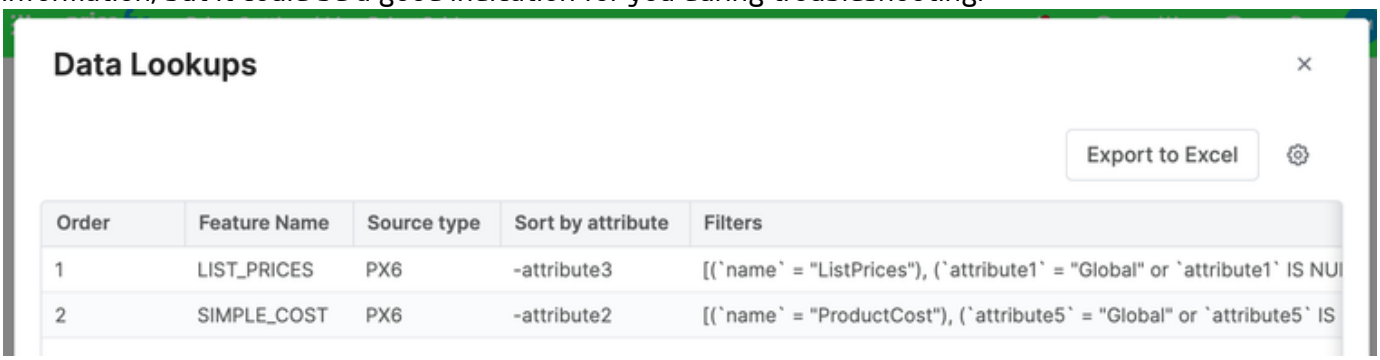
It shows you which tables (in the *PP Name* column) were searched, in which order, and for what Product Segmentation values (*Keys* column).



Order	PP Name	Keys	From cached results
1	[GlobalMinMargin, MinMargin]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
2	[GlobalRelevantCompetitionData, RelevantCompetitionData]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
3	[GlobalDiscount, Discount]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
4	[GlobalBaseStrategySelection, BaseStrategySelection]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
5	[GlobalStrategySelection, StrategySelection]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
6	[GlobalPriceIncrease, PriceIncrease]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
7	[GlobalPriceIncrease, PriceIncrease]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
8	[GlobalCostPlus, CostPlus]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
9	[GlobalCostPlus, CostPlus]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false
10	[MarginAlertsForPriceLists]	Discrete manufacturing,Residential and Small Business,Electrical Protection and Control	false

Review Data Lookups

Is warnings indicate that some data are missing, you can also review the Data Lookups field which will show you which data were searched, with what filters and sorting. This is more or less technical information, but it could be a good indication for you during troubleshooting.



Order	Feature Name	Source type	Sort by attribute	Filters
1	LIST_PRICES	PX6	-attribute3	[(`name` = "ListPrices"), (`attribute1` = "Global" or `attribute1` IS NUI
2	SIMPLE_COST	PX6	-attribute2	[(`name` = "ProductCost"), (`attribute5` = "Global" or `attribute5` IS

Admin User Reference (Price Setting)

As an administrator, you can find here information on deployment and upgrade of the package.

- Mandatory Data (Price Setting)
- Install Price Setting Accelerator
  - Installation Prerequisites (Price Setting)
  - Installation Steps (Price Setting)
  - Post-Installation Steps (Price Setting)
    - Review Installed Components (Price Settings)
    - Disable All Unnecessary Features (Price Setting)
    - Configure Core Elements Module (Price Setting)
    - Select Logics Available to Price List/Grid (Price Setting)

Mandatory Data (Price Setting)

This is a list of mandatory and optional data for the Price Setting Package. It helps you identify what data you need from the customer to set up and configure the package.

You can start with requirements for Core (Product Data) and at least mandatory data for one pricing strategy.

Type	Data	Fields	Mandatory for Use Case
Master Data	Products	• SKU • Field(s) to be used as Lookup Key	Core Technically SKU could be SKU, but this would probably make no sense – you could completely "skip" the lookup key concept by defining SKU as the only Lookup Key Dimension and configure everything with the "*" value.
	Product Cost	Mandatory: • SKU • Cost • Pricing Level Identifier (e.g. Country) Optional: • Validity Period • Currency	Pricing Strategy: Cost+ Margin Calculation / Margin Checks
	Advanced Cost Definition	Per Advanced Cost type: • The same as Product Cost • Optionally, you can use SUM, AVG when you want to use multiple cost entries and combine them into one	Advanced Cost Module
	Discount Structure	Discount Groups and Discount Values	Pricing on List Price Net Price level

	Exchange Rates	Currency and Conversion Rates	Dependent pricing in different currencies
	Competition Data	Mandatory: • SKU • Competition Price Optional: • Competitor Name • Validity Dates • Currency • Relevant Competition Definition	All Competition Based Pricing Strategies
	Price List Data	• SKU • Pricing Level (Country, Global, Channel, ...)	Initial Prices All checks and KPI regarding price increase Pricing Strategy: Price Increase
	Stock Data	• SKU • Pricing Level Identifier (e.g. Country) • Stock Optional: • Validity Dates	KPI Days of Cover (DoC) DoC related Pricing Strategies
	Product References	• SKU • Reference SKU	Pricing Strategies: Anchor Pricing, Attribute Based Pricing
	BoM (Kit List)	• SKU • Sub-Component SKU • Quantity	Pricing Strategy: Kit Pricing
Business Data	Plus for Product	• Lookup Key • Plus %	Pricing Strategy: Cost+
	Price Increase	• Lookup Key • Price Increase	Pricing Strategy: Price Increase
	Pricing relevant Attributes	• Attributes in P/PX • Value Impacts of Pricing Attributes	Pricing Strategy: Attribute Based
	Pricing Level Adjustment	• Pricing Level • Adjustment %	Dependent Pricing of Parent (e.g. Global Reference Price List Country Price List)
	Sales Forecast	• SKU • Sales Volume • Forecast Date • Pricing Level Identifier (e.g. Country) Optional: • Currency	Lookup Based Forecast (optionally, you can use Lookup Based or Transaction Based Forecast)
			All KPI regarding “Sales History”

Transaction Data	Sales Data (Transactions)	• SKU • Sales Volume • Sales Turnover (Sold Price) • Transaction Date • Pricing Level Identifier (e.g. Country) Optional: • Currency (you need filled CCY DM when using CCY)	• Historical Sales/Turnover • Forecast based on historical data

Install Price Setting Accelerator

The installation process consists of several important activities – some require a research, some provide certain data, and some are more or less automated.

- Installation Prerequisites (Price Setting)
- Installation Steps (Price Setting)
- Post-Installation Steps (Price Setting)

Installation Prerequisites (Price Setting)

Before you start installing Price Setting Package, you need to meet the following requirements:

- Get access to **PlatformManager** and the target **partition**, as described in common accelerator installation prerequisites.
- Find out your **Product Hierarchy/Segmentation** – see Product Segmentation (Price Setting). You will need it in one of the deployment steps. You can define the product segmentation for all of the features at once, and then also individually per feature (see Product Segmentation per Feature), if needed.
  ⚠ This setting directly influences the structure of tables created during installation. It is important to do a proper research of the required product segmentation before installation, because re-configuration after installation requires that you to go through a special manual process.
- Make sure the target partition has the following bare minimum **data** tables and fields:
  - Product Master table with configured fields for attributes which will be used for the Product Segmentation.
    - Data rows available (you need at least 1 row to be able to create a price list after installation).
- Review the Mandatory Data (Price Setting) article to understand which data and tables you will need depending on your required use cases. This is not necessary for the installation itself, but for the configuration afterwards.
- Prepare a **CSV file with Dependency Configuration** – see Pricing Levels (Price Setting). This file will be required in one of the deployment steps.
  ⚠ This setting directly influences the structure of tables created during installation.

  Examine your customer data and requirements before you fill in this configuration because any re-configuration requires that you go through some manual steps. You can find frequently needed procedures in Common Configuration Procedures (Price Setting).

  You will need to upload this configuration as a CSV file. It is recommended that you have *attributeIds* in the first line of the file as headers, so that PlatformManager can automatically map it with the proper fields. Otherwise, manual mapping might be required. The file will be stored in the DependencyConfiguration PP table, so you can import all the information that this table accepts.

- Here is an example of a dummy business CSV file with dependencies. Your "Preference" fields will most likely be different from this example.

```
DependencyLevelName,DependsOn,SourceType,SourceId,Dimension,
Currency,IsComplete,Preference1_IsoCode,Preference2_SalesOrg
Global,Independent,*,*,,EUR,No,GL,SO20
Germany,Global,*,*,Country,EUR,No,DE,SO24
United Kingdom,Global,*,*,Country,GBP,No,UK,SO30
United States of America,Global,*,*,Country,USD,No,US,SO26
```

- The only mandatory fields for installation are:
  - DependencyLevelName
  - DependsOn
  - Dimension
  - Currency
  - SourceType – Since you do not have any price lists yet, place a star "*" symbol here.
  - SourceId – Since you do not have any price lists yet, place a star "*" symbol here.
- The rest of the fields (i.e. IsComplete and the various "Preferences") can be set up later during configuration of the package.

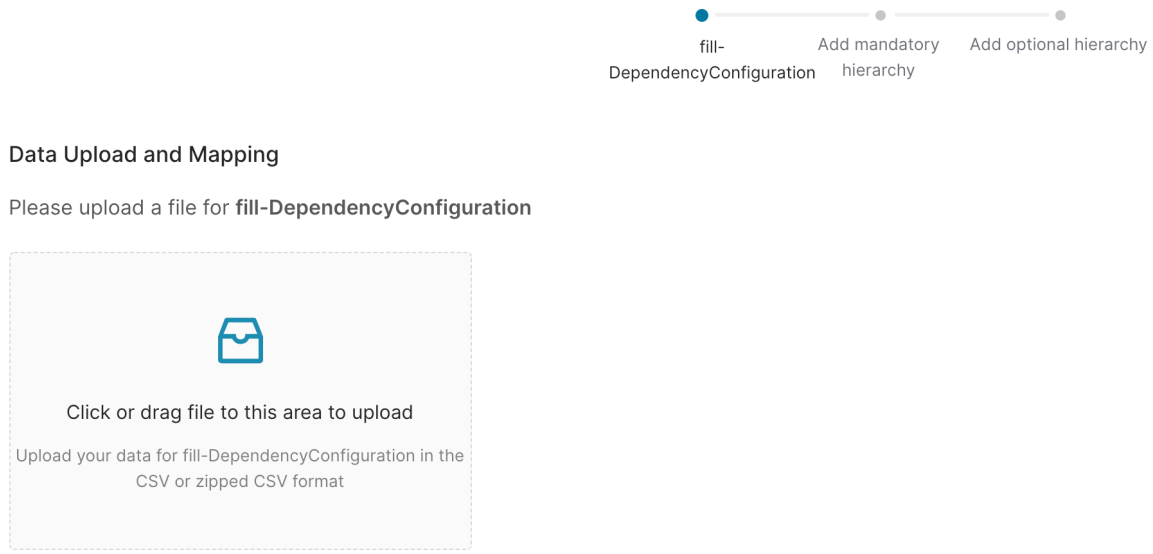
Installation Steps (Price Setting)

Common accelerator installation steps are described at How to Deploy Accelerator Package page. This article focuses only on the steps specific for Price Setting Accelerator.

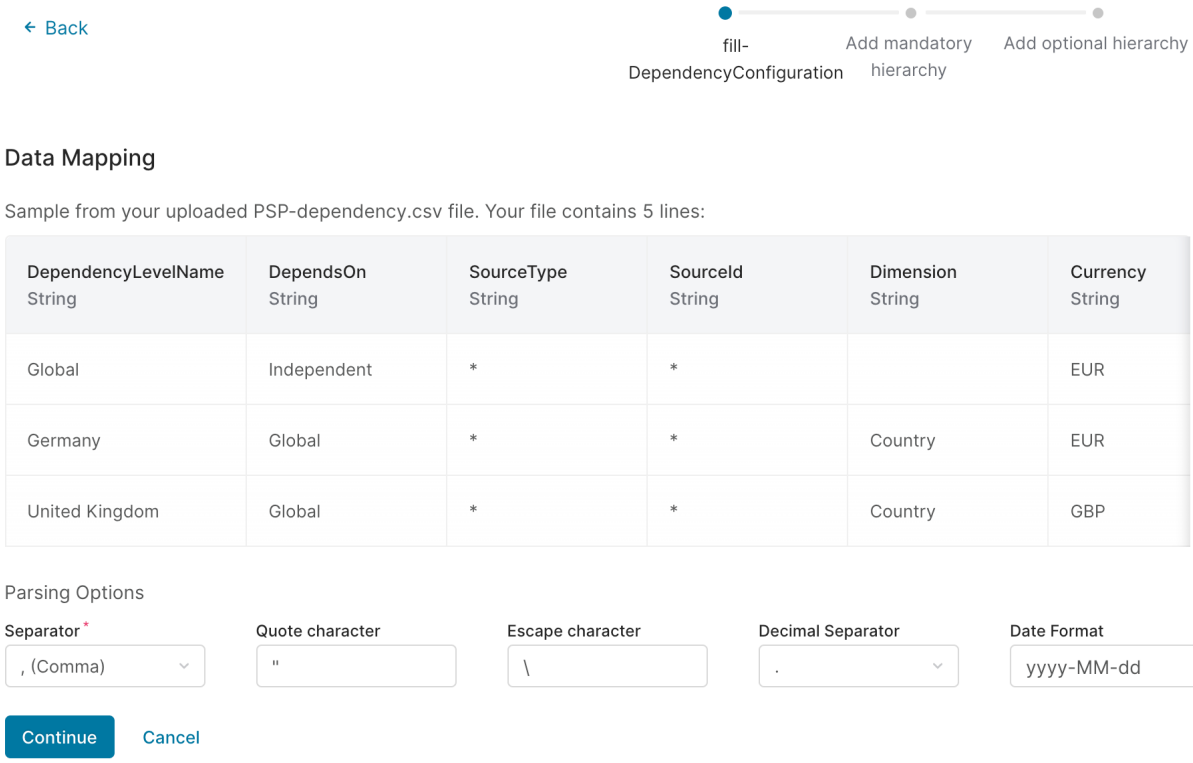
This article is based on the version 2.0.2, but it is probable that the installation process for newer releases will be similar.

To install Price Setting Package:

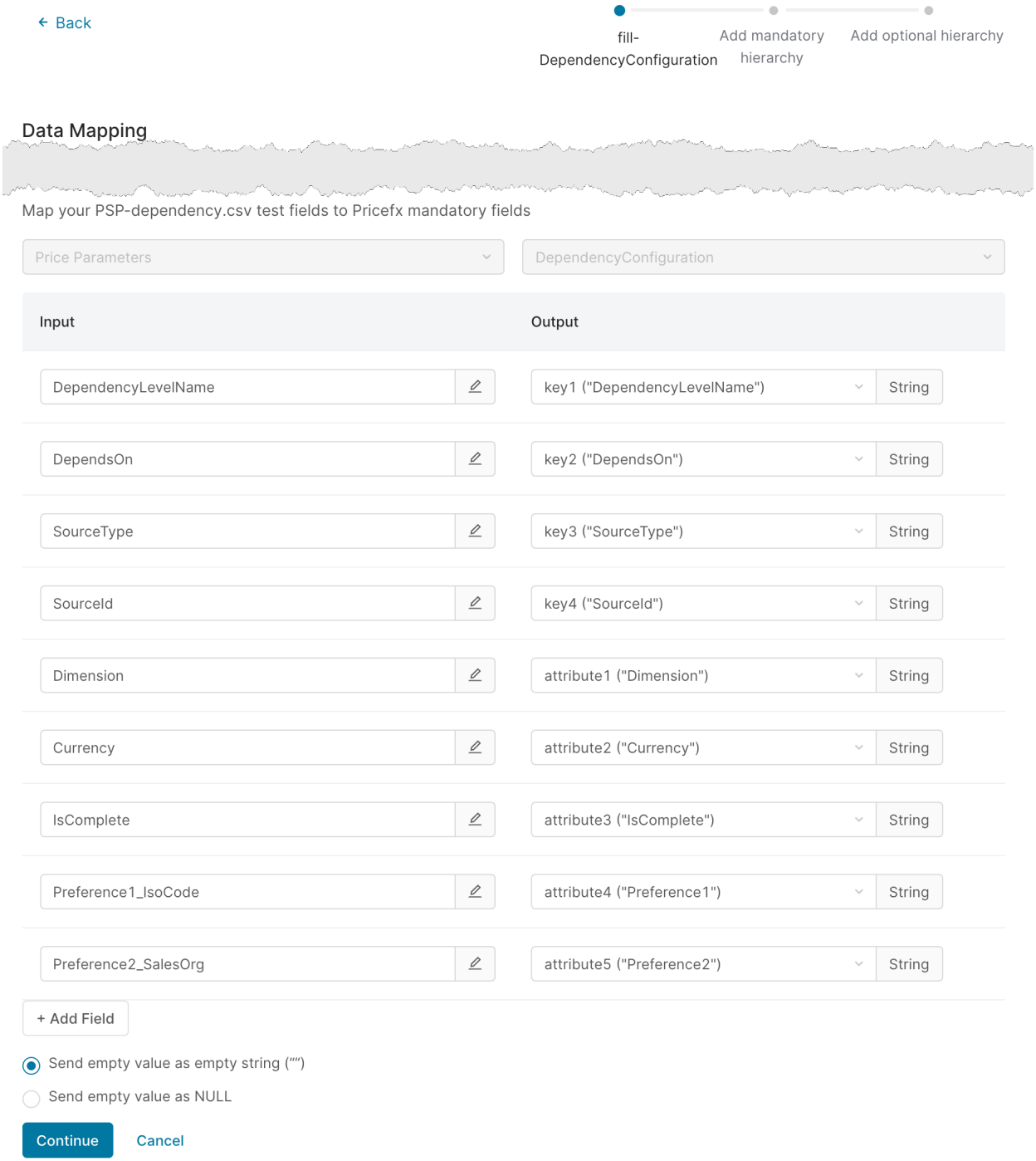
1. Access PlatformManager at https://platform.pricefx.com/ and log in.
2. Go to **Marketplace** and find the *Price Setting* package.
3. Click the package tile, select the partition where you want to deploy the accelerator package and confirm the deployment dialog to start.
   For detailed description of all deployment options, see PlatformManager documentation.
4. The installer will start with deploy logics to your partition. This can take up to several minutes.
5. Configure Pricing Levels / Dependencies.
   a. Upload the prepared CSV file with Dependency Configuration which you prepared in pre-installation steps. This step directly impacts the structure of tables created during the installation process. Later re-configuration will require a set of manual steps.



b. Define the parsing setting for the CSV import, and click **Continue**.

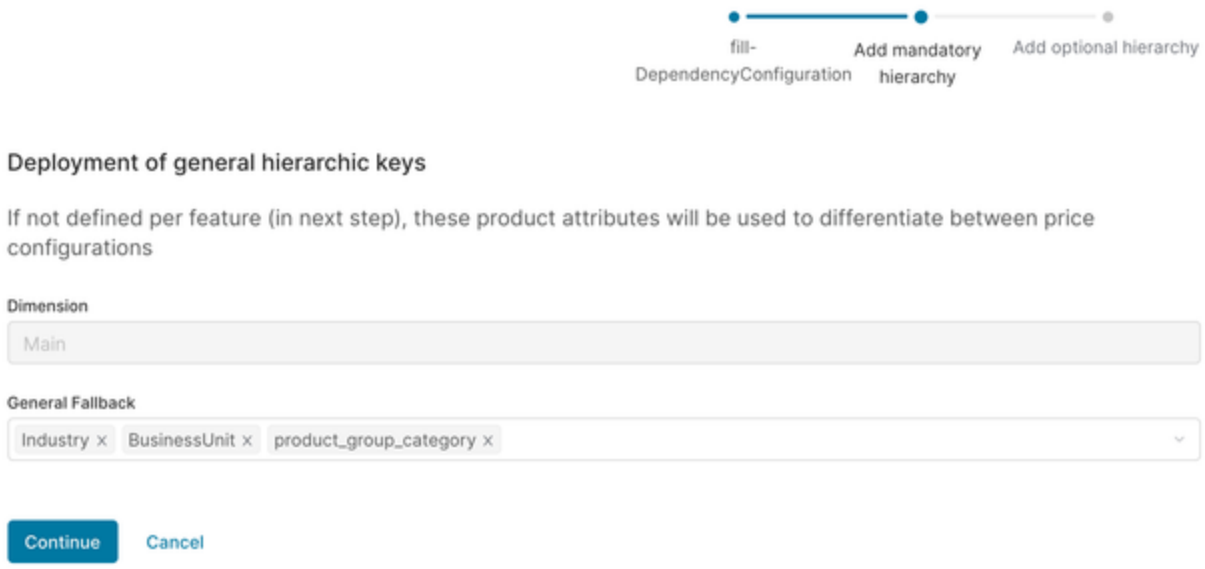


c. Define mapping of your column data from the CSV file to the fields of Company Parameter table *DependencyConfiguration.*

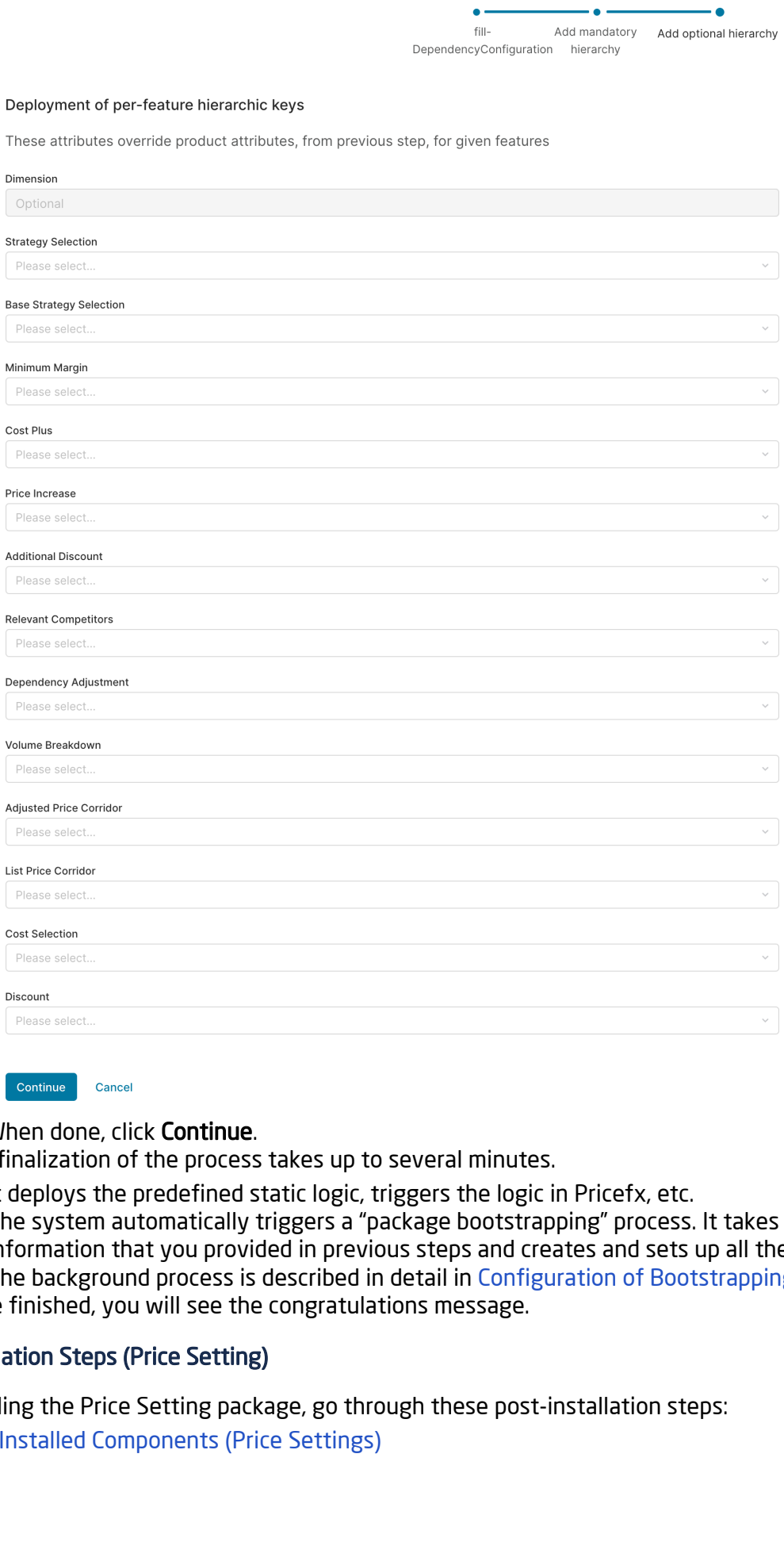


d. Wait for the process to complete.

6. Configure Product Segmentation.
   a. Set up the default Product Segmentation. This list of attributes depends on the business needs and was decided in the pre-installation steps. The attributes must be provided in the order from highest to the lowest level of the product attributes hierarchy.
   Note: These attributes come from the attributes found in the Product Master. An example of attributes could be Industry > Business Unit > Product Group. This setting directly impacts the structure of tables created during the installation process. Later re-configuration will require a set of manual steps.



b. When done, click **Continue**.
c. Wait for the process to complete.
d. Optionally, configure the Product Segmentation individually for a feature.



e. When done, click **Continue**.

7. The finalization of the process takes up to several minutes.
   - It deploys the predefined static logic, triggers the logic in Pricefx, etc.
   - The system automatically triggers a "package bootstrapping" process. It takes all the information that you provided in previous steps and creates and sets up all the required tables. The background process is described in detail in Configuration of Bootstrapping.
8. Once finished, you will see the congratulations message.

Post-Installation Steps (Price Setting)

After installing the Price Setting package, go through these post-installation steps:

- Review Installed Components (Price Settings)

- Disable All Unnecessary Features (Price Setting)
- Configure Core Elements Module (Price Setting)
- Select Logics Available to Price List/Grid (Price Setting)
- Test Deployed Package (Price Setting)

You may also need to prepare a price list/grid for business users to use; for details see Create Live Price Grid (Price Setting Package).

Review Installed Components (Price Settings)

The installation process deployed and created many components at the partition, so you can log in to the partition and review the logics, tables and settings installed by the process.

It is recommended that you review the components briefly before the configuration because you will use some of them when setting up Pricing Strategies etc.

The list of the components is at Architecture Components (Price Setting).

Disable All Unnecessary Features (Price Setting)

After deploying Price Setting Package you need to configure modules/features of the package. The package comes with a comprehensive wizard to help you with the configuration. With this wizard you can enable/disable and configure the different Price Setting Modules.

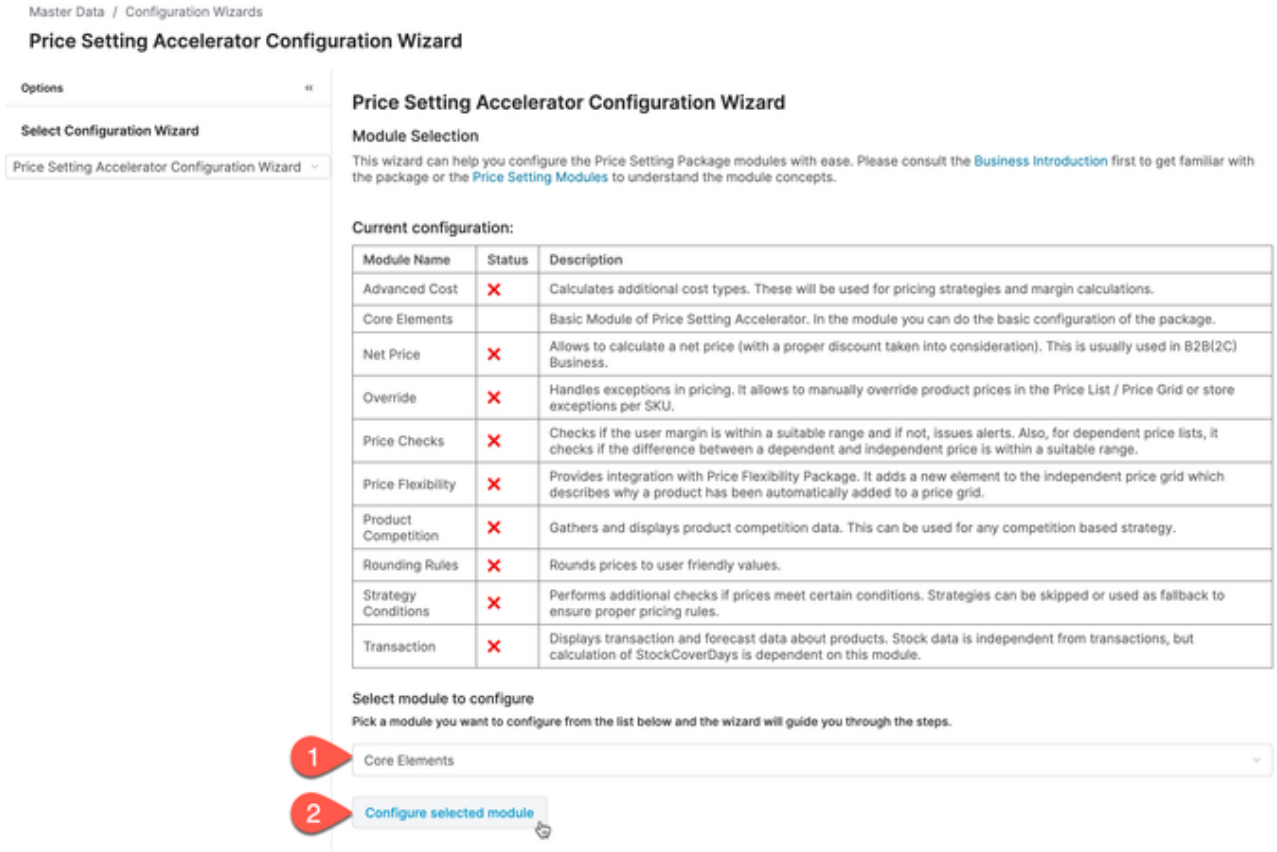
If you do not know yet which accelerator features you will need, we recommend that you disable all accelerator modules, so that you avoid many unnecessary error messages while calculating prices in the price list/grid.

To disable the accelerator modules:

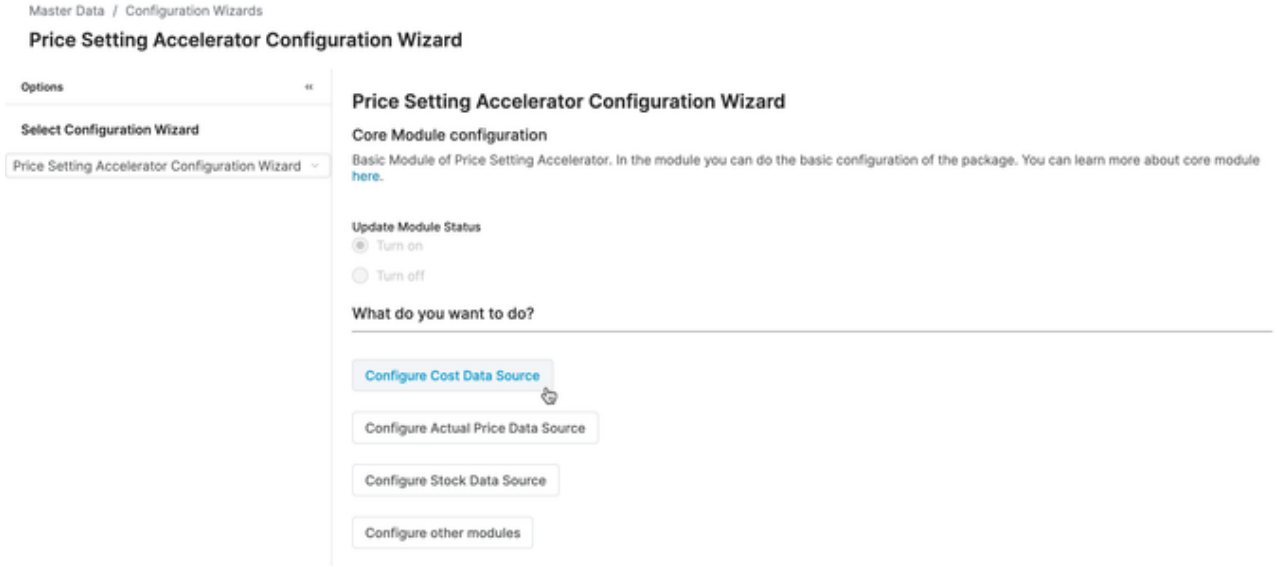
1. Open the wizard, navigate to **Company Processes** > **Price Setting Accelerator Configuration Wizard**.
2. For each module (with the exception of Core Elements) do the following:
   a. In the dropdown select the module to disable.
   b. Click the **Configure Selected Module** button.
   c. Select the option *Turn Off*.
   d. Click the **Update Module Status** button.
   e. Click the **Start Over** link.

Configure Core Elements Module (Price Setting)

1. Go to the partition where you installed Price Setting Package.
2. Navigate to **Master Data > Configuration Wizards > Price Setting Accelerator Configuration Wizard**.
3. As a module to configure select *Core Elements* and click **Configure selected module**.

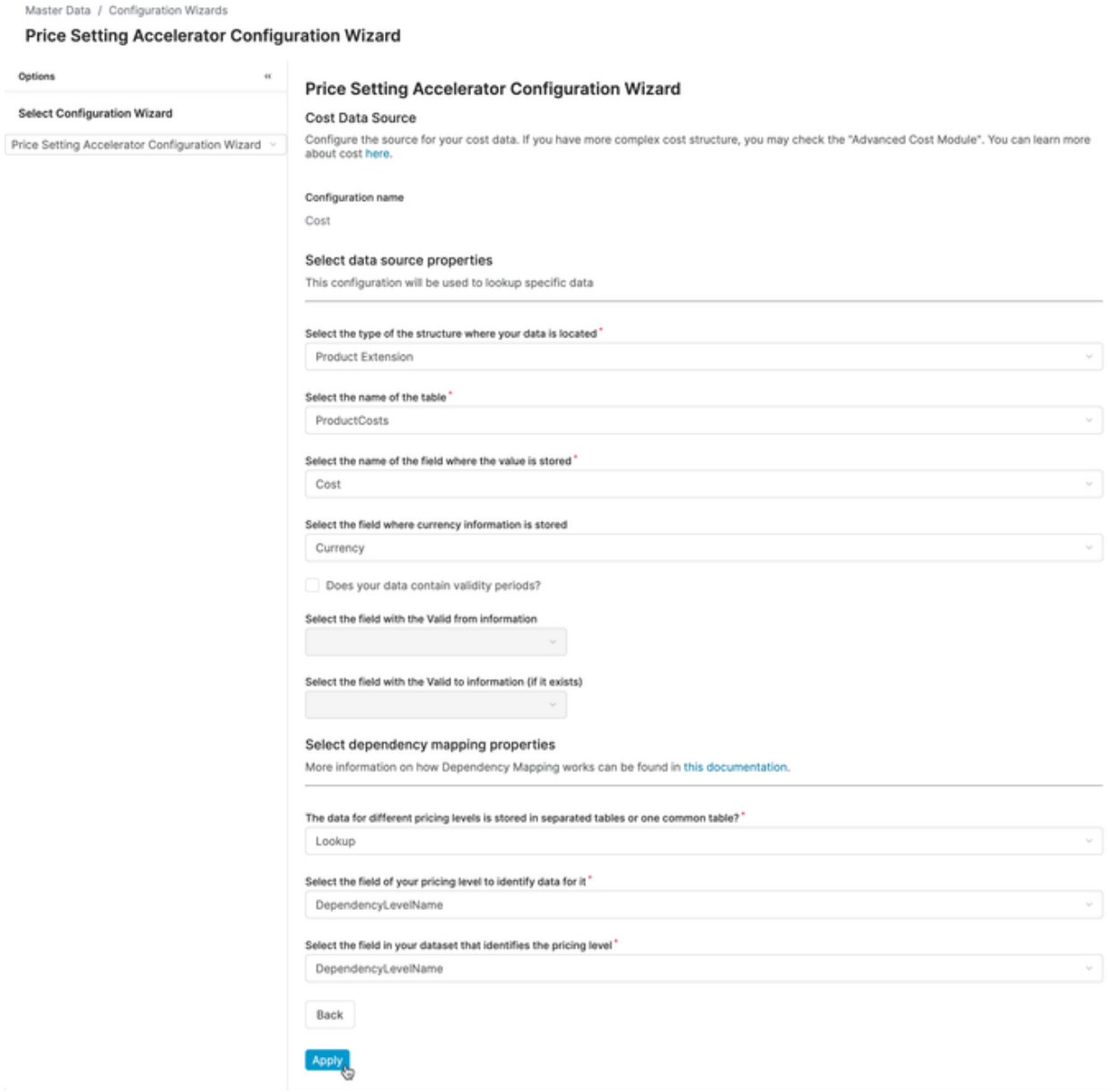


4. Click **Configure Cost Data Source**.

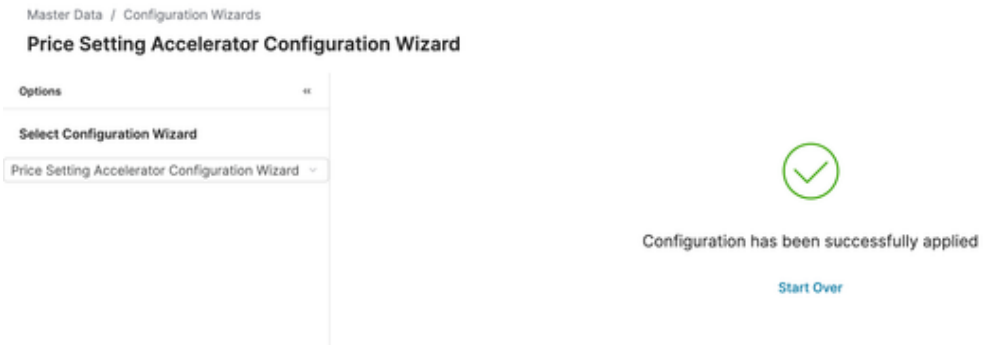


5. Set up Cost Data Source. In our case, we kept the pre-populated default values.
   a. Select Data Source table and fields to be used:
      - **Select the type of structure where your data is located** = Product Extension
      - **Select the name of the table** = ProductCosts
      - **Select the name of the field where the value is stored** = Cost
   b. Select dependency mapping properties:
      - **The data for different pricing levels...** = Lookup
      - **Select the field of your pricing level...** = DependencyLevelName

- **Select the field in your dataset that...** = DependencyLevelName



6. The wizard finishes.



Select Logics Available to Price List/Grid (Price Setting)

When users selects a logic to be used for calculation of prices in a price list/grid, they see by default a list of all “generic” logics which includes also logics for dashboards, quote items and other logics.

To limit the list of logics displayed to the users on the price list/grid definition page, take these steps:

1. Navigate to **Configuration** > **Price Setting** > **Defaults for Price List and LPG**.
2. Add the pricing logics from Price Setting accelerator to the list of "Price List Default Logics".
   a. Add ParentPriceListLogic.
   b. Add DependentPriceListLogic.
3. Save the setting.

Test Deployed Package (Price Setting)

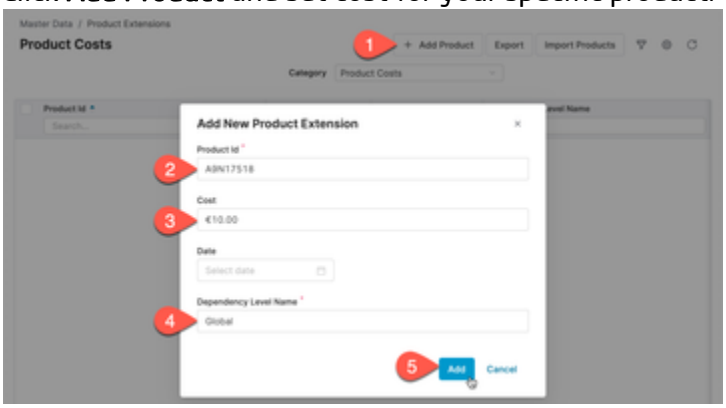
To test if the basic setup of the accelerator package is ready, you can calculate a simple price list.

- Prepare Sample Data
- Create and Calculate Price List

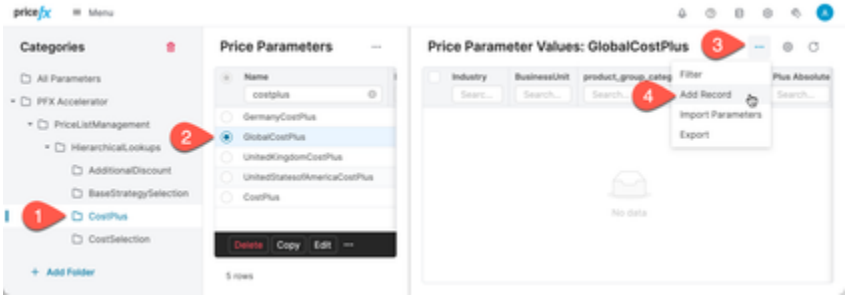
Prepare Sample Data

To calculate the price list, you will use the basic Cost Plus pricing method. For that you need to add a couple of data rows.

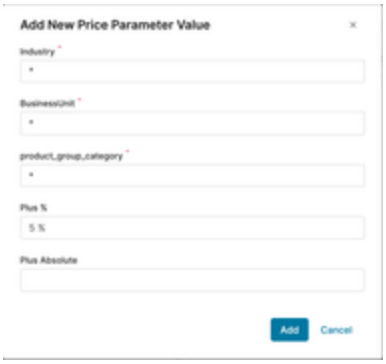
1. Manually add cost of some of your products.
   a. Navigate to **Master Data › Product Extensions**.
   b. Click **Add Product** and set cost for your specific product.



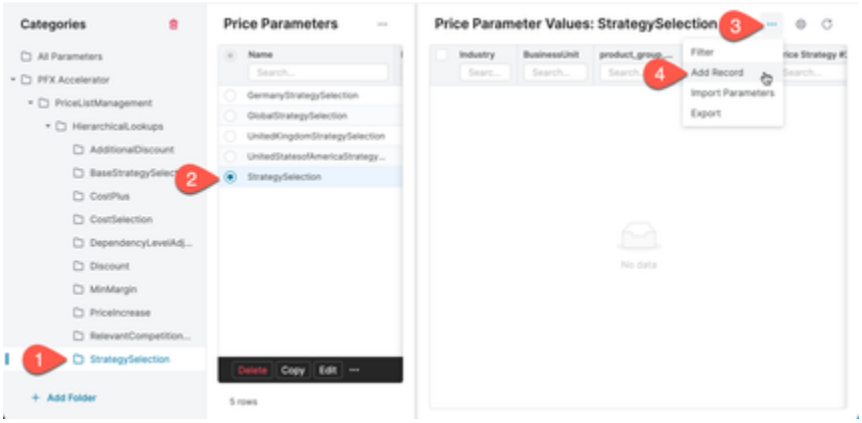
2. Manually add the percentage value added to the cost to get the price.
   a. Navigate to **Master Data › Company Parameters**.
   b. Find and open the detail of the Company Parameter *GlobalCostPlus* and then click **Add Record**.



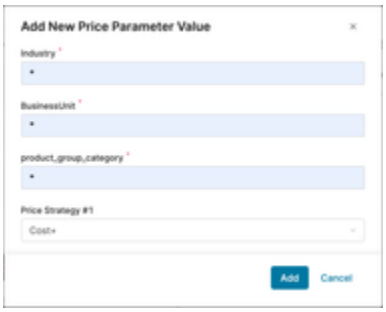
   c. Set up a default 5% for all products (regardless of the product hierarchy).



3. Set up a default price calculation strategy.

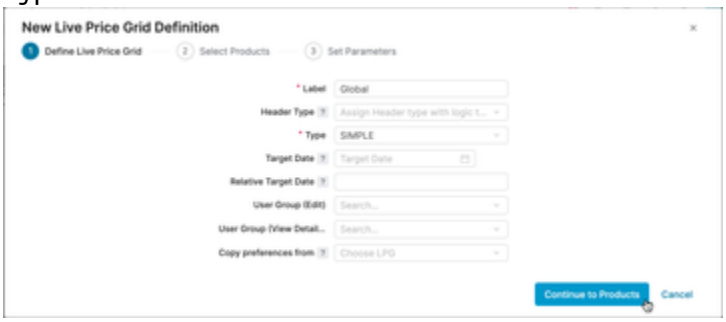


   a. Set the record data.

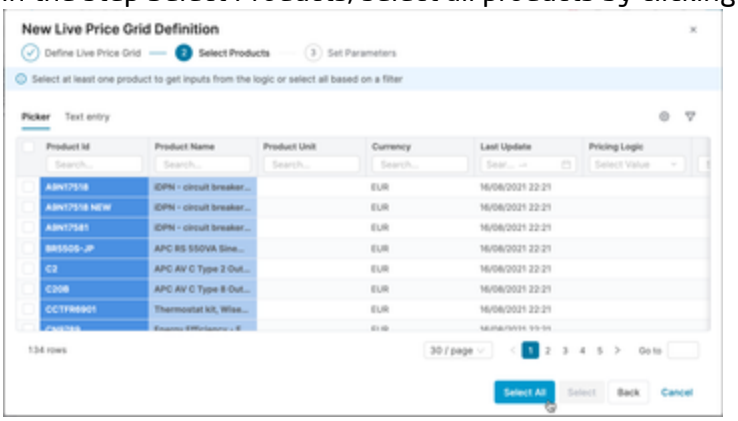


Create and Calculate Price List

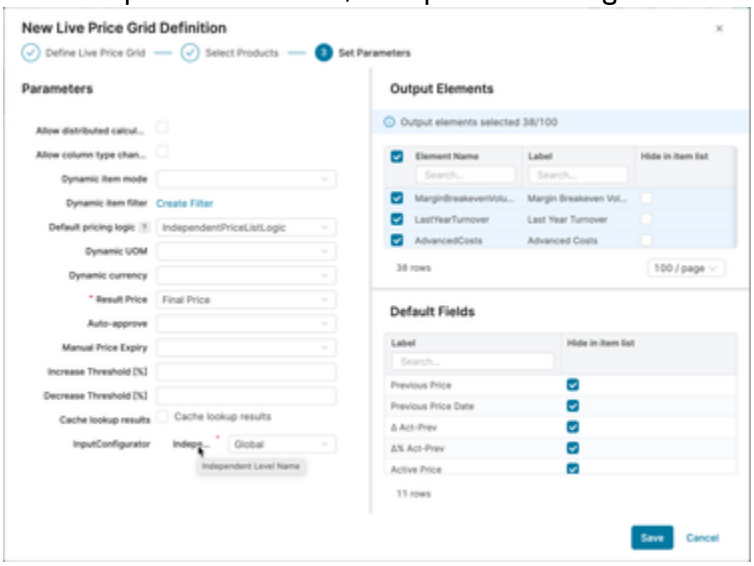
1. Navigate to **Price Setting › Live Price Grids**.
2. Click **New Price Grid**. A dialog with 3 steps will pop up.
   a. In the step *Define Live Price Grid*, set up the label and type and then click **Continue to Products**.
      - Label: Global
      - Type: SIMPLE



b. In the step *Select Products*, select all products by clicking **Select All**.



c. In the step *Set Parameters*, set up the following and then **Save** the definition.

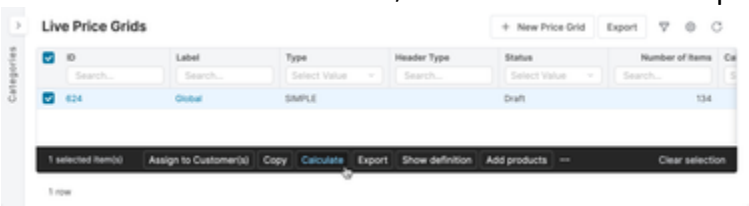


d. In the section *Parameters*, set up:
   - Default pricing logic: ParentPriceListLogic
   - Result Price: Final Price
   - InputConfigurator
     - Parent Level Name: Global

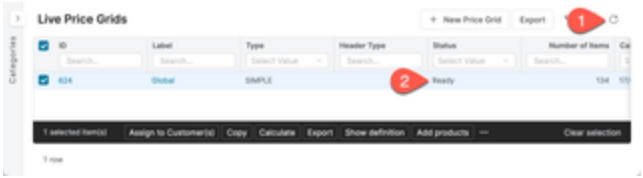
e. In the section *Output Elements,* select *all* output elements.

f. In the section *Default Fields*, set *Hide in item list* for all fields, so that the price list does not have too many columns for now.

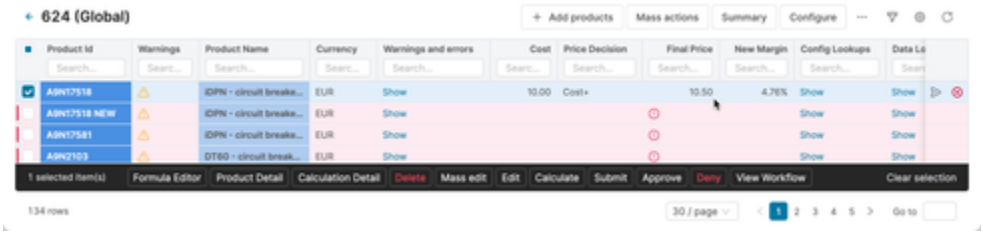
3. To calculate the Live Price Grid, select the row with the price list and click the **Calculate** button.



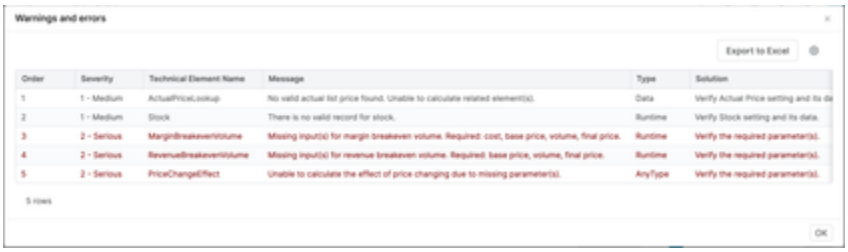
4. Wait for the *Ready* state.
   Note: The calculation will take time based on the number of products calculated. If you have thousands of products you may want to leave the calculation running in the background and come back later.
   Wait a couple of seconds and then refresh the list of LPGs to see if the calculation is ready.



5. Click the ID number link to open the Live Price Grid detail to review the line items. You should see a line for each product. Many of the fields will be empty or with errors because the data is not populated yet.
   Note that your view may differ from the following screenshot, so ensure to review the content of all columns.



   a. You should be able to see on the line:
      - Cost you defined for your product
      - Defined strategy
      - Final price
      - Margin gathered on the product when sold for the final price
   b. Click the *Show* link located in the **Warnings and errors** column to see why the prices were not calculated.



Upgrade (Price Setting)

- Upgrade of Customized Accelerator (Price Setting)
- What Parameters Can Be Changed Safely
- Upgrade Steps (Price Setting)

Upgrade of Customized Accelerator (Price Setting)

> ⚠ Before submitting a support ticket for an accelerator upgrade, please ensure that any issues encountered are not a result of your custom modifications. We do not offer support for modifications to the accelerator.

If the customer project customizes the accelerator's logics, it happens in one of the tiers described in Customization Tiers (Price Setting).

The accelerator is technically implemented as any other customer project. It uses the common Pricefx objects and it uses the pfxpackage tool for deployment (this tool is embedded in PlatformManager and in Pricefx Studio).

When you deploy the upgrade (which is also just a standard package), the tool overwrites the existing objects on the partition which have the same names as the deployed ones.

As a result, your custom modifications to the accelerator will be overwritten during the accelerator upgrade and will have to be re-applied manually.

What Parameters Can Be Changed Safely

This page tells you what parameters can be changed afterwards without redoing deployment or configuration.

Here is some general advice on how to handle configuration Price Parameters:

- All configurations work based on column names (UI name) and/or column labels/metadata (technical names).
  They do not work based on column *labels* (UI name) or labelTranslation (technical name).
- Do not change the name of any configuration PP column. It might be especially tempting in "DependencyConfiguration" PP, but don't do it.
- Column names of sample data can be freely changed assuming that the configuration will be properly adjusted.
- Values of dynamically created Price Parameters can be changed without redeployment. They act as standard Pricefx Price Parameters. The full list can be found here. Values can be added/removed as needed. If the user wants to handle all products without fallback to another DependencyLevel, we recommend to have one value with an asterisk "*" on every key. It provides a fallback without leaving the scope of the current Dependency Level configuration.
- Changing dynamically created Price Parameters requires caution, but can be done according to this guide.
- PriceSettingConfig and other configuration PPs should be changed with extreme caution. Technically every configuration can be changed during the system run but changes can affect a lot of things. That is why you should be careful here and customers are not even recommended to do this on their own in PROD environments without testing changes on some QA instance.
- Making changes to price setting levels is tricky. See Adjustments after Changing Price Setting Level.

Upgrade Steps (Price Setting)

This section explains how to upgrade Accelerator to a new version.

Note: Optional features might need to be reconfigured after the upgrade. For details see the documentation of specific modules/engines.

Sequence of steps:

- Create Backup of All Logics
- Find Out Which Accelerator You Have
- Read All Manual Changes You Need to Perform
- Execute All Upgrade Notes
- Run "Price Setting Package - Upgrade"
- Execute Rest of Upgrade Notes

Create Backup of All Logics

Changes to logics will not be preserved through the upgrade. They will need to be applied again in the best case scenario. In the worst case scenario, they will need to be developed from scratch if the code has changed too much.

Find Out Which Accelerator You Have

The easiest option to find out which Accelerator Package you have installed on your partition is to log in PlarformManager and navigate to **Marketplace** > **Templates Management** > **Deployed Templates** and search for your package and partition.

Read All Manual Changes You Need to Perform

Read through this directory: Release Notes (Price Setting)

All manual changes will be marked as "Upgrade Notes" or similarly.

⚠ You need to check all versions between your version and the target one, not only newest/major one.

Execute All Upgrade Notes

Follow all of the instructions in the release notes guiding you what to do before upgrading.

Remember that upgrade notes are created additively, so you should do them in the proper order.

Run "Price Setting Package - Upgrade"

Go to **PlatformManager** > **Marketplace** > **Price Setting Package - Upgrade** and perform the upgrade.

Execute Rest of Upgrade Notes

If there is no indicator when the upgrade note should be executed, it means it should be executed now.

Configure Pricing Strategies (Price Setting)

- Configure Cost Plus Pricing Strategy (Price Setting)
- Configure Competition Based Pricing Strategy (Price Setting)
- Configure Attribute Based Strategy (Price Setting)
- Visually Configure Custom Pricing Strategy (Price Setting)
- Select Pricing Strategies used for Calculation of Prices
- Order of Prices in Dependent Price List/Grid

Configure Cost Plus Pricing Strategy (Price Setting)

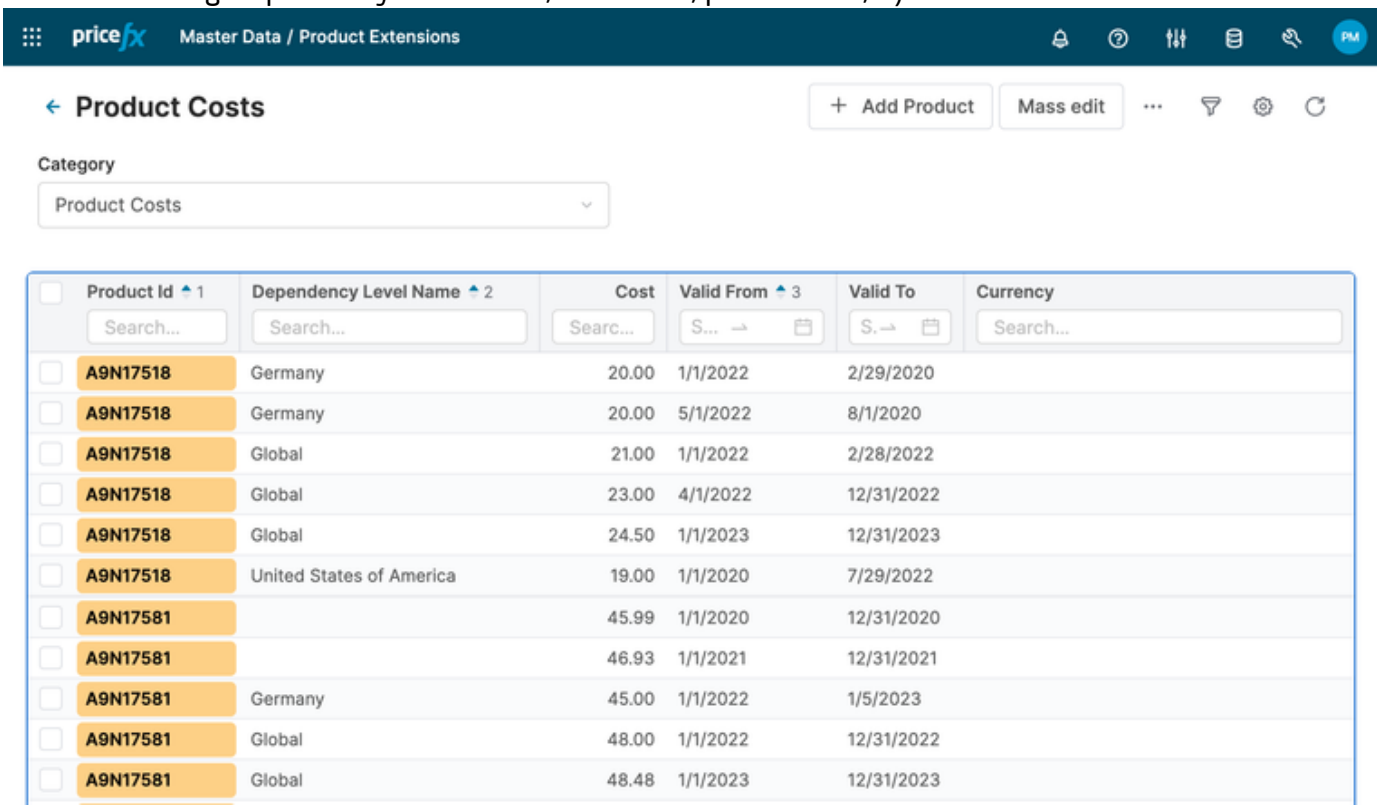
This article explains how to set up Cost Plus pricing strategy in the Price Setting accelerator using the out-of-the-box features.

Prerequisites

Before you start, ensure you know the following:

- Which Cost+ method for calculation you will use. Review the available methods in Adjustment Engine article.
- What part of the business will be priced by this strategy:
  - Pricing Levels
  - Product Segments

- Storage and structure of the Cost data table:
  - The cost data must be stored in a table type supported by the Price Setting accelerator (e.g. Product Extension).
    - When you store the cost data in more tables per Pricing Level, make sure you have consistent naming pattern for the tables. The name must consist of 2 parts:
      - Pricing Level attribute value – Usually the Pricing Level name, but it could also be the Dimension value, or Preference1 value, etc.
      - Name of the Cost Table – This part of the name must be the same for all the Cost tables.
  - Be familiar with contents of table fields, to understand which fields store the product ID, pricing level attribute (if used), cost money amount, cost currency, validity range of the cost.
    - Find out if you have different Cost amount values for different Pricing Levels' attribute (e.g. per Pricing/Dependency Level name, dimension, preference1, ...).



Product Id 1	Dependency Level Name 2	Cost	Valid From 3	Valid To	Currency
A9N17518	Germany	20.00	1/1/2022	2/29/2020	
A9N17518	Germany	20.00	5/1/2022	8/1/2020	
A9N17518	Global	21.00	1/1/2022	2/28/2022	
A9N17518	Global	23.00	4/1/2022	12/31/2022	
A9N17518	Global	24.50	1/1/2023	12/31/2023	
A9N17518	United States of America	19.00	1/1/2020	7/29/2022	
A9N17581		45.99	1/1/2020	12/31/2020	
A9N17581		46.93	1/1/2021	12/31/2021	
A9N17581	Germany	45.00	1/1/2022	1/5/2023	
A9N17581	Global	48.00	1/1/2022	12/31/2022	
A9N17581	Global	48.48	1/1/2023	12/31/2023	

Review Data Tables

1. Review the Cost table/s to know the field's names and labels, and to know what data are in these tables.
2. Review the Company Parameters with names ending with "StrategySelection". For new projects you will see several out-of-the-box tables to help you start, and in existing projects you will see how the project was managed until now.
3. Review the Company Parameters with names ending with "CostPlus". For new projects you will see several out-of-the-box tables to help you start, and in existing projects you will see for which Pricing Levels and for which Product Segments the Plus factors were defined (if at all).

Select a Pricing Strategy for Price Calculations

1. Review the article Select Pricing Strategies used for Calculation of Prices.
2. Decide on which Pricing Level you need to do the modification. Based on that, identify the proper "*StrategySelection" table to modify.

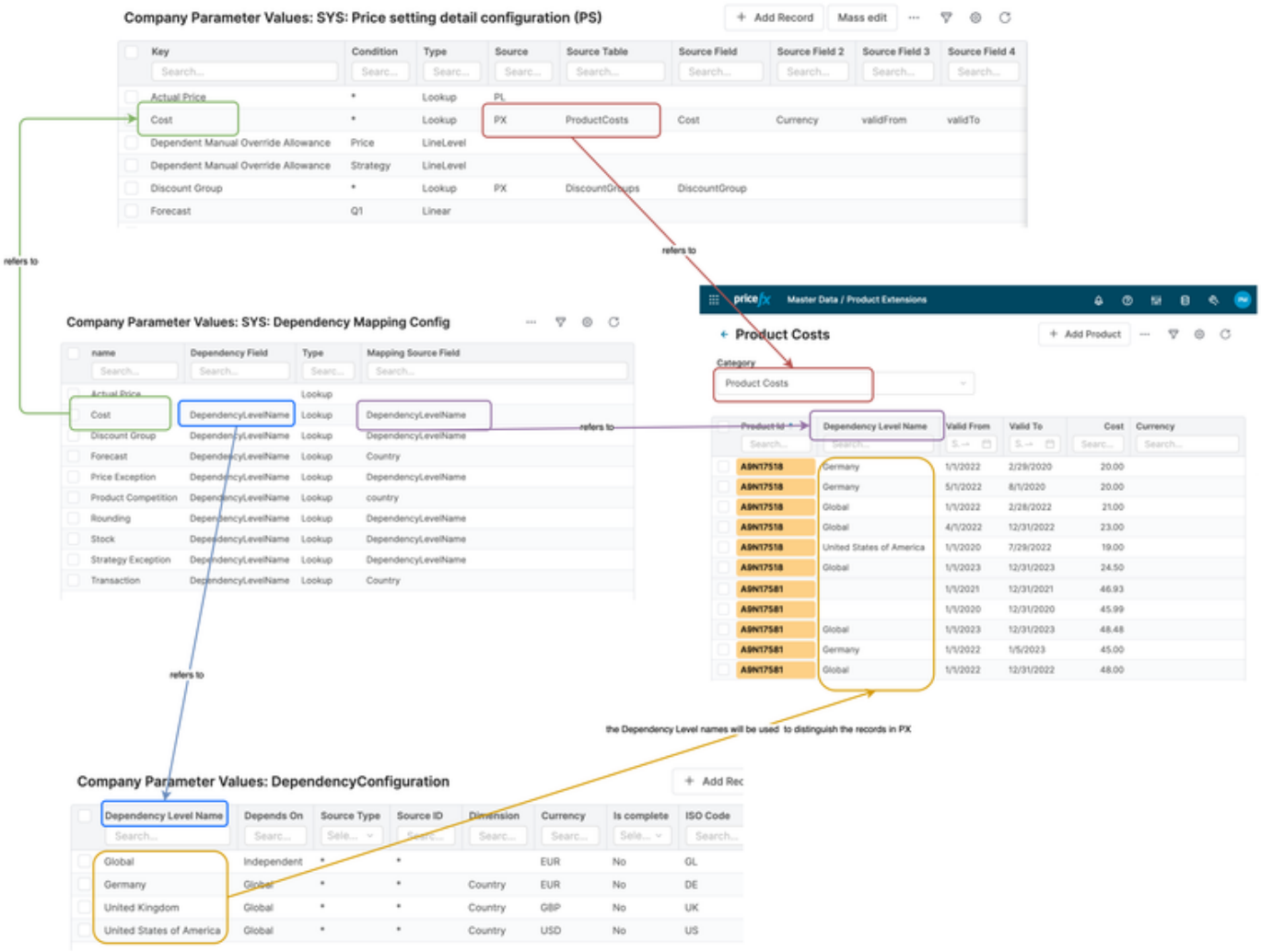
3. Add/modify records in that table for a specific combination of the ProductSegment for which you want to use Cost+ pricing strategy.
4. Set up the "Cost+" strategy for specific Product Segment combination or on "star" level.
5. Set up the proper priority of your pricing strategy – as first, second, etc. for the Product Segment.

Configure the Source of Cost Data

1. Navigate to **Company Processes › Price Setting Accelerator Configuration Wizard**.
2. Select **Core Elements** module for configuration.
3. Click **Configure Cost Data Source**.
4. Set **Data Source Properties**.
   - Choose the table type for storing Cost data, e.g., select "Product Extension.".
   - Specify the table containing Cost data, e.g., choose "ProductCost."
   - Indicate the field storing the Cost money amount, e.g., select "Cost."
   - Specify the field containing currency symbol information, e.g., choose "Currency." Leave empty if not available.
   - If your dataset includes validity dates for Cost, enter the field name which contains the "Valid From" date. Optionally, define the field name for "Valid To".
5. Set **Dependency Mapping Properties**.
   a. Choose how data are stored:
      - Select "None" if there are no different Cost amount values for various Pricing Levels attributes (e.g., Pricing Level name, dimension, preference1, ...).
      - Select "Table" if Costs are stored in separate tables per Pricing Level.
      - Select "Lookup" if Cost data for different Pricing Levels are stored together in a single table, distinguished by a field value.
   b. Select a field from Pricing Level.
      - For "Table" or "Lookup" options, specify which Pricing Level property/field distinguishes between tables (for "Table") or records (for "Lookup") containing cost information for the same product but for different Pricing Levels. For example, distinguish cost data by Pricing Level Name, Dimension, Preference1, etc.
   c. Select a field from the cost data table.
      - For "Lookup" option, specify which field in the cost table distinguishes cost records for the same product but different Pricing Levels.

Configuration Tables

Once you finish the wizard, the settings are stored in Company Parameters "PriceSettingConfig" and "DependencyMappingConfig".

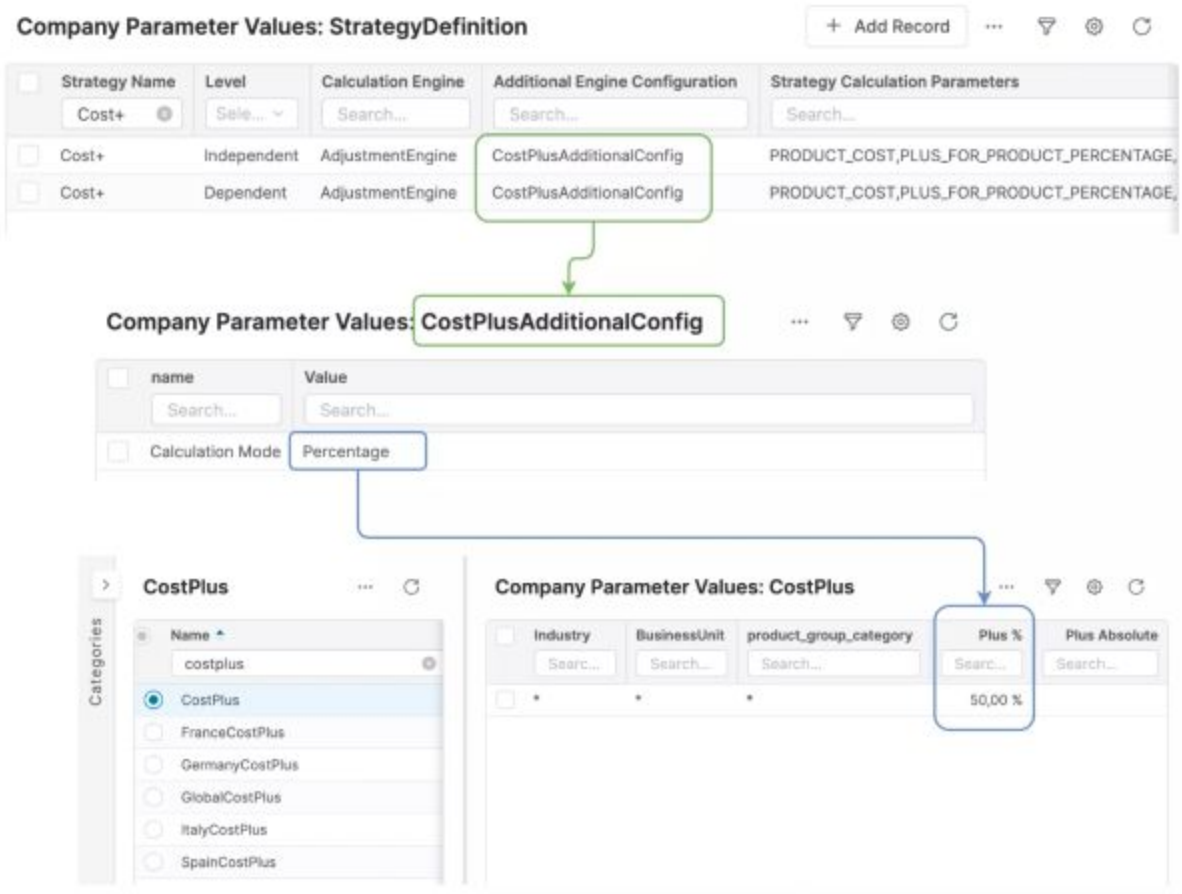


Select Price Calculation Formula

The out-of-the-box Cost+ strategy can calculate the result price in several ways – Absolute, Percentage and SellingPrice. Review the actual calculation formulas in Adjustment Engine which is used to calculate the Cost+ strategy.

To decide which of the formulas (and thus also which field of the CostPlus table) will be used, do following:

1. Change the “Calculation Mode” setting found in Company Parameter “CostPlusAdditionalConfig”.



Configure Plus Factors

As a final step, you need to set up the "Plus" for calculation of the Cost+ strategy. See article Update Cost Plus Adjustment (Price Setting).

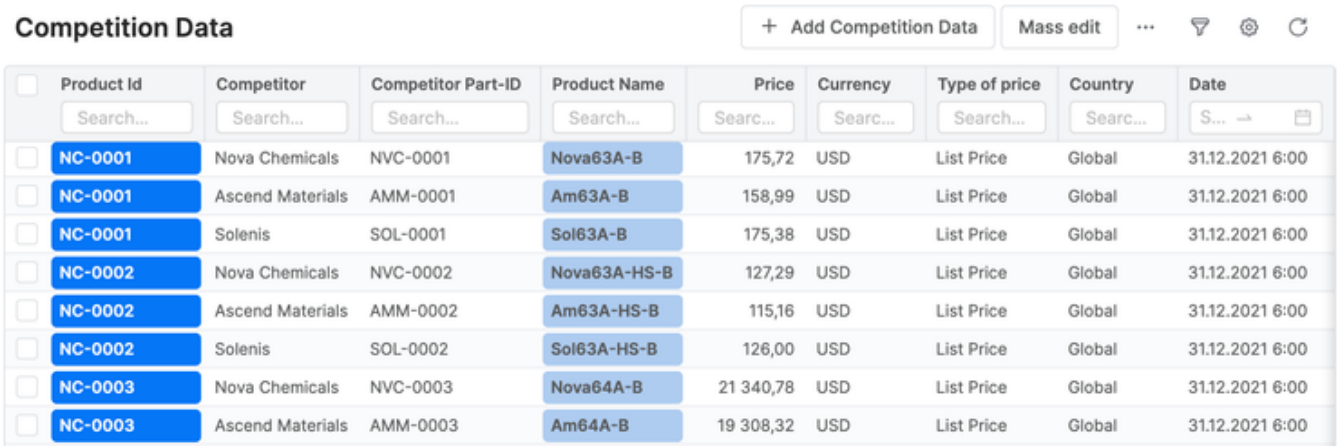
Configure Competition Based Pricing Strategy (Price Setting)

This article explains how to set up Competition Based Pricing Strategy in Price Setting accelerator using the out-of-the-box features.

Prerequisites

Before you start, ensure you know the following:

- What part of the business will be priced by this strategy:
  - Pricing Levels
  - Product Segments
- Storage and structure of the Competition data table:
  - Competitor prices data must be stored in a table supported by the Price Setting accelerator – in this case in the Competition Data table in Master Data.
  - Find out if you have different Competition records for different Pricing Levels' attributes (e.g. per Pricing/Dependency Level name, dimension, preference1, ...)
    - Find out which field of the Competition Data table stores the Pricing Level attribute (it could be, for example, the "country" field).



Product Id	Competitor	Competitor Part-ID	Product Name	Price	Currency	Type of price	Country	Date
NC-0001	Nova Chemicals	NVC-0001	Nova63A-B	175,72	USD	List Price	Global	31.12.2021 6:00
NC-0001	Ascend Materials	AMM-0001	Am63A-B	158,99	USD	List Price	Global	31.12.2021 6:00
NC-0001	Solenis	SOL-0001	Sol63A-B	175,38	USD	List Price	Global	31.12.2021 6:00
NC-0002	Nova Chemicals	NVC-0002	Nova63A-HS-B	127,29	USD	List Price	Global	31.12.2021 6:00
NC-0002	Ascend Materials	AMM-0002	Am63A-HS-B	115,16	USD	List Price	Global	31.12.2021 6:00
NC-0002	Solenis	SOL-0002	Sol63A-HS-B	126,00	USD	List Price	Global	31.12.2021 6:00
NC-0003	Nova Chemicals	NVC-0003	Nova64A-B	21 340,78	USD	List Price	Global	31.12.2021 6:00
NC-0003	Ascend Materials	AMM-0003	Am64A-B	19 308,32	USD	List Price	Global	31.12.2021 6:00

Review Data

1. Review the Competition Data table to know the field names and labels, and to know what data are in this table.
2. Review the Company Parameters with names ending with "StrategySelection". For new projects you will see several out-of-the-box tables to help you start, and in existing projects you will see how the project was managed until now.
3. Review the Company Parameters with names ending with "CompetitionAdditionalConfig". For new projects you will see several out-of-the-box tables to help you start. In existing projects, however, they may have different names, so you need to review the field Additional Engine Configuration in Company Parameter table StrategyDefinition, to know which tables are used.

Configure Product Competition Module

Enable Product Competition Module

1. Navigate to **Company Processes** > **Price Setting Accelerator Configuration Wizard**, select the *Product Competition Module*, and click **Configure**.
2. Turn the module ON and click **Update Module Status**.

Set up Dependency Mapping for Competition Data Table

1. Navigate to **Company Processes** > **Price Setting Accelerator Configuration Wizard**, select the *Product Competition Module*, and click **Configure**.
2. Click **Configure Dependency Mapping**.
3. If your Competition Data are NOT different per Pricing Level, then select *None*, and click **Apply**.
4. If your Competition Data are different per Pricing Level, set the mapping by *Lookup*, and then select the mapping between 2 fields - field of the Pricing/Dependency Level (e.g. Name, Dimension, Preference1, etc.) and the field in your Competition Data table (e.g. country, additionalInfo3, etc.). Save the setup by clicking **Apply**.

Configure Strategy

1. Review the article Configure Pricing Strategy for Pricing Level and Product Segment.
2. Add/modify records in that "*StrategySelection" table for a specific combination of the ProductSegment, for which you want to use Competition Based pricing strategy.
3. Select the *MinCompetition*, *MaxCompetition* or *AvgCompetition* strategy for specific Product Segment combination or on "star" level.
4. Set up the proper priority of your pricing strategy - as first, second, etc. for the Product Segment.

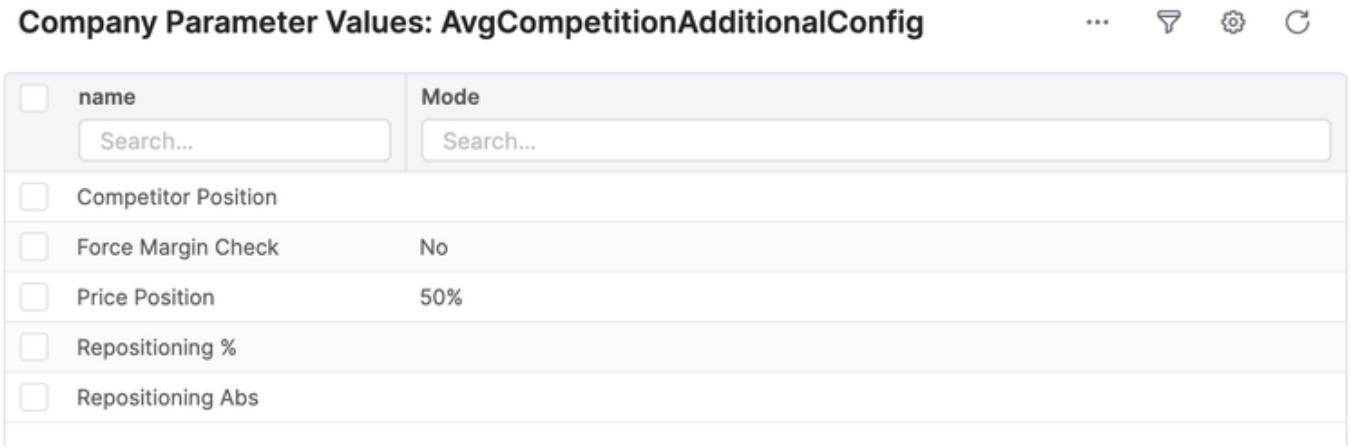
Adjust Setting of Competition Additional Config

The strategies MinCompetition, MaxCompetition and AvgCompetition come with additional configuration out-of-the-box which you may modify.

⚠️ Remember that if you modify the default additional configuration, it will apply to **each product segment** which will be calculated by that strategy.

So if you need special additional setting for your competition strategy (in the StrategyDefinition company parameter), create a new strategy and make another additional config table for it.

The out-of-the-box setup for those strategies is stored in Company Parameters MinCompetitionAdditional Config, AvgCompetitionAdditionalConfig and MaxCompetitionAdditionalConfig.



Verify

1. Recalculate the price list/grid, and verify that the strategy is used for products from the Product Segment.
2. Review the fields *Competition Data* and *Price Selector* for any errors.

Configure Attribute Based Strategy (Price Setting)

This article is for a technical person who wants to configure Attribute Based pricing strategy in Price Setting Accelerator.

Prerequisites

Before you start, ensure you know the following:

- **Product Attributes** - You need to know which tables store the product attributes and in which fields. Such data may be in multiple tables. For example, Competitiveness attribute in a Product Master table, or Color and Size in a Product Extension table.
- **Product Anchors** - You need to know which table stores the anchors for products, i.e. data defining which SKU is an anchor for another SKU. For example, they can be stored in Company Parameter "AnchorData".
- For parts of the business calculated by the Attribute Based pricing strategy find out the following:
  - Independent or dependent
  - Pricing Level
  - Product Segmentation

Review Data Tables

1. Review the table with product anchors to know the field names and labels, and to know what data are in this table.
2. Review the Company Parameters with names containing word "Attribute". For new projects you will see several out-of-the-box tables to help you start, and in existing projects you will see if some project specific tables were defined.
3. Review the Company Parameters with names ending with "StrategySelection". For new projects you will see several out-of-the-box tables to help you start, and in existing projects you will see how the project was managed until now.

Define Pricing Strategy

This strategy is not predefined out-of-the-box, because it has very flexible setup, so you need to set it based on your needs.

1. To define a new strategy, add a new record to the Company Parameter StrategyDefinition, with the following field values:
   - **Name** – For example "AttributeBased". You will refer to this name later from other tables.
   - **Level** – Depends on your Pricing Level for which you will use the strategy, so either parent, or dependent.
   - **Calculation Engine** – Use the out-of-the-box "AttributeBasedEngine".
   - **Additional Engine Configuration** – Name of the Company Parameter which contains configuration of anchor data. We recommend to name the table based on this pattern "{StrategyName}AdditionalConfig", for example "AttributeBasedAdditionalConfig".
   - **Strategy Calculation Parameters** – Use the list of predefined parameters, found in the section Default Strategy Calculation Parameters in the Attribute Based Engine article.

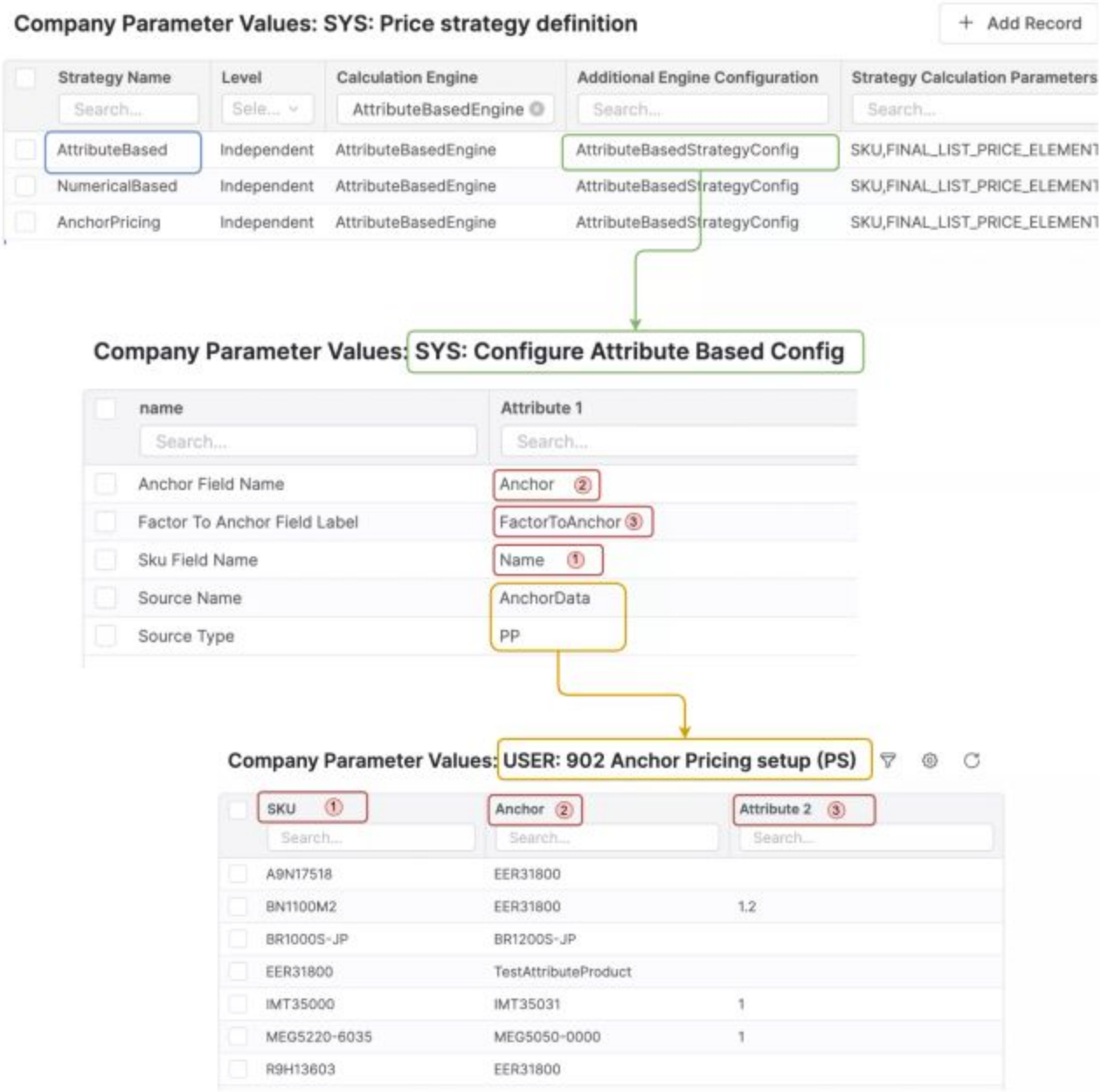
Configure Anchor Data Source

Create AdditionalConfig Table

1. Navigate to **Company Parameters**, to the folder **PFX Accelerator** > **PricingStrategiesConfig**.
2. Create a new Company Parameter table with the following settings:
   - **Name** – We recommend to name it based on this pattern "{StrategyName}AdditionalConfig", for example "AttributeBasedAdditionalConfig".
   - **Valid After** – From when should this setting be valid. If you are going to use it for a price list/grid with effective date (validity from) as of date X, set this field to value X-1.
   - **Table Type** – Matrix
   - **Value Type** – Matrix
   - **Status** – Active
3. To configure the value field of the table, make the field "attribute1" visible, and set its **Name** to "parameterValue" and **Label** to "value".

Link Table with Anchor Data

Your anchor data is stored in a table at a Pricefx partition. You need to set up several configuration options in the AdditionalConfig table, so that the accelerator knows where to find the table with anchor data, and which fields to use.



For details about the AdditionalConfig table, review section Additional Engine Configuration in the Attribute Based Engine article.

To add the configuration options, add several records to your AdditionalConfig table:

Source Type	Type of table containing the anchor data. For example “PP”.
Source Name	Name of the table. For example “AnchorData”.
Sku Field Name	Name of field containing ID of the product, for which we will define the anchor. For example “Name”.
Anchor Field Name	Name of field containing ID of an anchor product. For example “Anchor”.
Factor To Anchor Field Label	Name of field containing the factor used for calculation.

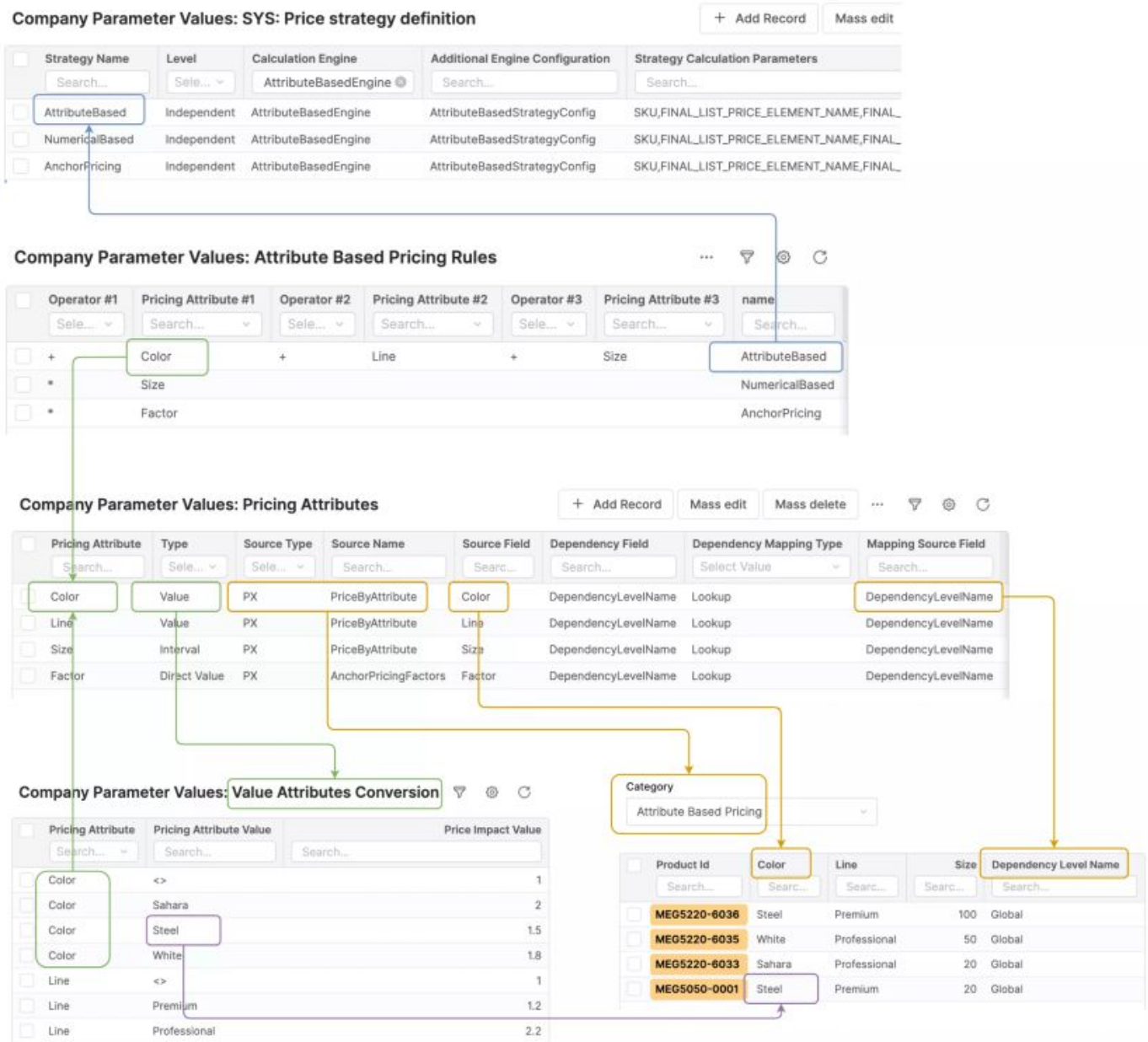
Set up Pricing Rules

To set up the pricing rules for attribute based pricing, define the following:

1. **Pricing Attribute/s**, e.g. Color, Size, etc. and which tables store their values. The Pricing Attributes can be of three types: Value Based, Interval Based and Direct Value.
2. **Attribute impact** to the price calculation: how much it influences the price. This step is not for all types of the pricing attributes.
3. **Calculation Pricing Rules** – Calculation expression, how to calculate the price based on the attributes.

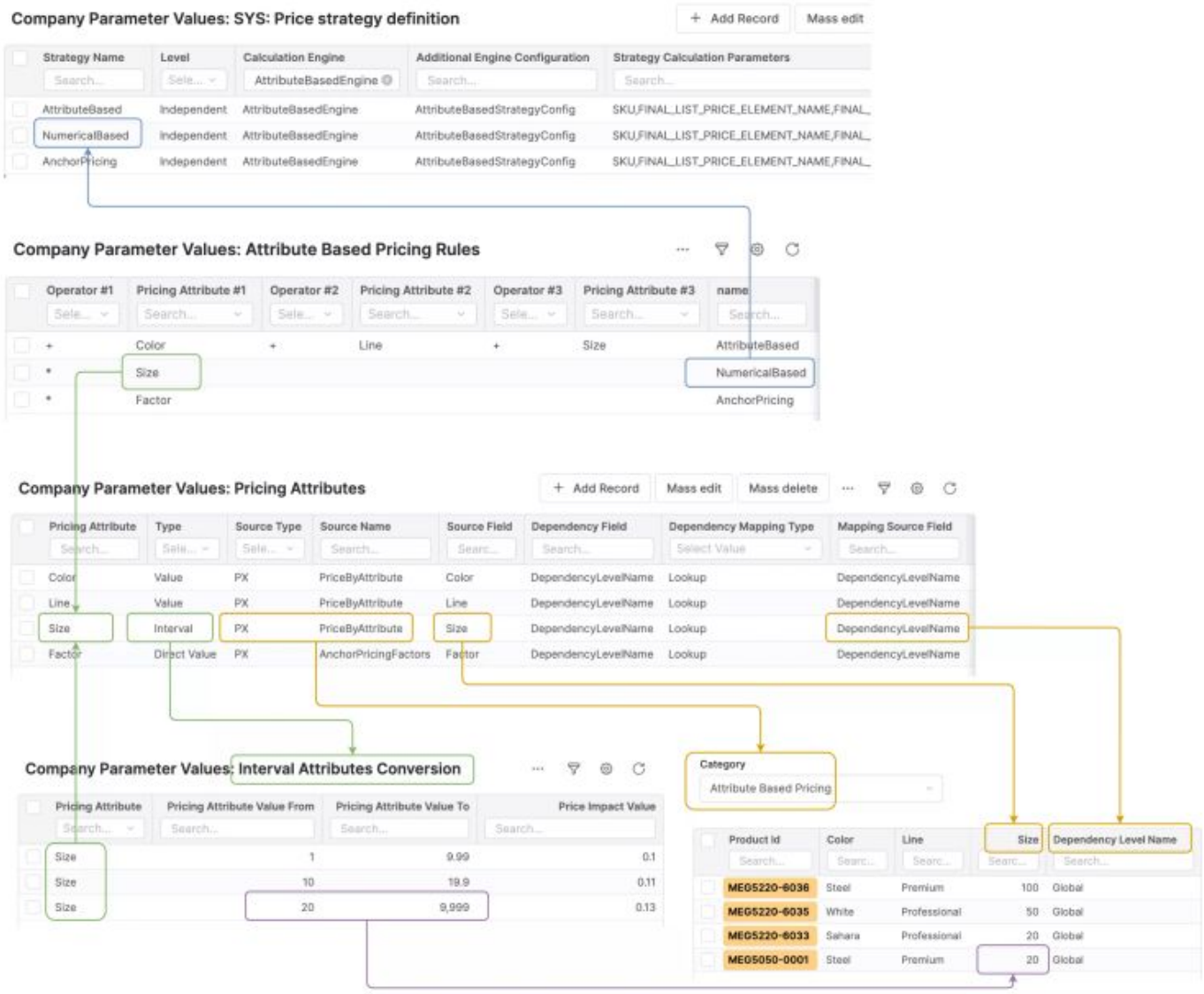
Value Based Attribute

When you define the pricing rules using the Value Based attribute, you need to set up the following:



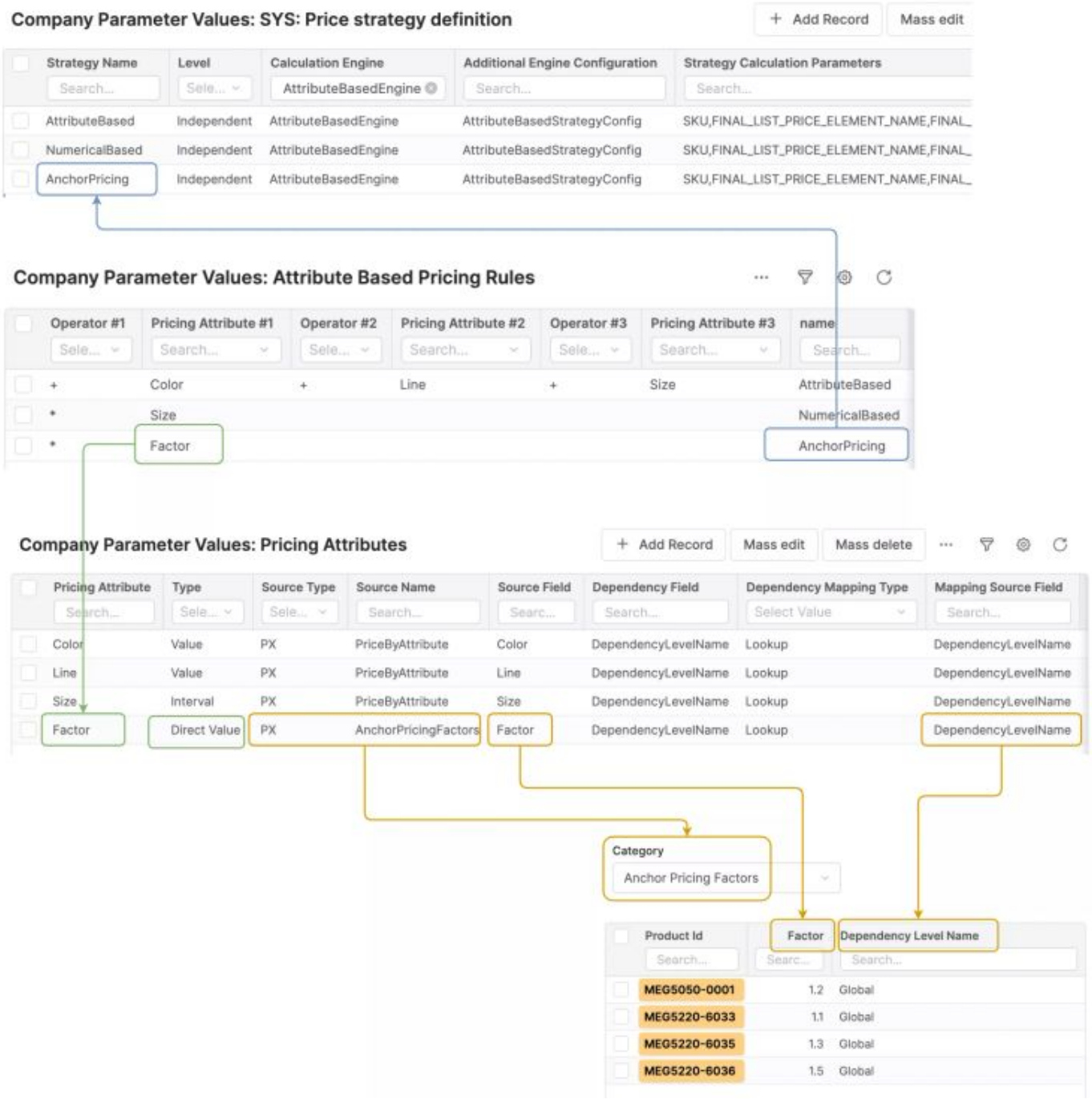
Interval Based Attribute

When you define the pricing rules using the Interval Based attribute, you need to set up the following:



Direct Value Attribute

When you define the pricing rules using the Direct Value attribute, you need to set up the following:



Configure Strategy

1. Review the article Select Pricing Strategies used for Calculation of Prices.
2. Add/modify records in the "*StrategySelection" table for the required Pricing Level and for a specific combination of the ProductSegment, for which you want to use Cost+ pricing strategy.
   - Set up the "AttributeBased" strategy for a specific Product Segment combination or on "star" level.
   - Set up the proper priority of your pricing strategy – as first, second, etc. for the Product Segment.

Visually Configure Custom Pricing Strategy (Price Setting)

When you need a Pricing Strategy, which is not included in the accelerator, you can create your own in a visual way, using Strategy Designer.

Prerequisites

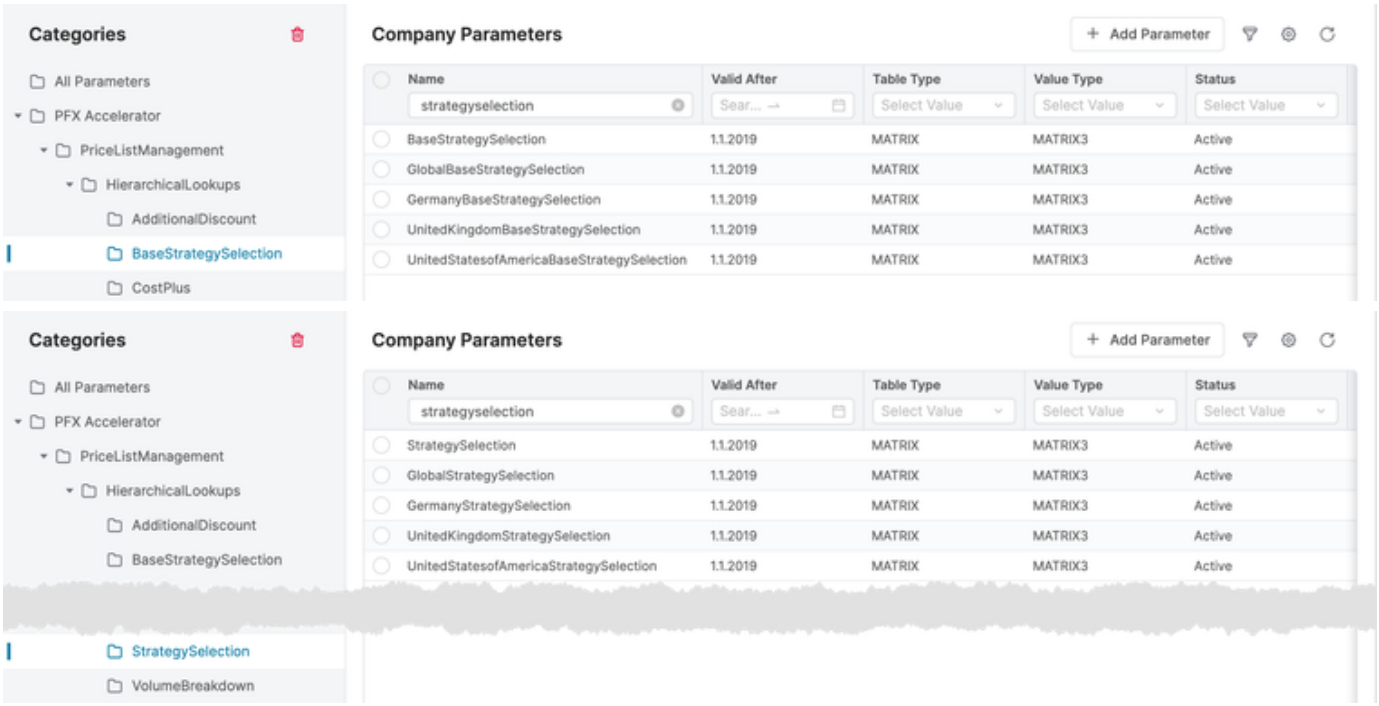
- User account with the Business Role "Strategy Designer" (to configure the strategy).
- Business knowledge of the calculation process – how you want to calculate the price.
- Data tables needed for the calculation process must be available on the partition (e.g. various Master Data tables to calculate the prices).
- Decision for which part of the business (which Pricing Levels and Product Segments) you would like to use the new Pricing Strategy.

Configuration Steps

1. To open the Strategy Designer, go to **External Applications** > **Strategy Designer**.
2. On the Design step do following:
   a. Add tables you need to read from.
      - Some tables are available out-of-the-box, and you can find them when you edit the Strategy, under the tools Product Attributes and Parameters.
      - If you need an additional table, add it on the Data Lookups tab.
   b. Design the pricing strategy calculation process. To get familiar with the tool, you can use one of the Strategy Templates.
   c. Rename the Pricing Strategy to a name which will be then used in the configuration tables (so ideally it should not change in the future).
   d. Using Live Preview, test the calculation of the pricing strategy, if you get the expected result.
      i. Use Live Preview to test the calculation for specific combinations of:
         1. Product Segmentation – This will be derived from the selected product specific.
         2. Pricing Level (i.e. Dependency Level)
         3. Target Date – Remember, the Company Parameters could exist in more copies, valid in different time frames.
      ii. If you get Validation Errors, review them and fix the issues.
      iii. If you are curious what Groovy code will be generated from the visual configuration, you can check out the Code tab, in Live Preview.
   e. Continue to the Prioritize step.
3. On the Prioritize step:
   a. You will set up for which part of the business (for which Pricing Levels and for which Product Segmentation) you would like to use your Pricing Strategy.
   b. You need to set up the priority – in which order the Pricing Strategies will be calculated for a product in the price list.
   c. This setting is the same, as described in Select Pricing Strategies Used for Calculation of Prices.
4. Deploy the strategy and the settings.
5. Recalculate (or create a new price list) to ensure the new strategy is used properly and verify the results.

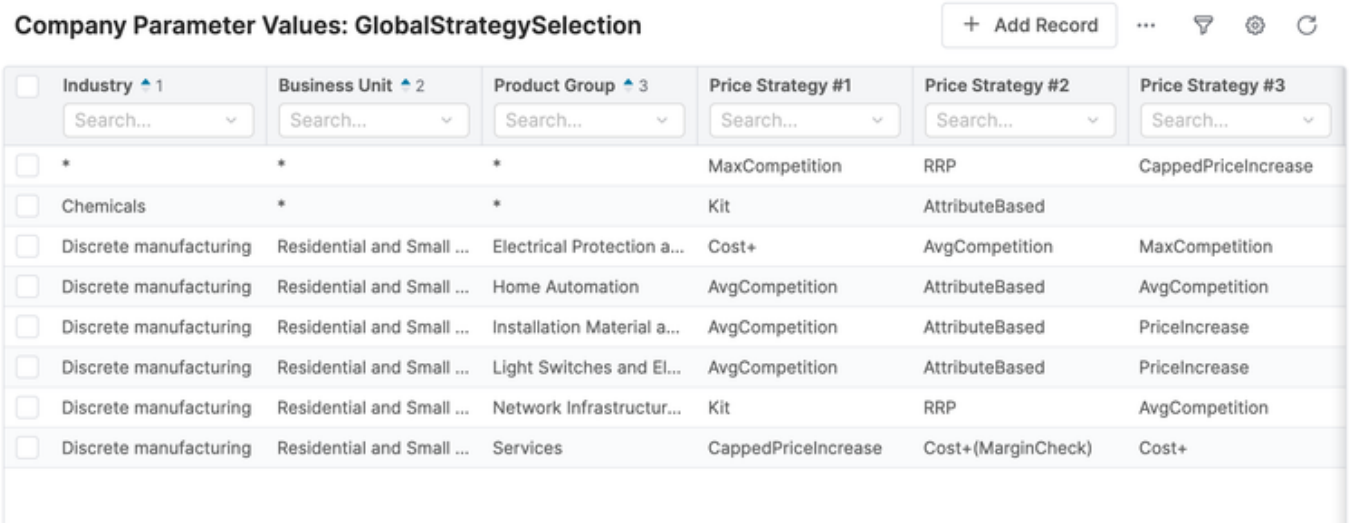
Select Pricing Strategies used for Calculation of Prices

To determine which strategy applies to a given part of the business (depending on the Pricing Level and Product Segment), use tables named "*BaseStrategySelection" and "*StrategySelection".



Steps:

1. Select which tables you want to modify – the "Base" one or the standard one. The difference is that Base strategies appear before the standard strategies. However, if they fail, they are removed from "Prices" field in the price list/grid.
2. Select for which Pricing Level you want to set up the strategy – if for the common level or for a specific level of the Pricing Level hierarchy. Modify the table for that Pricing Level.
   On an existing project, it could happen that some of the tables were not used, so the admin deleted them. If you want to create a missing one, make it as a copy of an existing (and actively used) "*StrategySelection" table.
3. Add/modify records in the "*StrategySelection" table for a specific combination of the ProductSegment for which you want to use your pricing strategy.
4. Set up the Pricing Strategy.
   - For specific Product Segment combination or on "star" level (any value of that attribute).
   - As first, second, etc. for the Product Segment. The accelerator will first try to calculate the price using the #1 strategy, and if it fails (e.g. when data are not available), it will try #2, etc.



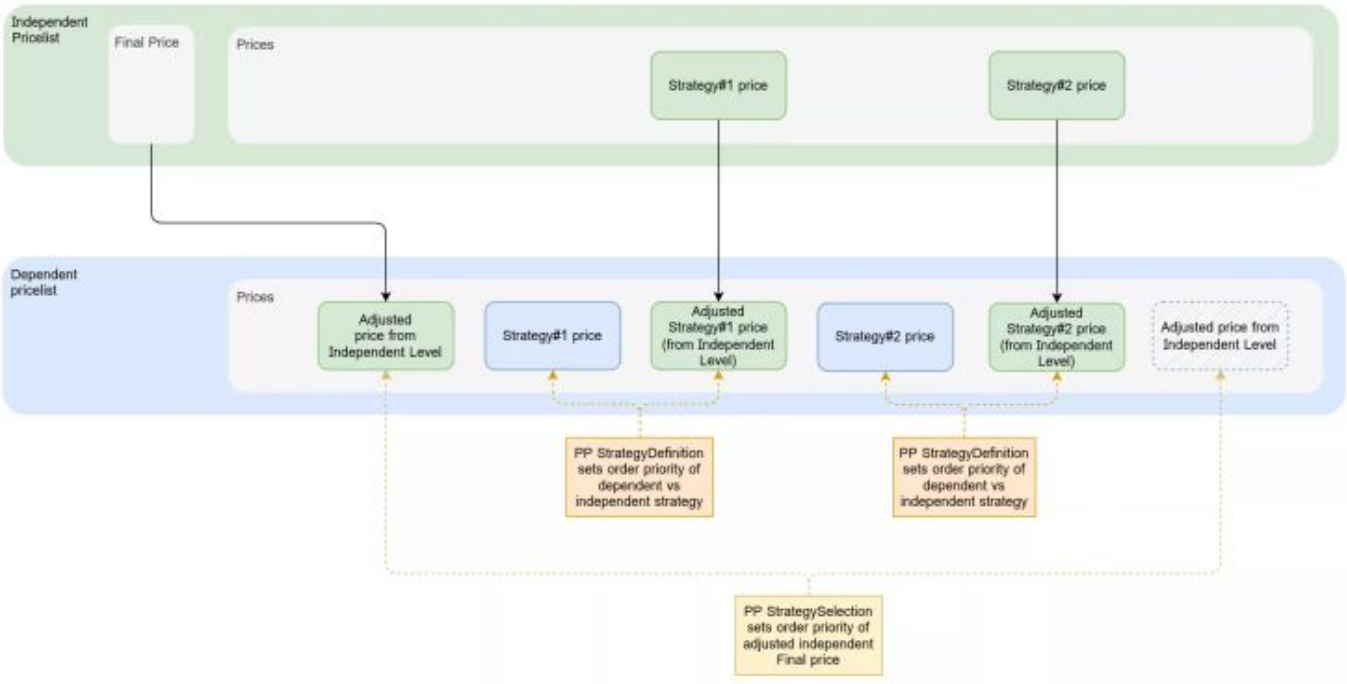
Note: Pricing strategies are defined in the Company Parameter table StrategyDefinition, so you can also find their names there.

Order of Prices in Dependent Price List/Grid

Dependent price lists/grids provide not only prices calculated for the dependent level, but also from the parent level. So you need to configure the order/priority of the prices available in the Prices field.

The order of the calculated prices is based on the following:

1. PP PricingExceptions - Can specify:
   a. Specific price for a specific product for a certain Pricing Level.
   b. Pricing strategy override - Which strategy will be used for calculation.
2. PP StrategySelection, field Prioritize Parent Level Price.
3. PP StrategyConditions - Some prices can be ignored or taken with lower priority.
4. PP StrategyDefinition
   a. field ParentLevelPriority
   b. field ParentLevelOnly - If this parent strategy could be copied and adjusted to the dependent price list. This setting is for the Parent strategy only.



Configure Dependent Pricing (Price Setting)

This article describes a scenario when you need to create a discreet price list for (e.g. different regions, countries, stores) where you need the local (e.g. country) price list to cross reference the global price list to produce the result price.

To achieve this, you will create an parent price list to calculate the global price and then several dependent price lists to calculate the prices adjusted for various local conditions.

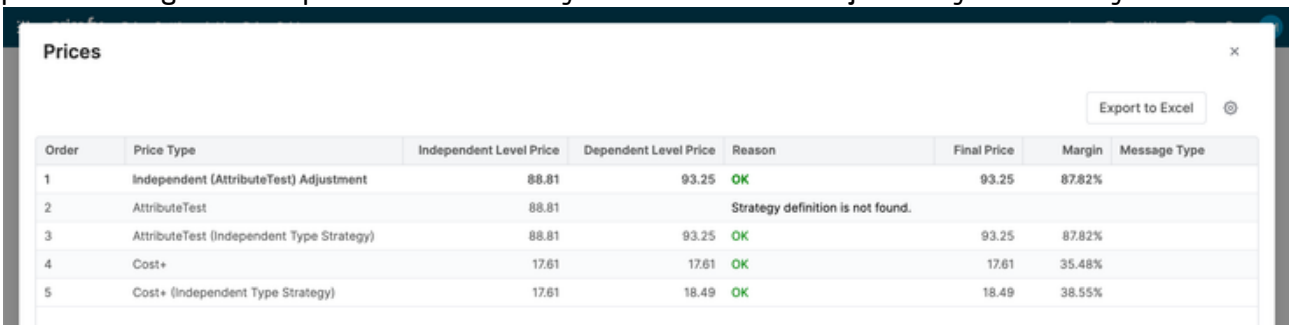
Prerequisites

You need to find out:

- which Pricing Level will represent:
  - parent level (e.g. "Global" pricing level)
  - dependent/local level (e.g. pricing levels representing various countries)
- what % adjustment will be applied to prices coming from the global price list, and for which part of the business you need what size of the adjustment - i.e. for which Pricing Levels and for which Product Segmentations.

Configuration Steps

1. Create a parent price list/grid which will represent the "global" (parent) prices.
2. Find out the ID of this price list/grid.
3. Set up the reference from the dependent/local price lists to the parent/global one.
   - This reference is a property of Pricing Levels, so you need to modify Company Parameter DependencyConfiguration PP.
   - Modify the property "Source Type" and "Source ID" for all local/dependent pricing levels in which you want to calculate the adjusted prices from the parent price list.
4. Set up the Level Adjustment % to be applied to the prices coming from parent level.
5. Set up the priority of the strategies, if it is required by the business.
6. Create the dependent/local price lists for those dependent pricing levels you identified.
7. Calculate the price lists and review the "Prices" calculated in them. You should see also additional prices coming from the parent level and they should have been adjusted by the factor you defined.



Order	Price Type	Independent Level Price	Dependent Level Price	Reason	Final Price	Margin	Message Type
1	Independent (AttributeTest) Adjustment	88.81	93.25	OK	93.25	87.82%	
2	AttributeTest	88.81		Strategy definition is not found.			
3	AttributeTest (Independent Type Strategy)	88.81	93.25	OK	93.25	87.82%	
4	Cost+	17.61	17.61	OK	17.61	35.48%	
5	Cost+ (Independent Type Strategy)	17.61	18.49	OK	18.49	38.55%	

Architecture Components (Price Setting)

After you installed the Price Setting accelerator, you will find many logics, tables and other setting on the partition. This article lists all the components installed.

Core Components

Library Logics

- PSP_ConfigWizardCommonLib
- PSP_ConfigWizardScreenFactory
- PriceBuilderCommonElementUtils

Calculation Flow Logics

- CF_BuildPricingTables - buildDynamicTables. Triggered from the installer.
- CF_PlatformPostprocess - Postprocess-Platform_Manager_Input. Triggered from the installer.

Generic Logics

- DependentPriceListLogic
- ParentPriceListLogic
- PSP_ConfigWizard
- PSP_ConfigWizardExecutor
- PriceSettingPackageInputConfigurator

Matrix Logics

- VolumeBreakdownMatrixLogic

Company Parameters

- PriceSettingDimensions (including preferences)
- DependencyConfiguration (including preferences) - Uploaded during the installation.
- WarningConfig (including preferences)

- temporaryBootstrappingConfig - Used during installation. Contains general hierarchic keys, per-feature hierarchic keys.
- AdditionalDiscountTempHook
- AdjustedPriceCorridorTempHook
- BaseStrategySelectionTempHook
- CostPlusTempHook
- CostSelectionTempHook
- DependencyLevelAdjustmentTempHook
- DiscountTempHook
- ListPriceCorridorTempHook
- MinMarginTempHook
- PriceIncreaseTempHook
- RelevantCompetitionDataTempHook
- StrategySelectionTempHook
- VolumeBreakdownTempHook

Other Components

Company Parameters

- AnchorAdditional Config (including data)
- AnchorData
- AttributeBasedPricingRules

- AvgCompetitionAdditionalConfig (including data)
- CostPlusAdditionalConfig (including data)
- CostTypeDefinition
- DependencyMappingConfig (including data)
- ExchangeRates
- IntervalAttributesConversion
- MarginAlertFlagTempHook
- MaxCompetitionAdditionalConfig (including data)
- MinCompetitionAdditionalConfig (including data)
- PriceIncreaseAdditionalConfig (including data)
- PriceSettingConfig (including data)
- PriceSettingLevel
- PriceSettingModules (including data)
- PricingAttributes
- PricingExceptions
- PromotionLookupEngineConfig (including data)
- RRPLookupEngineConfig (including data)
- RoundingRulesConfig (including data)
- StockData
- StrategyConditions
- StrategyDefinition (including data)
- ValueAttributesConversion
- VolumeBreakdownExceptions

Product Extensions

- ListPrices
- ProductCosts
- PromotionPrices
- RecommendedRetailPrices

Configuration Wizard

- PSP_ConfigurationWizard, using logics:
  - PSP_ConfigWizard
  - PSP_ConfigWizardExecutor

Dependencies

This accelerator depends on the following accelerators which will be deployed during the installation too.

- Calculation Engines Library
- Shared Library

Technical User Reference (Price Setting)

- Customization Tiers (Price Setting)
- Common Customization Tasks
- How to Extend the Accelerator (Price Setting)
- Configuration (Price Setting)
- Common Configuration Procedures (Price Setting)

- Technical Information (Price Setting)

Customization Tiers (Price Setting)

If the customer project customizes the accelerator's logics, it happens in one of the following tiers:

- Tier 1 - User Facing Logics
- Tier 2 - Business Logic Libraries, Supporting Logics
  - Common Logic Libraries
  - Supporting Logics
- Tier 3 - Shared Library / Util Logics

Generally, customization should be done at the top tier, otherwise potential upgrade of the accelerator becomes very complicated.

Tier 1 - User Facing Logics

Tier 1 logics define what data are shown to the user in a price list/grid together with some hidden supporting elements. Logics are implemented (as much as possible) as simple "dispatchers", to gather required values (inputs) from previous elements and dispatch the calculation to Tier 2 logic.

If you need to make changes, we recommend that they are made on this level, rather than on other levels, because it is the really easy to swap the dispatching code to use/call some custom library.

Upgrades should not cause any complex conflicts even if the underlying Accelerator business logic changes and re-applying your modifications should be easy.

Logics:

- ParentPriceListLogic
- DependentPriceListLogic
- VolumeBreakdownMatrixLogic

Tier 2 - Business Logic Libraries, Supporting Logics

Common Logic Libraries

This is where most Accelerators business requirements are implemented. The libraries are full of detailed utilities that calculate and prepare data for Tier 1 logics. This is where most of the package complexities are handled.

Changes at this tier are possible but not particularly encouraged.

Upgrades of accelerators with modifications in this tier can end up in resolving many conflicts and the process can be quite demanding.

Logics:

- PriceBuilderCommonElementsUtil

Supporting Logics

These are all the remaining logics that do not directly affect business requirements.

Modifications of these logics are usually not necessary.

Upgrades of accelerators with changes in this tier can end up in resolving many conflicts and the process can be quite demanding.

Logics:

- PriceSettingPackageInputConfigurator
- PSP_ConfigWizard
- PSP_ConfigWizardCommonLib
- PSP_ConfigWizardExecutor
- PSP_ConfigWizardScreenFactory

Tier 3 – Shared Library / Util Logics

These are calculations and utilities that are designed to work also independently from this accelerator. For example, Calculation Engines or Pricefx Groovy Library (SharedLib).

Changes in this tier are not recommended.

Logics:

- CalculationEnginesLib
- SharedLib from Shared Groovy Library

Common Customization Tasks

- Add New Column to Price List/Grid (Price Setting)
- Remove Column from Price List/Grid (Price Setting)
- Modify Calculation or User Experience of Existing Columns in Price List/Grid (Price Setting)
- Make Editable Price List/Grid Column (Price Setting)

Add New Column to Price List/Grid (Price Setting)

To implement project requirements, you may need to introduce extra columns in the price list/grid tailored to business users. For instance:

- Incorporate product hierarchy fields
- Add new price points alongside Final Price, such as:
  - Final Price Rounded
  - Final Price in EUR
  - Final Price including VAT
- Introduce a new price point preceding the Final Price

To address these requirements, modifications to the price list logic are necessary. This involves adding new logic elements and potentially adjusting existing ones.

Steps

1. Create a new price list logic.
   a. Make a copy of the Price Setting Accelerator logic you need to change – most likely the parent or dependent one. (In future releases, you will also be able to use "logic inheritance" in Studio, instead of copying the logic.)
      i. Give it a different name.
      ii. Set its Valid After date based on the guidelines in your project.
   b. Change the code of the ActualPriceLookup element to reflect the new name of the logic.
2. Add the new logic element.

a. Remember to follow best practices, e.g. the code should fail gracefully, so use the Warning Manager. Example of a code template:

```
return out.WarningManager.tryToExecuteElement
("YourElementName") {
    // Your code goes here
}
```

b. Give the element an appropriate name.
    i. Review the naming conventions of existing elements, and follow the patterns. For example, if you have only a "calculation" element, whose result is not presented to users, but is used in some later elements, name it "Raw...".

3. If the result value from you new element needs to be considered also in calculation of other existing elements (e.g. in Final Price), modify also the implementation of these elements to use your element results.
4. Re-configure the price list/grid.
    a. If it is expected to be displayed to end users, make your new element visible in the calculated price list.
    b. Ensure the option "Allow column type change" is switched on in the LPG definition.
    c. Ensure the price list uses the new logic.

Remove Column from Price List/Grid (Price Setting)

In your project implementation, you may not need all of the results calculated out of the box by the price list logic included in Price Setting Accelerator. In such case, you can consider their removal.

Steps

1. Some of the elements are calculated only when a certain Price Setting Accelerator module is switched on. So maybe it will be enough to simply switch off such module, and you can keep the element as is, because it will not be executed anyway.
    a. To find out, review the element's "Condition" property, if it checks that a module is switched on or off.
    b. If you can solve the situation only by switching the module off, switch it off and you are done.
2. In case the element is always executed, or when the element is part of a module which you cannot switch off because you need some of its other features, then you can consider its removal.
    ⚠ Before you remove it, double check that its value is not used in any subsequent elements. If so, then you may only "hide" it, but do not remove it.
3. Remove the element.
    a. Soft removal – Only "disable" the execution of the element, but keep the code. Change the value of the element's property "Condition" to "false". This is a good solution, if you think that you may need to put the code back later, as it will be very easy to revert the change.
    b. Hard removal – Remove the element completely from the logic.
4. Test if your price list/grid still calculates all things as needed, and does not fail for some products.

Modify Calculation or User Experience of Existing Columns in Price List/Grid (Price Setting)

It may happen that you need to change the behavior of an existing price list column/element. For example, you may want to change:

- Implementation:
  - How prices are calculated
  - How data are retrieved – e.g. how you load Competition Data
  - Adjust parameters of some operations – e.g. change the code of Actual Price Lookup
  - Make column values overridable
- User experience:
  - Highlight the whole column with a color – this is appreciated by customers because they can easily differentiate between the prices in large pricelists
  - Color only specific values in a column – to easily spot problematic values
  - Add an icon to the label of the column – e.g. add a "writing hand" icon (it is a letter in UTF-8) to the labels of columns whose values can be overridden
  - Add an icon to the suffix of the specific value – e.g. to add an up-arrow to the suffix of rising values

Steps

1. Make a copy of the out-of-the-box Price Setting Accelerator logic which you plan to change. (In future releases, you will also be able to use "logic inheritance" in Studio, instead of copying the logic.)
   a. Give it a different name.
   b. Set up the Valid After date.
   c. Change the code of the ActualPriceLookup element to reflect the new name of the logic.
2. Modify/replace the element's implementation code inside the "tryToExecuteElement" block.
   a. Ensure that your code returns the same type of data as was returned in the original implementation. For example, if it was a Map, it must be a Map again with the same keys and the same types of values.
   b. When the element output is a simple value, such as number, which is displayed to users in the price list, you can modify its presentation by using api.attributedResult().
3. Modify the element's properties.
   a. You can change the colors of the element.
   b. When modifying the suffix, ensure that it has the same meaning. For example, if it was displaying a currency, then it should still contain a currency, but potentially with some additional information (like the "up-arrow" and similar).
4. Test the changes in your price list/grid.

Make Editable Price List/Grid Column (Price Setting)

The customer may need to make some of the columns in the Price List/Grid editable, so that it is possible to override the values calculated by the calculation logic.

Steps

1. Check if the columns you intend to make editable are supported by any available out-of-the-box features.
   a. The Price Setting Accelerator module "Override" allows product price and strategy to be editable. Review the page Fields Provided in Price List/Grid (Price Setting), fields in Override module:
      i. Price Selector
      ii. Override Price
      iii. Override Reason
      iv. Changes

b. Field Manual Override – This field is editable in Pricefx out-of-the-box, and is used to override the value of the Result Price column. If this fits to your business use case, you can use it.

2. If not found in the out-of-the-box features of the accelerator, you need do change the price list logic using the standard Pricefx Core configuration features (see Make Price List/Grid Column Editable) and make the column editable via the logic.

How to Extend the Accelerator (Price Setting)

This section contains "How To..." articles focusing on extending Price Setting Package.

- How to Add and Modify Columns in Price List/Grid
- How to Create and Set up New Price Calculation Strategy
- How to Modify Existing Price Calculation Strategy
- How to Create Publishing Template
- How To Set up Custom Dimension Lookup
- Configurable Lookups Guide

How to Add and Modify Columns in Price List/Grid

Adding and modifying existing PL / LPG columns is one of more common customizations that can be applied to Price Setting Accelerator. This is why most elements in user-facing PL / LPG logics act as "dispatchers" for more complex operations. It makes it easier to add custom behavior or modify one step of the calculation without affecting other parts of the underlying logic. You can find more information about this approach in Upgrade of Customized Accelerator (Price Setting).

Currently there are two recommended ways to modify available columns and this article will cover both of them:

1. Logic Inheritance
2. Standard Groovy Modifications

⚠ Logic Inheritance is a relatively new feature and as of July 2023, there is no full Pricefx Studio support which makes it harder to move changes between partitions (PFAUT-717 - Logic inheritance in Studio TO DO).

Prerequisites

This article assumes that:

- Price Setting Accelerator is deployed to the partition and you have a working Live Price Grid configured with `ParentPriceListLogic` where you can test your changes. Check Install Price Setting Accelerator for more info.
- You are a Configuration Engineer or have sufficient knowledge of Groovy coding, Pricefx Studio and managing and linking libraries of code to undertake the technical configuration.

Configuration Overview

User facing logics are called `ParentPriceListLogic` and `DependentPriceListLogic`. They are pretty similar because they share a lot of features. These shared features, along with other utils, are defined mostly in `PriceBuilderCommonElementUtils` Groovy library.

Let's take a look at a simple `CompetitionData` element from `ParentPriceListLogic` and try to understand what we willl be working with:

```
/**
 * Inputs:
 * - RawCompetitionData
 * Outputs
 * - Matrix of competition data
 * Details:
 * - This element is conditional, Competitions module needs to be
turned on
 * - This is one of "End points" of our logic, return is displayed to
user
 * - All competition data is displayed. For relevant competition data,
see RelevantCompetitionData element
 */
return out.WarningManager.tryToExecuteElement("CompetitionData") {
    Map productCompetitionConfigManager = out.
ProductCompetitionConfigManager
    def competitionData = out.RawCompetitionData

    return libs.PriceBuilderCommonElementUtils.Library.
buildCompetitionDataResultMatrix(productCompetitionConfigManager,
competitionData)
}
```

Most non-utility elements have a comment that explains what the required inputs are (other elements or configurations) that are used by a given element. They will also explain what the structure of outputs is. Details will contain additional information that may be useful for understanding what is going on in this element.

To support modularization and proper warning handling, all elements will go through this pass-through `tryToExecuteElement` method. What is important for you is that the body of this method is executed as if it was a normal element code. For more info check out Warning Handling and Error Handling Deep Dive.

Most calculations and business logic will be delegated into a library method, especially if it is common for parent and dependent calculation.

All elements that produce PL / LPG columns visible for users will have this structure, so usually this is what you will be looking for during analysis of your requirements.

Further reading:

- Logic Inheritance
- Standard Groovy Modifications
- Tips on Logic Customization

Logic Inheritance

In Pricefx, there is a feature called *logic inheritance* which allows you to:

- create/modify elements of the accelerator logic to suit your use case; but

- keep the accelerator's out-of-the-box logic unchanged, so you can easily upgrade to future versions.

See also documentation about Parent Logic.

> As of July 2023, Pricefx Studio support of logic inheritance is on the way, so in this article, we refer to Pricefx user interface for building logics (PFAUT-717 - Logic inheritance in Studio TO DO).

In this section:

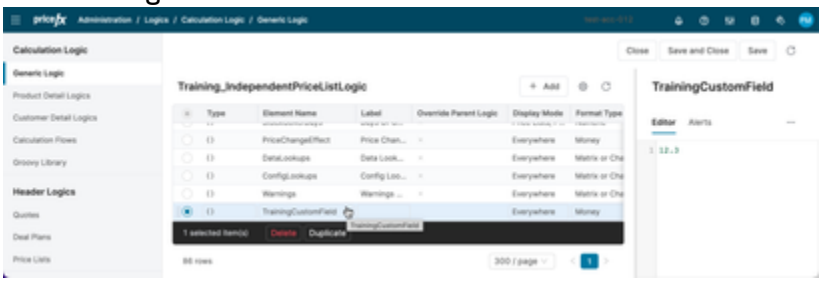
- Add Custom Field to Price List/Grid
- Modify/Override Behavior of Existing Field

Add Custom Field to Price List/Grid

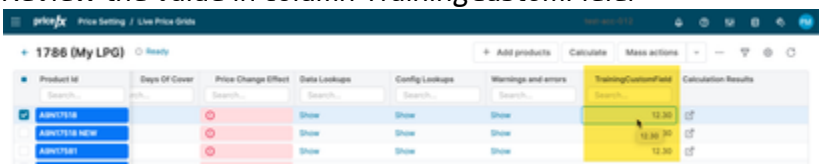
1. Navigate to **Administration > Logics > Generic Logics**.
2. Add a new generic logic:
   - **Name** - Select *Training_ParentPriceListLogic.*
   - **Parent Logic** - Select *ParentPriceListLogic.*
     Note: This dropdown field contains all the existing calculation logics available within the system and by selecting one, the system will transfer all the fields and logic calculations to your new logic.
   - **Valid After** - Set it based on guidelines in your project, or now, for test purposes, simply use January 1st of the actual year.
   - **Status** - Set it to *Active*.
3. Review the logic content.
   - When you edit your new logic, you will see all the elements inherited from the parent logic.
   - When looking at the code of some inherited element, you will see a message that this element is inherited, i.e. its code comes from the original parent logic and we can eventually overwrite that code within this new logic.
4. Add a new element. It will be added at the end of the list of elements.
   - **Type** - Select *Groovy (Function)*.
   - **Name** - *TrainingCustomField*
   - **Display Mode** - *Everywhere*
     Caution: Remember to uncheck the option *Never.*
   - **Format Type** - *Money*
   - Enter this code:

```
12.3
```

   - Note that the code editor of this element does not have the inheritance indicator because its code is defined within this logic.
   - Save the logic.



5. Test the calculation.
    a. Recalculate your Live Price Grid.
    b. Review the value in column *TrainingCustomField*.



See Tips on Logic Customization for more details on configuring a test LPG.

Modify/Override Behavior of Existing Field

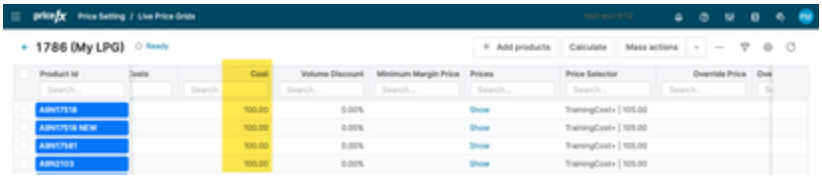
When you open your inherited logic *Training_ParentPriceListLogic*, there is an element named *Cost* which has the code/behavior inherited from the logic *ParentPriceListLogic*.

You can override the out-of-the-box behavior of the accelerator and calculate *Cost* in a different way. But the out-of-the-box accelerator logic will stay unchanged because the change will only be in your inherited logic.

1. Find and edit your generic logic *Training_ParentPriceListLogic*.
    a. Select the element *Cost*.
    b. In the Editor, use the **Override** action.
    Note: This will copy the element's content from the parent logic to your logic. You can modify it and perform any operation and lookup you need to gather the cost in the way you need.
    c. Comment out the original code.
    d. At the very beginning of the element code, insert your own implementation:

```
return 100.0
```

    e. Save the logic.
2. Test the calculation.
    a. Recalculate your Live Price Grid.
    b. Review the value in column *Cost*.



Standard Groovy Modifications

- Add Custom Field to Price List/Grid
- Modify/Override Behavior of Existing Field

Add Custom Field to Price List/Grid

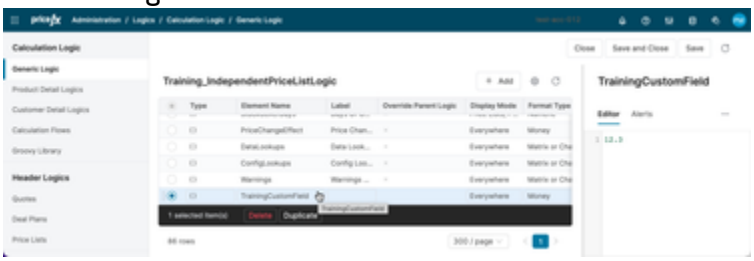
1. Navigate to **Administration > Logics > Generic Logics**.
2. Select *ParentPriceListLogic* and duplicate it.
    - Rename it to "Training_ParentPriceListLogic".

- **Valid After** - Set it based on guidelines in your project, or now, for test purposes, simply use January 1st of the actual year.
- **Status** - Select *Active*.

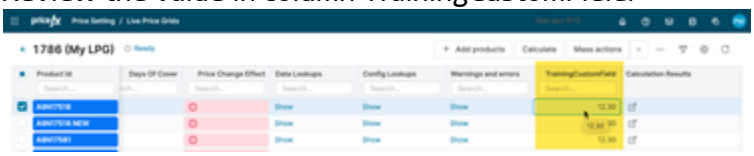
3. Add a new element. Open the logic and set the following:
   - **Type** - Groovy (Function)
   - **Name** - "TrainingCustomField"
   - **Display Mode** - *Everywhere*
     Caution: Rremember to uncheck the option *Never.*
   - **Format Type** - *Money*
   - Enter this code:

```
12.3
```

- Save the logic.



1. Test the calculation.
   a. Recalculate your Live Price Grid.
   b. Review the value in column *TrainingCustomField*.



Check Tips on Logic Customization for more details on configuring test LPG.

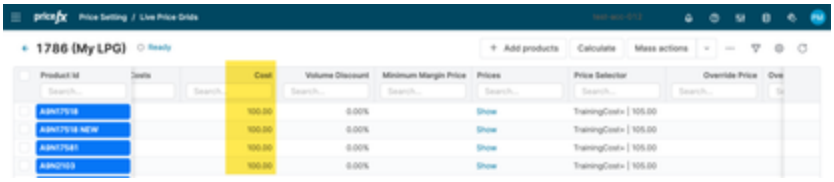
Modify/Override Behavior of Existing Field

When you open your logic *Training_ParentPriceListLogic*, there is an element named *Cost*.

1. Find and edit your generic logic *Training_ParentPriceListLogic*.
   a. Select the element *Cost*.
   b. Comment out the original code.
   c. At the very beginning of the element code, insert your own implementation:

```
return 100.0
```

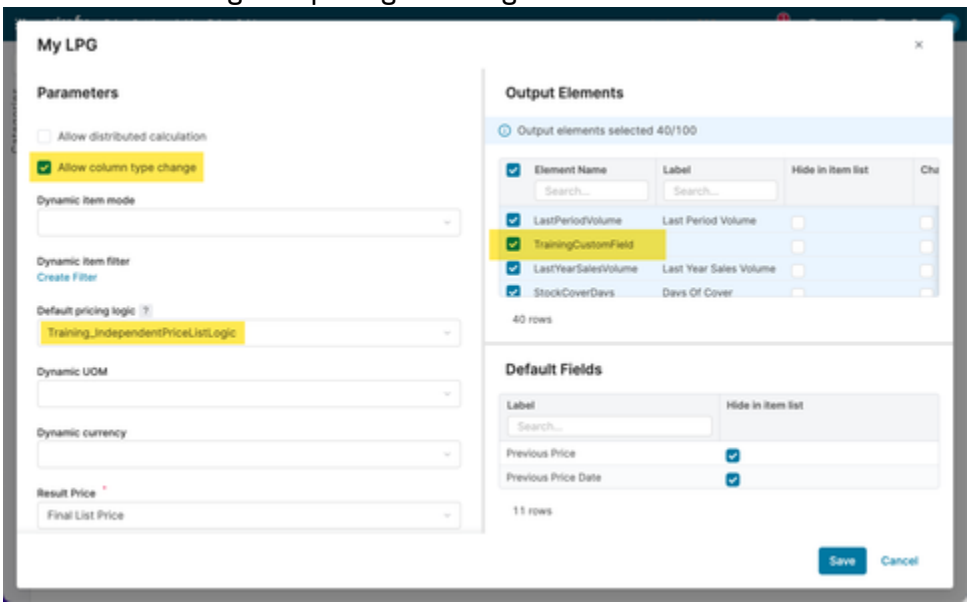
   d. Save the logic.
2. Test the calculation.
   a. Recalculate your Live Price Grid.
   b. Review the value in column *Cost*.



Tips on Logic Customization

Configure Your Custom Logic

1. Locate Live Price Grid you want to use for testing your changes. If you do not have one, create a new one.
2. Update the price grid configuration.
   a. Select the Live Price Grid, and use **Configure** to open the configuration dialog for the existing price grid.
   b. In the price grid configuration, update several settings:
      i. **Allow column type change** – Select YES.
      Note: This is needed because we added a new field and so we need to regenerate the meta-data about columns during full recalculation of the price grid.
      ii. **Default Pricing Logic** – Select *Training_ParentPriceListLogic.*
      iii. **Parent Level Name** – Select *Global* (it may be different in every installation, so use whatever dependency level name you have configured during deployment).
      Note: When you change the default logic, the content of this input field will be wiped, so you need to set the value again.
      iv. **Output Elements** – Make sure that the new element named *TrainingCustomField* is checked too.
   c. Save the changes in price grid configuration.



Approach to Modifications

All Price Setting Accelerator modifications best practices are built on top of modularization-first approach. Thanks to that, it should be relatively easy to replace parts of the accelerator implementation. E.g. if you want to change how you load competition data, you can just replace the code inside `tryToExecuteElement` in the `CompetitionData` element and the rest of the app will still work fine as long as the structure you return is the same as the original one.

One thing to keep in mind is how to modify behavior of the `PriceBuilderCommonElementUtils` library. It is usually safer to create your own library with your own version of the util and call it from the element in a parent/dependent logic. It will make it easier to re-apply the changes if you want to upgrade the accelerator. Also, it will make it much easier to check if issues you are experiencing are related to the accelerator or your custom code.

Element Structure

Both extension methods do not restrict how you write your code, so you can develop whatever is in your requirements. That is why the example code is very basic. The point of this article is to show you how to plug it in into the accelerator.

You can completely ignore the `tryToExecuteElement` method, but if you want your code to fail gracefully, the recommended structure for a new or modified element is:

```
/**
 * Ideally some doc should be here
 */
return out.WarningManager.tryToExecuteElement("YourElementName") {
    // Your code goes here
}
```

Actual Price Lookup - Special Case

⚠ Both approaches suggest cloning Price Setting Accelerator logic. While doing so, it is important to pay attention to the element *ActualPriceLookup* which retrieves the actual price of the product.

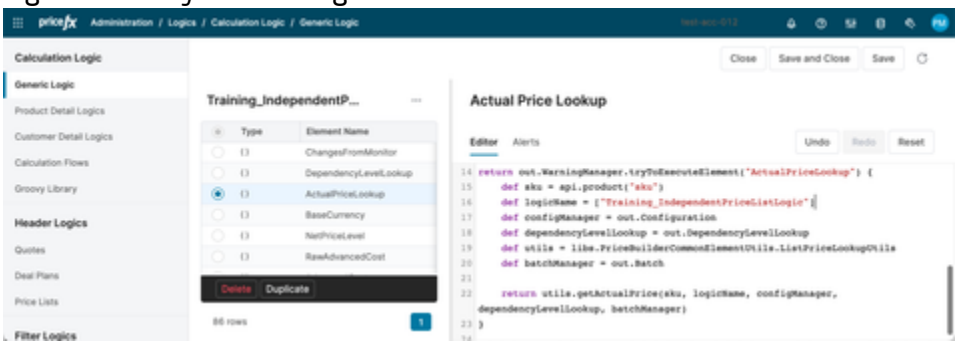
There are different modes to get the actual price:

- **Live Price Grid mode** – Commonly used mode in Live Price Grids. It takes the last approved price of the Live Price Grid, and if there is no approved one (e.g. because you are in the initial state), it will take the actual or the new price.
- You can store the actual price in a **Product Extension or other data source**, e.g. if you have an ERP, you usually have your reference price list there.
- From the **latest approved price list of your level**. This is used mostly when working with price lists (and not the price grids) where you want to make a new price list for a new season or next year, and you want the actual price to be taken from the year before. So it will always find the latest approved price list. When using dependency levels (like global and countries) it will always use the same dependency level – i.e. when calculating a new global price list, it will always look for the last approved global price list.

The last of the mentioned modes is the reason why you need to override the *ActualPriceLookup* element. This is how to do it:

1. Modify the name of the logic in the element *ActualPriceLookup.*

a. Edit your cloned/inherited version of *ParentPriceListLogic.*
b. Select the element *ActualPriceLookup.*
c. Find the line of code which defines the logic name to be *ParentPriceListLogic* and change the logic name to your new logic name.



d. Save the logic.

2. Test the change. To be able to test it, you need to simulate the scenario above which is actually using this logic name.

How to Create and Set up New Price Calculation Strategy

Price Setting Accelerator was created with a "plug-in" architecture in mind. It means that you can safely add your own custom price strategy calculations and they will be treated the same way as existing out-of-the-box strategies. All the other features of Price Setting Accelerator will work the same way for them.

This article is a step-by-step guide for creating and configuring you own custom strategy.

> Strategy Designer works on top of the mechanism that this article describes. Please check it out before proceeding, because it may solve your use case in a more user friendly way.

Prerequisites

This article assumes that:

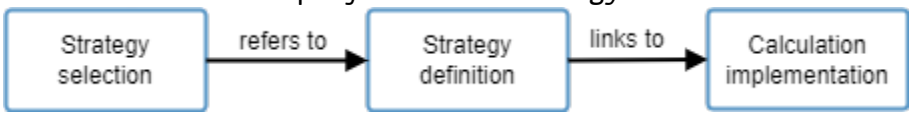
- Price Setting Accelerator is deployed to the partition and you have a working Live Price Grid configured with `ParentPriceListLogic` where you can test your changes. Check Install Price Setting Accelerator for more info.
- You are a Configuration Engineer or have sufficient knowledge of Groovy coding, Pricefx Studio and managing and linking libraries of code to undertake technical configuration.

Configuration Overview

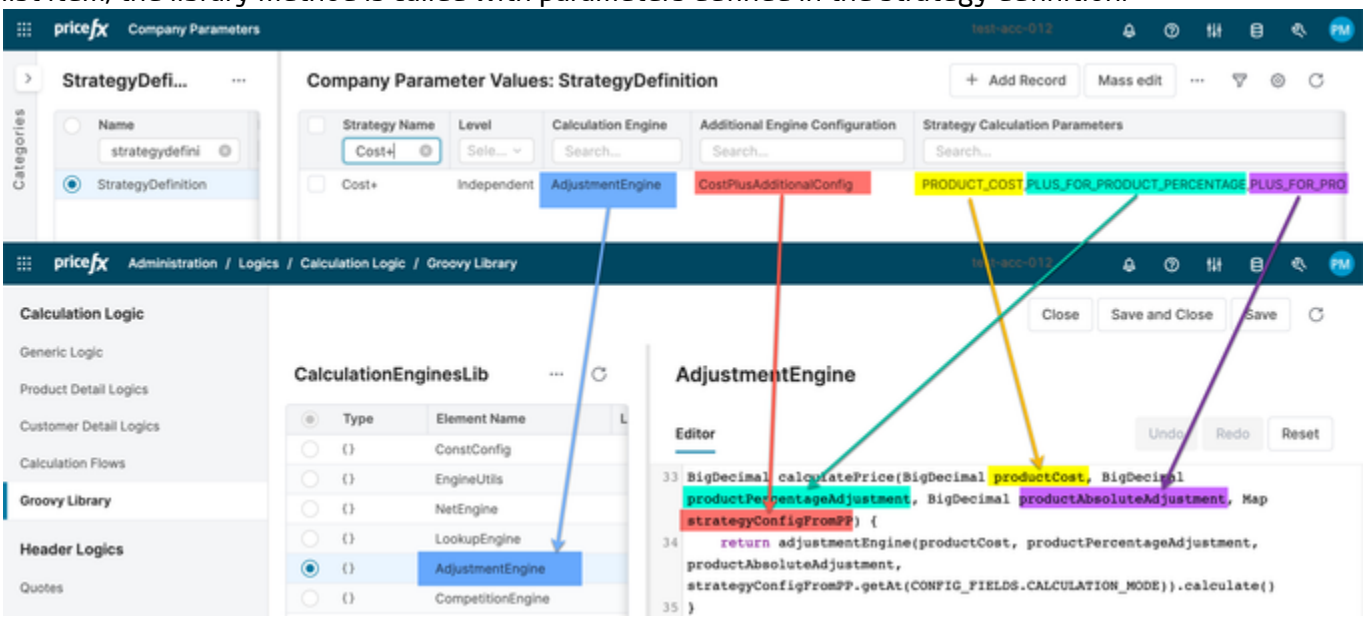
Each strategy is configured at two places:

- **Strategy Implementation** - Groovy code stored in a method in the *Library* type logic. This is source of truth for both out-of-the-box and custom strategies. For details see Custom Engines.
- **Strategy Definition** - Record in the Company Parameter *StrategyDefinition* which defines the name and parameters to be provided when the Price Setting Accelerator calls the method. For details see StrategyDefinition PP.

The administrator can set up the Price Setting Accelerator to use the strategy for the price list calculations in the Company Parameter *StrategySelection*. For details see Strategy Selection Lookup.



For example, review the following definition of the out-of-the-box strategy *Cost+*. You can see the relation between the *definition* and *implementation*. When Price Setting Accelerator calculates the price list item, the library method is called with parameters defined in the strategy definition.



See Adjustment Engine for more info about the out-of-the-box *Cost+* calculation engine parameters.

Steps to create a custom pricing strategy:

- Create New Calculation Engine Implementation
- Define New Pricing Strategy
- Link Pricing Strategy to Business Units or Products
- Test Calculation using Custom Pricing Strategy
- Set Messaging from Calculation Logic
- Add Built-in Parameter to Strategy Call
- Add Custom Parameter to Strategy Call

Create New Calculation Engine Implementation

First, we need to create a Groovy library to have a basic function returning something to confirm that the new strategy works at all, before we properly define the Calculation Engine.

1. Create a new *Groovy Library* logicˇ.
   - We will use "TrainingCalculationEngineLib" name in this guide, but it can be any library logic.
   - Set **Valid After** to a date in the past (e.g. beginning of the year).
   - Set **Status** to *Active.*
2. Add a new *Groovy* element "TrainingEngine" (or any other) with the `calculatePrice()` method and very simple implementation.

```
BigDecimal calculatePrice() {
    return 5.0
}
```

   For now, it has no parameters (we will discuss parameters later) and it will simply return a price of 5.0 as a result.

Next step: Define New Pricing Strategy

Define New Pricing Strategy

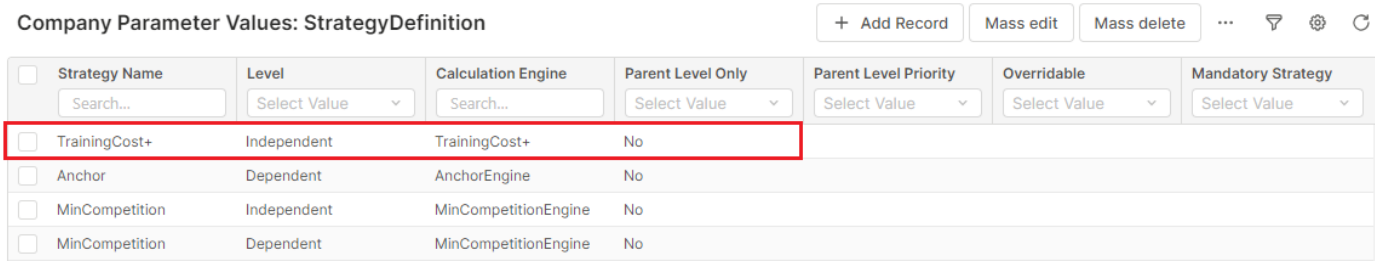
The Company Parameter *StrategyDefinition* contains a list of pricing strategies that are available in the price setting package, along with details about the strategic hierarchy, calculation engine and strategy calculation information.

Let's explain some of fields:

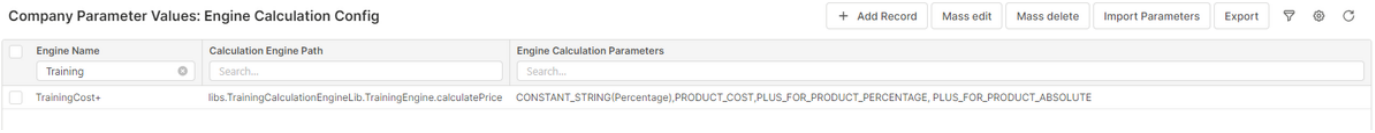
- **Level** - When you create a new pricing strategy, you must consider the *level* that the strategy will operate at. In the *StrategyDefinition* parameter you can see that most Pricing Strategies have two levels - *Parent* for global scope and *Dependent* level for regions, countries or even branches set in your hierarchy.
- **Calculation Engine** - This is an attribute of the Pricing Strategy where we have existing Calculation Engines, such as *AnchorEngine*, *CompetitionEngine* etc. All of them are listed in Calculation Engines. In other words, this is a name of the engine or a path where the custom engine implementation can be found.
- For more details, see Strategy Definition parameter.

To define the new strategy:

1. Find the Company Parameter *StrategyDefinition* and add a new record to define a new strategy:
   - Set **Strategy Name** as *TrainingCost+* (mandatory field).
   - Set **Level** to *Parent* (mandatory field).
   - Set **Calculation Engine** to a path to locate your method, i.e. set it to "libs.TrainingCalculationEngineLib.TrainingEngine.calculatePrice". This is the same pattern which is used in Groovy logic when you call a library method.
     Note that this is different from definitions of the out-of-the-box strategies.



1. Find the Company Parameter *PSP_EngineCalculationConfig* and add a new record to define a new strategy implementation in Groovy library. For more details, see Engine Calculator Library.
   - Set **Engine Name** as *TrainingCost+* (mandatory field).
   - Set **Calculation Engine Path** to a path to locate your method, i.e. set it to "libs.TrainingCalculationEngineLib.TrainingEngine.calculatePrice". This is the same pattern which is used in Groovy logic when you call a library method.
   - Set **Engine Calculation Parameters** to parameters needed to calculate prices in your method.



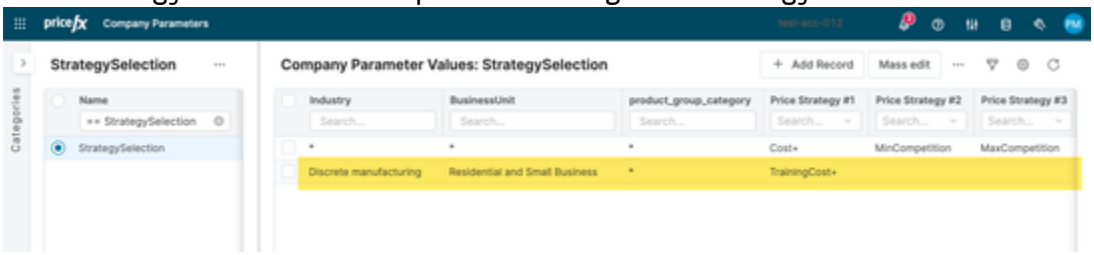
Previous step: Create New Calculation Engine Implementation

Next step: Link Pricing Strategy to Business Units or Products

Link Pricing Strategy to Business Units or Products

Steps:

1. Navigate to the Company Parameter *StrategySelection*. Here we have the various business units and product groups as well as the various pricing strategies assigned to them. It all depends on your data and Product Segmentation, so keys may be different.
   - Keep in mind that *StrategySelection* works as a general fallback, so if changes you make to it are not applied in PL/LPG, you probably have some data defined in *yourDependencyLevelNameStrategySelection* PP.
2. You can use any configuration that fits your use case. The simplest choice for testing is to use only the asterisk * because it will act as a general fallback for any product. This type of lookup is described in more detail in Hierarchical Lookups. For this article we will use:
   - Industry *Discrete manufacturing*
   - Business Unit *Residential and Small Business*
   - Other keys as "*"
   - Price Strategy #1 - here we will put our *TrainingCost+* strategy



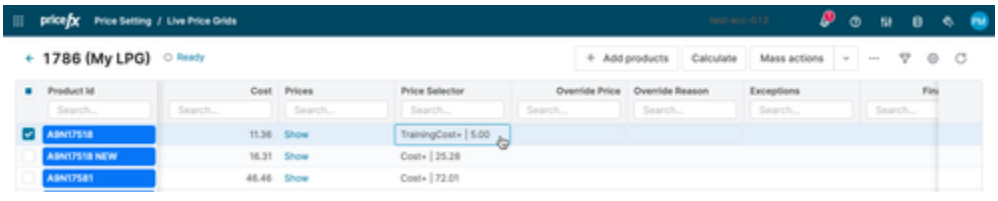
Previous step: Define New Pricing Strategy

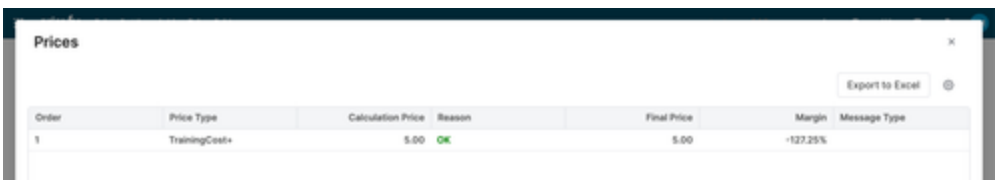
Next step: Test Calculation using Custom Pricing Strategy

Test Calculation using Custom Pricing Strategy

Now we can calculate the result price for a product that falls into *Discrete manufacturing/ Electrical Protection and Control* categories. Again - this depends on your data and configuration, so it will be different on every project. When our pricing calculation runs for any product within that Industry and Product Group, it will take this lookup key and with our new strategy it will calculate a new price, which, based on the element we created, should return a price of (result) of 5.

1. Navigate to your working Live Price Grid.
2. Make sure you have at least one product from the Industry and Business Unit specified in the *StrategySelection* parameter.
3. Recalculate that line item.
4. Review the calculated result. You should see the expected result, based on your implementation. Click the link *Show* in the Prices field to see the result in detail.





Next step: Set Messaging from Calculation Logic

Previous step: Link Pricing Strategy to Business Units or Products

Set Messaging from Calculation Logic

There are two types of results to be returned - you can find out more about that in Custom Engines. What we returned right now is a big decimal with a price.

When we cannot calculate and return a price, we can report the problem from the logic:

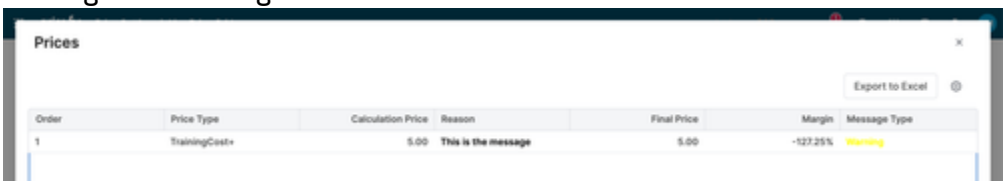
- Throw an exception with an error message.
  - The reason message will be displayed as white text on a red background.
- Return a more detailed result with a message and type.
  - The reason message will show the provided message, and you will see the message type with an appropriate color on the side.
  - This could be used, for example, when calculating a competition based price. We can return the name of the competitor or any other information the pricing manager may need to understand the calculated price.

Add the messaging to your implementation:

1. Change the implementation of your engine to:

```
Map calculatePrice() {                  //
    return [
            price       : 5.0,
            message     : "This is the message",
            messageType : "Warning"     //
    ]
}
```

   Watch out - the returned type is *Map*.
   For possible message types, see Custom Engines.
2. Remember to deploy the modified logic.
3. Recalculate the product in the Live Price Grid.
4. Review the results in the column Prices, and make sure that you see not only the price but also the message with the right color.



Previous step: Test Calculation using Custom Pricing Strategy

Next step: Add Built-in Parameter to Strategy Call

Add Built-in Parameter to Strategy Call

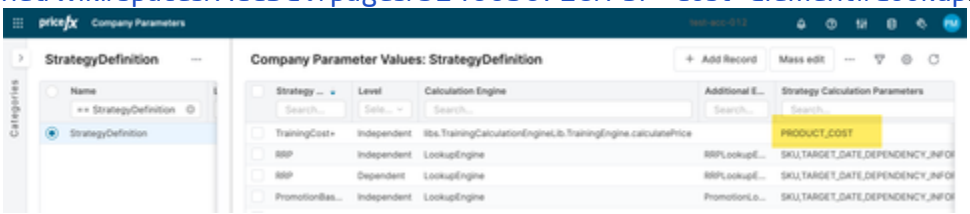
Now let's investigate how to pass parameters to the strategy calculation. We will begin with a pretty simple CostPlus calculation.

Let's assume that we want to have the cost of the product and we want to calculate a price, which should be the cost plus €5 in this case.

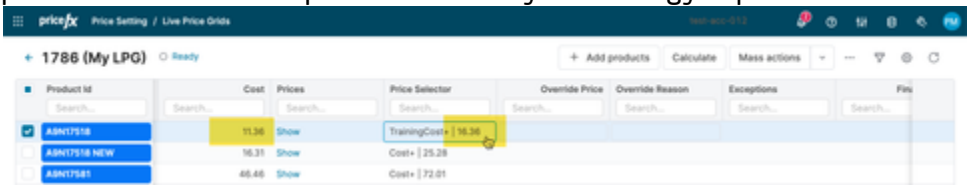
1. Change the implementation of your strategy to accept a parameter:

```
Map calculatePrice( BigDecimal cost ) {

    BigDecimal price = cost + 5.0

    return [
        price       : price,
        message     : "OK",
        messageType : "Info"
    ]
}
```

   - **Note**: This example has no validation, so it will not check if we have a cost or not. In more complex examples you can build a logic to check this. If no cost is found, you can return a warning message for the pricing manager to investigate.
2. Set the Strategy Calculation Parameters:
   a. Navigate to the Company Parameter *StrategyDefinition.*
   b. Review the content of the *Strategy Calculation Parameters* column for several strategies.
      - Those parameters represent the values which are passed to the strategy `calculatePrice` method, in the specified order.
      - You can see there for example SKU, FINAL_LIST_PRICE_ELEMENT_NAME, FINAL_PRICE_ELEMENT_NAME, DEPENDENCY_PROPERTIES, STRATEGY_NAME. For a full list, you can review the code of the element *PriceManagerUtils* of the calculation logic *PriceBuild erCommonElementUtils*, which is part of the installed Price Setting Accelerator. You can find them also in documentation: Calculation Engines.
   c. For your strategy *TrainingCost+*, set the value for the *Strategy Calculation Parameters* field to "PRODUCT_COST" - this is one of the out-of-the-box parameters, so it will work only if you have a basic cost lookup configured and data is available for tested product: https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/3246096728/PSP+Cost+Element#Lookup.



3. Recalculate prices of your product in the Live Price Grid.
   a. On the recalculated item, review the value of fields *Cost* and *Final Price* to make sure that the price is calculated as expected based on your strategy implementation.



Previous step: Set Messaging from Calculation Logic

Next step: Add Custom Parameter to Strategy Call Logic

Add Custom Parameter to Strategy Call

You have learned how to pass existing parameters to your logic, but you may want to add some other factors/parameters which are not available out-of-the-box.

Let's add our own custom simple parameter:

1. Modify *Strategy Calculation Parameters* to pass one more parameter to our calculation method call:
   a. Navigate to the Company Parameter *StrategyDefinition.*
   b. In definition of your strategy *TrainingCost+*, set the value for the *Strategy Calculation Parameters* field to "PRODUCT_COST,OWN_PARAMETER".
2. Modify the implementation of your strategy to accept one more parameter – we call it *ownParamete r.*

```
Map calculatePrice ( BigDecimal cost, BigDecimal ownParameter ) {

    BigDecimal price = cost + ownParameter

    return [
        price       : price,
        message     : "OK",
        messageType : "Info"
    ]
}
```

3. Add implementation of the custom parameter:
   a. Find logic *ParentPriceListLogic* which is part of Price Setting Accelerator installation which you have on your partition.
      i. The same element exists in *DependentPriceListLogic* and it works exactly the same, but in this article we are focusing only on the parent one.
   b. Review the code of the element *AdditionalCalculationParameters*. It defines a map, used for additional custom parameters. It is explained in detail inside the element. The key should be used for static values like constants that are already available in the calculation logic and it shou ld be used also for any dynamic parameters because they are lazy loaded only when an engine using them is being used. The value must be a Closure.
   c. Modify the definition of *AdditionalCalculationParameters* to provide a Closure value for *OWN_PA RAMETER.*

```
return out.WarningManager.tryToExecuteElement
("AdditionalCalculatorParameters") {
    return [OWN_PARAMETER: {return 10}]
}
```

   d. Deploy the change.
4. Recalculate the product item in your Live Price Grid, and:
   a. Show the *Prices* field to see if the result of your strategy calculation is displayed correctly.
   b. Review the fields *Cost* and *Final Price* to see if the calculation is done as expected.

Previous step: Add Built-in Parameter to Strategy Call

Conclusion

Now you know how to create your own custom strategy, how to pass parameters between Pricefx Studio and your Pricefx partition and how to create additional parameters that you can define through StrategyDefinition.

Additionally, you have learnt that if you need more control over messages (both info and error messages), you can return a more complex map structure with info, warning and critical types of messages.

You have also learnt where to find the libraries that drive the calculation engines and can select from out-of-the-box libraries within Pricefx Studio CalculationEnginesLib or create your own.

How to Modify Existing Price Calculation Strategy

Out-of-the-box strategies use the same mechanism as custom price calculation strategies. The only difference is where they are stored. They tend to be more generic and configurable, but sometimes they may not be configurable enough. That is why you may decide to extend them on your own. This article explains how to do it.

> Strategy Designer gives you ability to use "Strategy Templates" which are based on out-of-the-box Price Setting Accelerator strategies. Please check it out before proceeding, because it may solve your use case in a more user friendly way.

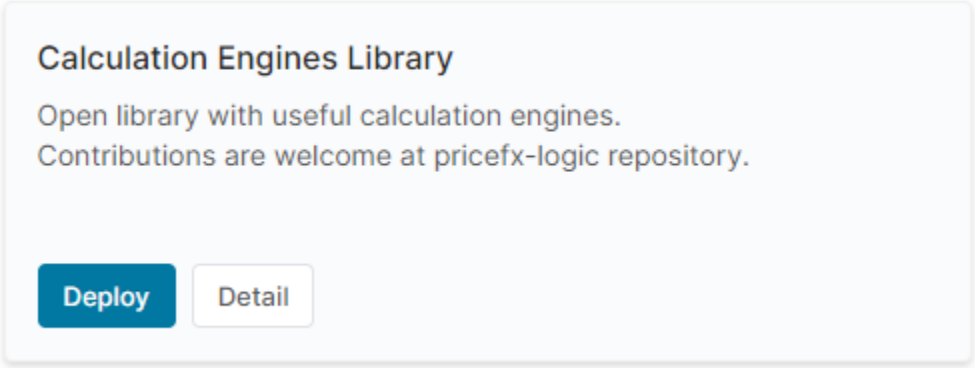
Prerequisites

This article assumes that you know How to Create and Set up New Price Calculation Strategy.

Where to Find Out-of-the-Box Strategies

All out-of-the-box strategies are stored in the pricefx-logic Git repository in `CalculationEnginesLib`. This repository stores our global utilities and we keep them backwards compatible.

`CalculationEnginesLib` is deployed automatically when deploying Price Setting Accelerator. If you want to use it on a project without Price Setting Accelerator, it is also possible. You can deploy it independently from Platform Manager Marketplace:



Using Out-of-the-Box Strategy in Custom Strategy

Out-of-the-box strategies follow the same requirements as custom strategies, so each of them will have a `calculatePrice(...)` method that returns either `BigDecimal` or `Map`. All of them also have a detailed explanation about what input parameters are expected and what errors can be thrown. For example, this comes from `AdjustmentEngine`:

```
/**
 * Returns the price adjusted by the given value based on the selected calculation mode.
 * @param productCost BigDecimal with base product cost
 * @param productPercentageAdjustment BigDecimal with percentage price adjustment. The expected range is 0.0 - 1.0
 * @param productAbsoluteAdjustment BigDecimal with absolute price adjustment
 * @param strategyConfigFromPP Map that defines additional parameters. Expects a structure [CALCULATION_MODE : "Calculation Mode"].
 * @return BigDecimal with adjusted price
 *
 * @throws XExpression The custom calculation exception in "${ERROR_CODE}||${TECHNICAL_MESSAGE}" format.
 * Available error codes for this engine are:
 * - UNSUPPORTED_CALCULATION_TYPE
 * - CALCULATION_ERROR
 * - WRONG_INPUT_DATA
 */
BigDecimal calculatePrice(BigDecimal productCost, BigDecimal productPercentageAdjustment, BigDecimal productAbsoluteAdjustment, Map strategyConfigFromPP) {
    return adjustmentEngine(productCost, productPercentageAdjustment, productAbsoluteAdjustment, strategyConfigFromPP.getAt(CONFIG_FIELDS.CALCULATION_MODE)).calculate()
}
```

Usually the most complicated input parameter is going to be `strategyConfigFromPP`, but you can check details of what it expects by reading documentation on Calculation Engines. All supported values and expected columns are also defined at the top of the element with the engine's implementation. For example, `AdjustmentEngine`:

```
/**
 * Currently defined calculation modes for the Adjustment Engine
 */
@Field CALCULATION_MODES : LinkedHashMap<String, String> = [absolute    : "Absolute",
                          percentage  : "Percentage",
                          sellingPrice: "SellingPrice"]


/**
 * Structure of the PP Additional configuration.
 */
@Field CONFIG_FIELDS : LinkedHashMap<String, String> = [CALCULATION_MODE: "Calculation Mode"]

@Field Map ERROR_TYPES = [UNSUPPORTED_CALCULATION_TYPE: "UNSUPPORTED_CALCULATION_TYPE",
                          CALCULATION_ERROR           : "CALCULATION_ERROR",
                          WRONG_INPUT_DATA            : "WRONG_INPUT_DATA"]
```

This design makes it easy to treat out-of-the-box engines as black boxes. So if you want to add a step before or after, you can manually call the `calculatePrice(...)` method from your own custom engine. For example, to do a simple multiplication of a result you could create a custom engine like this:

```
BigDecimal calculatePrice() {
    BigDecimal productCost = 10G
    BigDecimal productPercentageAdjustment = 1.05
    BigDecimal productAbsoluteAdjustment = null
    Map strategyConfigFromPP = ["Calculation Mode": "Percentage"]
```

```
    BigDecimal ootbEngineResult = libs.CalculationEnginesLib.
AdjustmentEngine.calculatePrice(productCost,
productPercentageAdjustment, productAbsoluteAdjustment,
strategyConfigFromPP)

    return ootbEngineResult * 1.05G
}
```

Of course, it can be as simple or as complex as you need, as described in How to Create and Set up New Price Calculation Strategy, so you can add your own inputs etc.

Using this approach means that all fixes and changes applied to the out-of-the-box engine will be automatically applied to your custom engine when you deploy a newer version of `CalculationEnginesLib`.

Modifying Out-of-the-Box Strategy

If you want to reuse the engine and you need to do more "breaking" changes, the recommended way is to create a new custom engine in your own library type logic as described in How to Create and Set up New Price Calculation Strategy and copy the whole implementation from an existing engine.

> ⚠ As of now, if the engine you are copying has the `strategyConfigFromPP` parameter which you need to remove from the parameters list and replace it with a hard-coded version. All the other parameters can stay untouched. Example of this based on `AdjustmentEngine` can be seen in the previous paragraph.
>
> In Price Setting Accelerator version 2.2.0 and higher, this step will not be necessary and it will be as simple as doing a copy-paste of the whole implementation and configuration in StrategyDefinitionPP.

Copying the engine is the recommended approach because it assures that your changes will not be overwritten when you decide to upgrade the accelerator or `CalculationEnginesLib`.

How to Create Publishing Template

Price Setting Accelerator does not support any out-of-the-box publishing templates and preprocessing logics.

If you need to add one, you can do it the same way you would do for any other code. A list of element names available through `api.currentItem()` in the processing logic can be found in `ParentPriceListLogic` and `DependentPriceListLogic` logics.

Reference documentation worth checking out:

- Publishing Templates - Handbook
- Publishing Templates
- Word Publishing Template Preprocessing Logic

How To Set up Custom Dimension Lookup

Prerequisites

- Price Setting Accelerator is deployed.
- New custom dimension is **not** the 7th dimension.
- You are a Configuration Engineer or have sufficient knowledge of Groovy coding, Pricefx Studio and Pricefx Sandbox.

Assumptions

This example assumes:

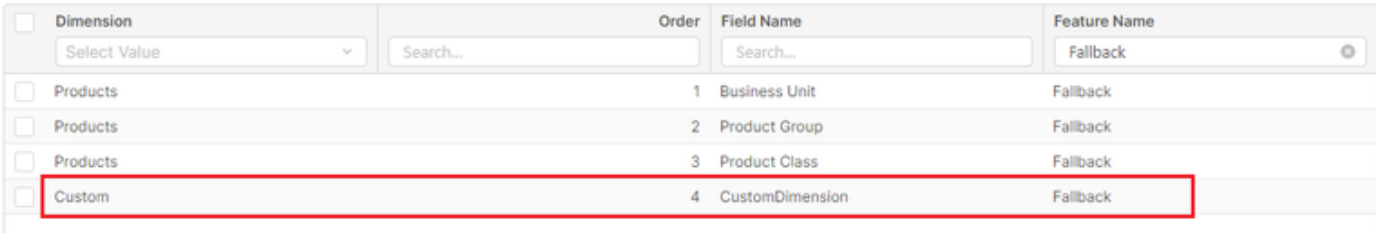
- You already have 3 product dimensions under "Fallback" Feature Name.
- You want to add a custom dimension as the 4th dimension.
- If you have "per-feature" hierarchical lookups defined, these will be unaffected.

Configuration Steps

Add Custom Dimension to PriceSettingDimensions Price Parameter

Go to the Price Parameter **PriceSettingDimensions** and add a new row:

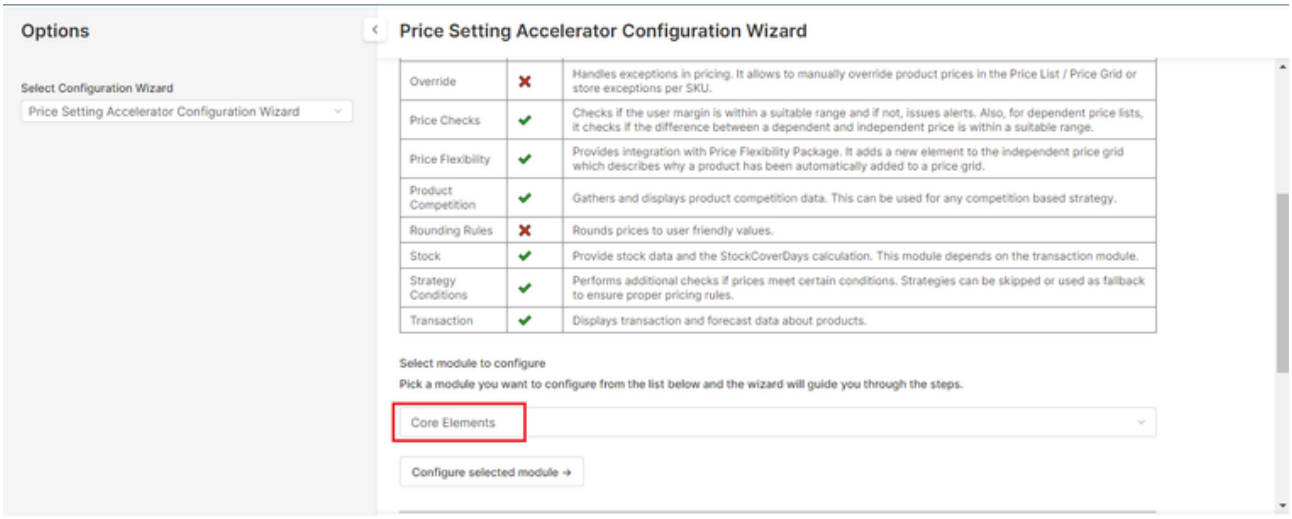
- Dimension: "Custom"
- Order: "4"
- Feature Name: "Fallback"
- Field Name: According to your needs. This variable will be the name of the column and will be referenced in the code.



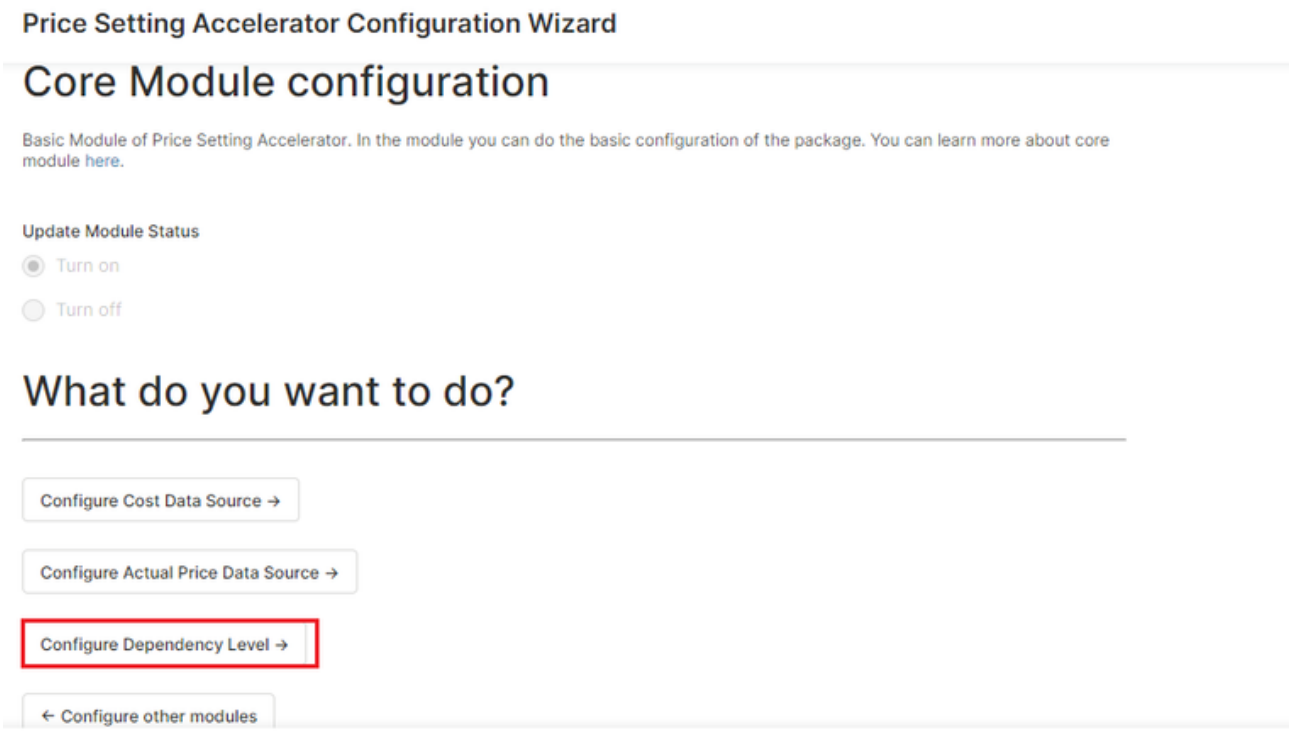
Run Bootstrapping Again

To make this new custom dimension available in <dependency> Price Parameters, use **Price Setting Accelerator Configuration Wizard** to add a new dependency level.

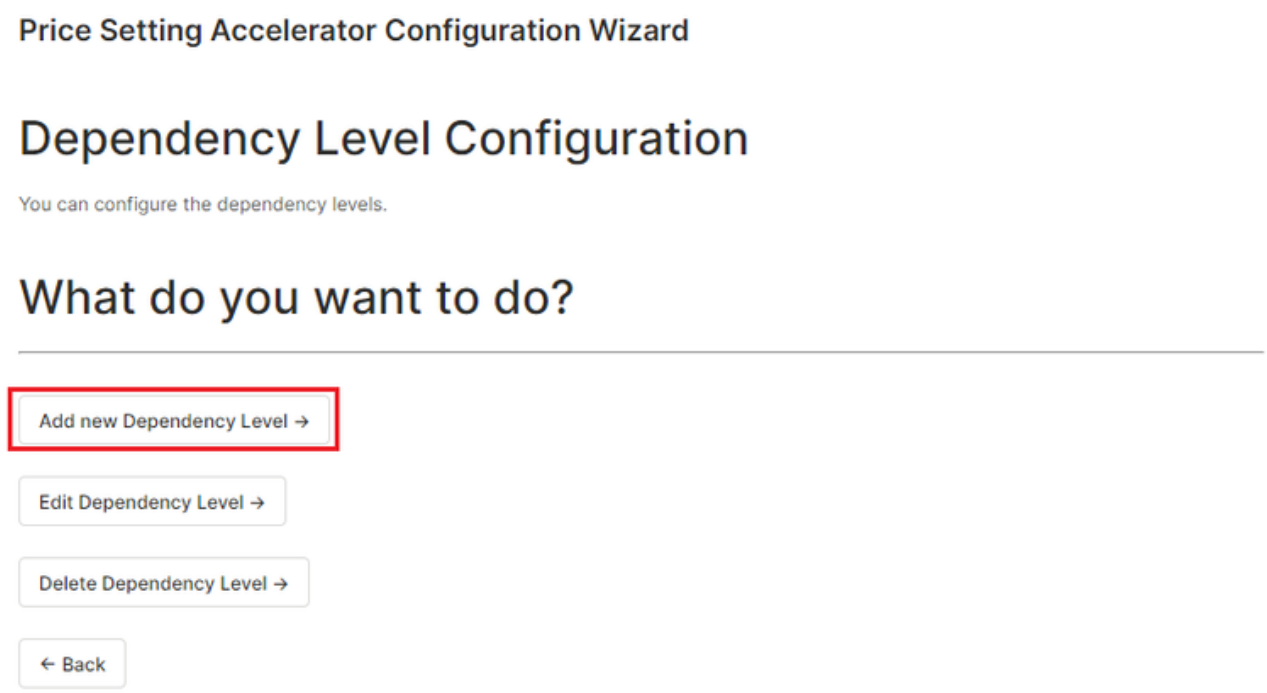
1. Select Core Elements in the list and click the **Configure selected module** button.



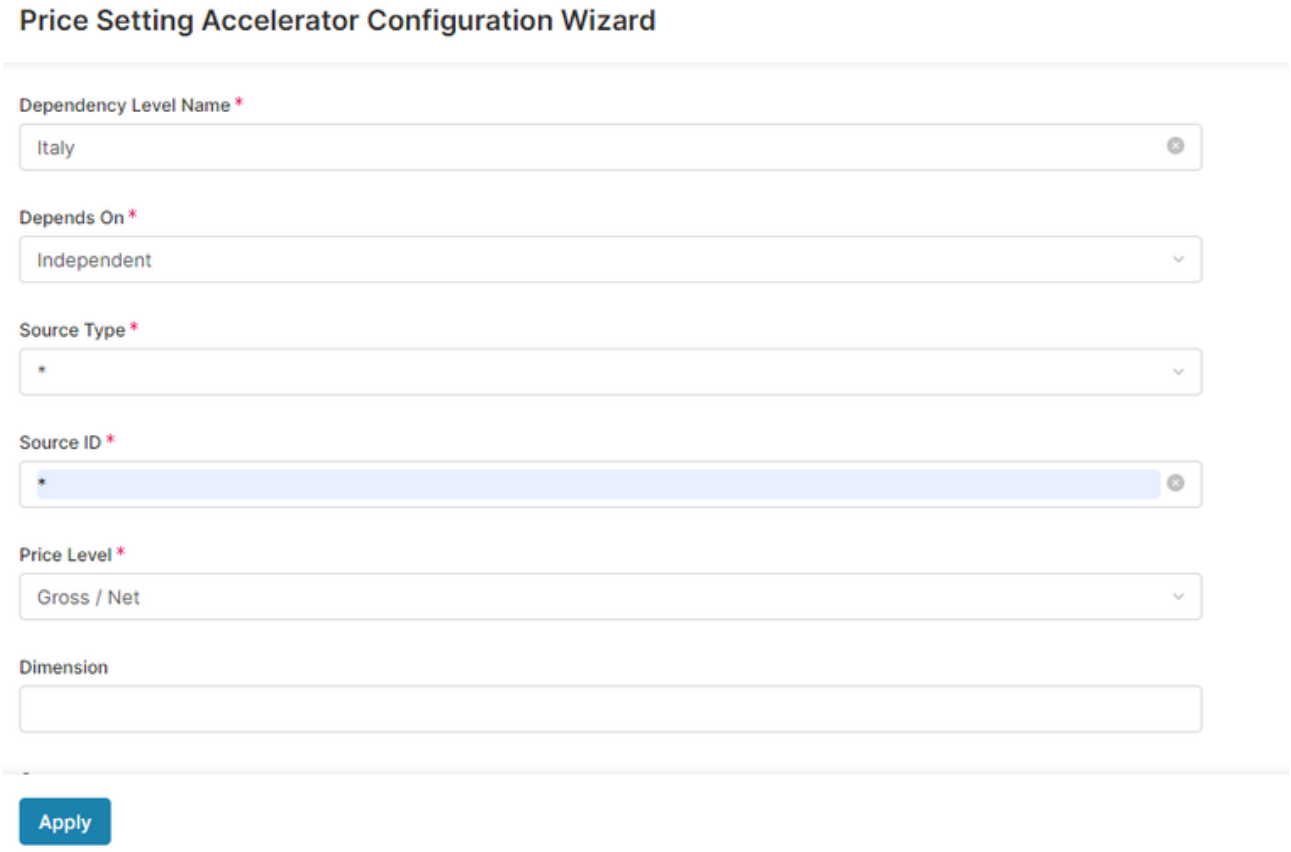
2. Select **Configure Dependency Level** button.



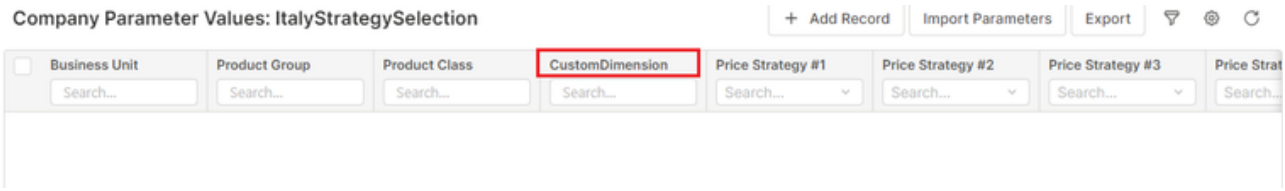
3. Select **Add new Dependency Level** button.



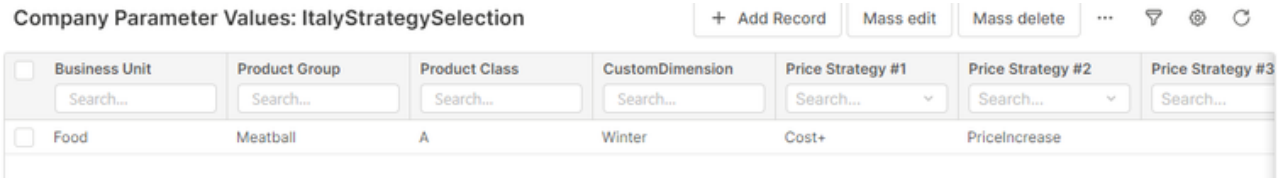
4. Fill in necessary information and click the **Apply** button.



5. Now, you will have the *CustomDimension* column in <Italy> Price Parameters:



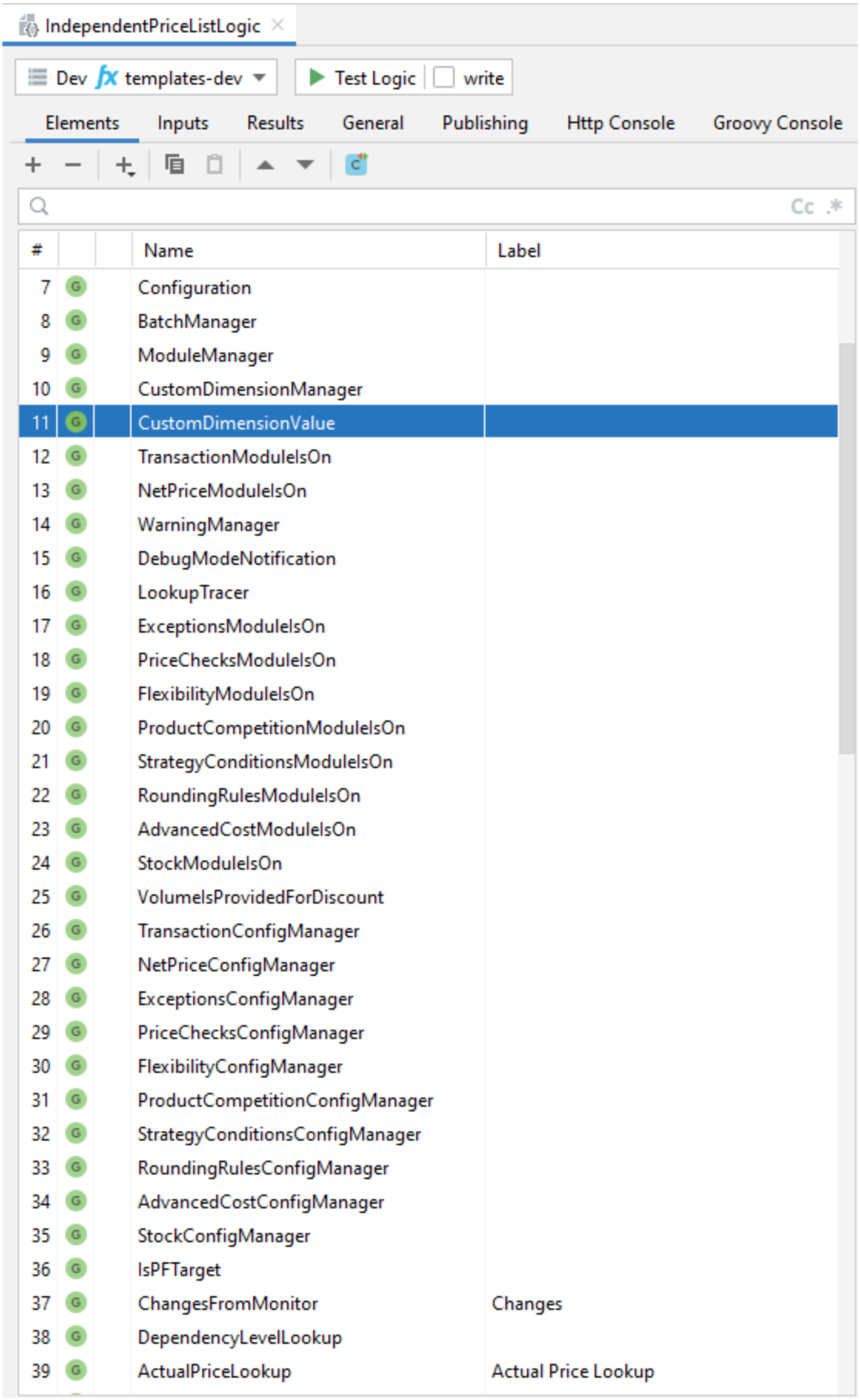
6. You need to add pricing data into the Price Parameters to start using the custom dimension, as shown below.



Edit Price List Logic

The previous steps prepared data and configuration and now you need to adjust lookups by updating the Price List logic. In the (In)DependentPriceListLogic, you need to get a custom dimension value for each product and pass it into CustomDimensionManager.

1. Create a new element called “CustomDimensionValue”. It needs to be placed below the out-of-the-box “CustomDimensionManager“. Your new element does not need to be visible, you do not need to occupy any of 100 allowed attributes.


IndependentPriceListLogic
Dev templates-dev
Test Logic
write
Elements
Inputs
Results
General
Publishing
Http Console
Groovy Console
Cc .*
Name Label
7 Configuration
8 BatchManager
9 ModuleManager
10 CustomDimensionManager
11 CustomDimensionValue
12 TransactionModuleIsOn
13 NetPriceModuleIsOn
14 WarningManager
15 DebugModeNotification
16 LookupTracer
17 ExceptionsModuleIsOn
18 PriceChecksModuleIsOn
19 FlexibilityModuleIsOn
20 ProductCompetitionModuleIsOn
21 StrategyConditionsModuleIsOn
22 RoundingRulesModuleIsOn
23 AdvancedCostModuleIsOn
24 StockModuleIsOn
25 VolumeIsProvidedForDiscount
26 TransactionConfigManager
27 NetPriceConfigManager
28 ExceptionsConfigManager
29 PriceChecksConfigManager
30 FlexibilityConfigManager
31 ProductCompetitionConfigManager
32 StrategyConditionsConfigManager
33 RoundingRulesConfigManager
34 AdvancedCostConfigManager
35 StockConfigManager
36 IsPFTarget
37 ChangesFromMonitor Changes
38 DependencyLevelLookup
39 ActualPriceLookup Actual Price Lookup

2. In the new element:
    a. Add a logic to get the custom dimension value.
    b. Fetch manager by `out.CustomDimensionManager`.
    c. Pass PLI/PGI's value of your custom dimension into CustomDimensionManager by using the method `addCustomDimensionValue`.

CustomDimensionValue.groovy

```groovy
Map customDimensionManager = out.CustomDimensionManager

String customDimensionFieldName = "CustomDimension" //the Field Name value in PP PriceSettingDimensions
String customDimensionValue = "Winter" //or get somewhere

customDimensionManager.addCustomDimensionValue(customDimensionFieldName, customDimensionValue)

```

Configurable Lookups Guide

The PSP lookup framework allows you to define PX/PP/PCOMP lookups in a Price Parameter table, enabling you to modify existing lookups without any Groovy coding and add new ones with minimal Groovy required.

Benefits of Configurable Lookups

- **Unification** – Every out-of-the-box PSP Data Lookup uses the same features, allowing you to add your own lookups.
- **Performance** – These lookups are cached (if called without context and without PLI/PGI key) or cached and batched (if called with context).
- **Ease of use** – Typically complicated support for Dependency Levels is embedded into Configurable Lookups with a single flag.

Note on Caching and Batching

Caching and batching are two techniques used to enhance data lookup performance in databases.

- **Caching** looks up data only once and saves a single lookup result for reuse across multiple line items – because this value will be the same.
- **Batching** looks up data for 200 line items at once. This improves the performance significantly, as there are fewer connections to the database.

Both of these approaches are supported.

If the lookup does not rely on the product context, there is no point in batching it. In such a case, the lookup is performed only once and its value is cached. This occurs when the lookup fields 'key1Field' and 'key2Fields' are empty, meaning there is no column in the data structure that refers to a specific product.

Examples of such cached lookups are out-of-the-box PSP lookups: RoundingRules and StrategyConditions.

Further Reading

- Configurable Lookups and PSP Lookups
- Define Your Own Lookup - Cookbook
- Configurable Lookup Columns Description

Configurable Lookups and PSP Lookups

With the introduction of configurable lookups, users now have the freedom to customize how they conduct lookups, but the data must still be provided in a specific format.

This requirement is fulfilled by the approach outlined.

- It is up to the user to choose:
  - Source table type
  - Source table name
  - Source table columns
  - Secondary keys
  - Additional custom filters
  - Data sorting
  - ValidAfter or ValidFrom-ValidTo support
  - Warnings/errors if there is no data
  - Usage of dependency levels fallbacks
- PSP requires:
  - Lookup of specific feature to be defined at all.
  - In “selectedFields”, data should be mapped to specific Groovy names.

Example for basic cost:

Edit Value

Name *

Cost

User Source Type

PX

User Source Name

ProductCosts

Selected Fields

{"cost":"Cost","currency":"Currency"}

Checks Config

{"MORE_THAN_1":"PSP_WARNING", "NO_ENTRY":"PSP_THROW", "NO_

Custom Filter

Sort By Field

Key 1 Field

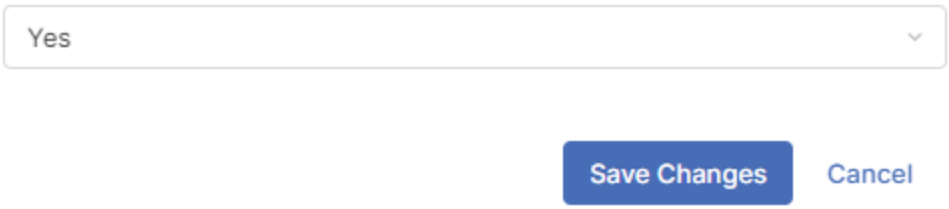
sku

Key 2 Field

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Description per feature

- Base Cost:
  - LookupName: Cost
  - Mandatory fields:
    - cost
  - Optional fields:
    - currency
- Advanced Cost:
  - Advanced Cost table should point at a specific lookup name
  - Mandatory fields of specific lookup name:
    - cost
  - Optional fields of specific lookup name:
    - currency
- Actual Price (only if looked up from the data, not PG/PL)
  - Mandatory fields:
    - price
  - Optional fields:
    - currency
- Stock
  - Mandatory fields:
    - stock
- RoundingRules
  - Mandatory fields:
    - lowerBound
    - upperBound
    - roundingRule
    - roundingMode
- priceException
  - Mandatory fields:
    - value
  - Optional fields:
    - currency
- strategyException
  - Mandatory fields:
    - value

As of current version, the below lookups are not configurable through a Price Parameter. The same mechanism is used, but the lookup configuration is part of the Groovy code:

- Product Competition

    - LookupName: ProductCompetition
    - Mandatory fields:
        - competitor
        - price
    - Optional fields:
        - infoDate
        - priceType
        - sku
        - attributeId of column with dependency level info (if DependencyLevelHierarchy is used)

- StrategyConditions
    - Mandatory fields:
        - order
        - condition
        - rule
        - checkException
        - ruleExplanation
        - dependencyLevel

Define Your Own Lookup - Cookbook

1. Most Basic Scenario

Step 0: Task

I want to create the most simple cost lookup which will be batched. On algorithmical level, we need to know the following:

- Identifying table:
    - What is the type of table?
    - What is the name of table?
- What are columns we are looking for:
    - What is the name of a column with SKU?
    - What is the name of a column with Cost?

Step 1: Prepare Lookup Configuration

Let's put the information from Step 0 to the LookupConfiguration table:

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn"}

Checks Config

Custom Filter

Sort By Field

Key 1 Field

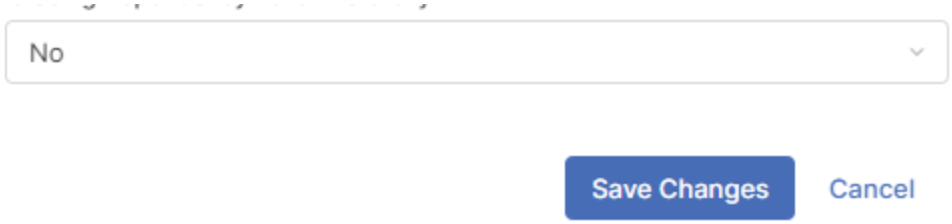
sku

Key 2 Field

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Fields:

- **Name** - Identifier of the Lookup operation. Here we called it `CustomCost`.
- **userSourceType** - Type of the table. LookupConfiguration has been tested with PX/PP/PCOMP tables. Here, we are using PX.
- **key1Field** - Column name or attributeId. Value of that column needs to match key1 of the current context. In PLI and PGI, key1 is always SKU. In PX, this column name is not editable, so we need to put there `sku` (lowercase - that is the name of the PX column).
- **selectedFields** - Map in JSON format. Keys of this map will be referenced in the code. After the lookup is performed, there will be record's data in these keys in Groovy Map. Values match either ColumnNames or atributeIds. In our case we will refer to our looking value as "cost" and it should be looked up from a column called "CostColumn" - this the input `{"cost":"CostColumn"}`.
- **IsUsingDependencyLevelHierarchy** - Allows you to turn on and off (`Yes/No`) fallback on parent Dependency Levels. This is used for integration of complex PSP mechanism. In this simple scenario, we are turning the fallback off.

Step 2: Call Lookup Library

Put this anywhere in your code before you need to use your cost value:

```
libs.ConfigurableLookupsLib.LookupInstanceFromPP.prepareConfiguredLookup
("CustomCost")
```

The above code handles all database operations for the whole batch. In a more advanced scenario, you will also pass to this function elementName (for debugging purposes) and dependency config (if using DependencyLevelHierarchy).

Step 3: Read Results

Reading entry does not invoke database calls:

```
Map lookupResult = libs.ConfigurableLookupsLib.LookupInstanceFromPP.
getFirstEntry("CustomCost").
```

You will find your value from "CostColumn" under the key you defined.

```
BigDecimal customCost = lookupResult?.cost //lookupResult might be null
if there is no entry at all
```

2. Add PSP Dependency Level Fallbacks

Prerequisites

- You understand how the three values in PSP DependencyMapping work.
- You already went through the previous cookbook https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/edit-v2/5049352277.

Step 0: Task

I want to look up Cost from a table using PSP Dependency Mapping. On algorithmical level, I need to know the following:

- How to match data to proper dependency level.

Step 1: Prepare Lookup Configuration

This information is configured outside of Configurable Lookups, we will only specify that we are using this feature:

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn"}

Checks Config

Custom Filter

Sort By Field

Key 1 Field

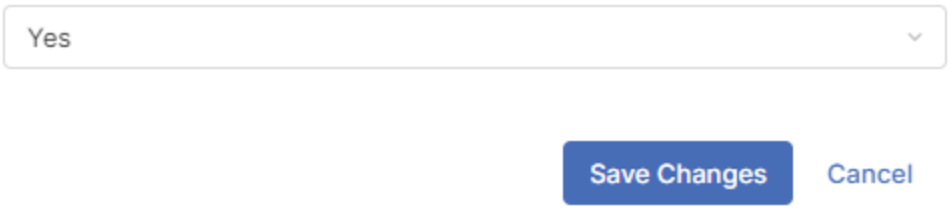
sku

Key 2 Field

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Fields:

- **IsUsingDependencyLevelHierarchy** - Dependency level hierarchy will be passed through the Groovy code to Configurable Lookup Library.

Step 2: Call Lookup Library

You need to prepare two more values before calling the lookup library.

1. Entry from DependencyConfiguration PP. In PSP you can get it by calling:

```
Map dependencyLevelLookup = out.DependencyLevelLookup
```

2. Map with the same format as stored in DependencyMappingConfig (Dependency Mapping). We will refer to this map as "customCostDependencyMapping". Sample:

```
Map customCostDependencyMapping = [mappingType: "Lookup",
    dependencyLevelFilterField: "DependencyLevelName",
    sourceFilterField: "DependencyLevelNameColumn"]
```

   a. mappingType - `Lookup/Table/None`
   b. sourceFilterField - Name of the column in dependencyConfiguration.
   c. dependencyLevelFilterField - Name of the column in the data.

Look up the whole batch at once:

```
libs.ConfigurableLookupsLib.LookupInstanceFromPP.prepareConfiguredLookup
("CustomCost",
    "customCost", // This is element name, used for debugging
    [thisDependencyInfo: dependencyLevelLookup,
dependencyMappingConfig: customCostDependencyMapping]
    )
```

Step 3: Read Results

Reading results has not changed since the previous cookbook.

3. ValidFrom

Prerequisites

Prerequisites

- You already went through the previous cookbook 2. Add PSP Dependency Level Fallbacks.
- Your CustomCostTable already has a column with "Valid From"/"Valid After" date.

Step 0: Task

Cost changes every month. I want to add validity dates to my PX entries.

I want to look up Cost from a table using Dependency Mapping. On algorithmical level, I need to know the following:

- What makes cost entry valid?
- What is the time of calculation?

Step 1: Prepare Lookup Configuration

Using data from Step 0, "Valid From" requirement will be implemented by:

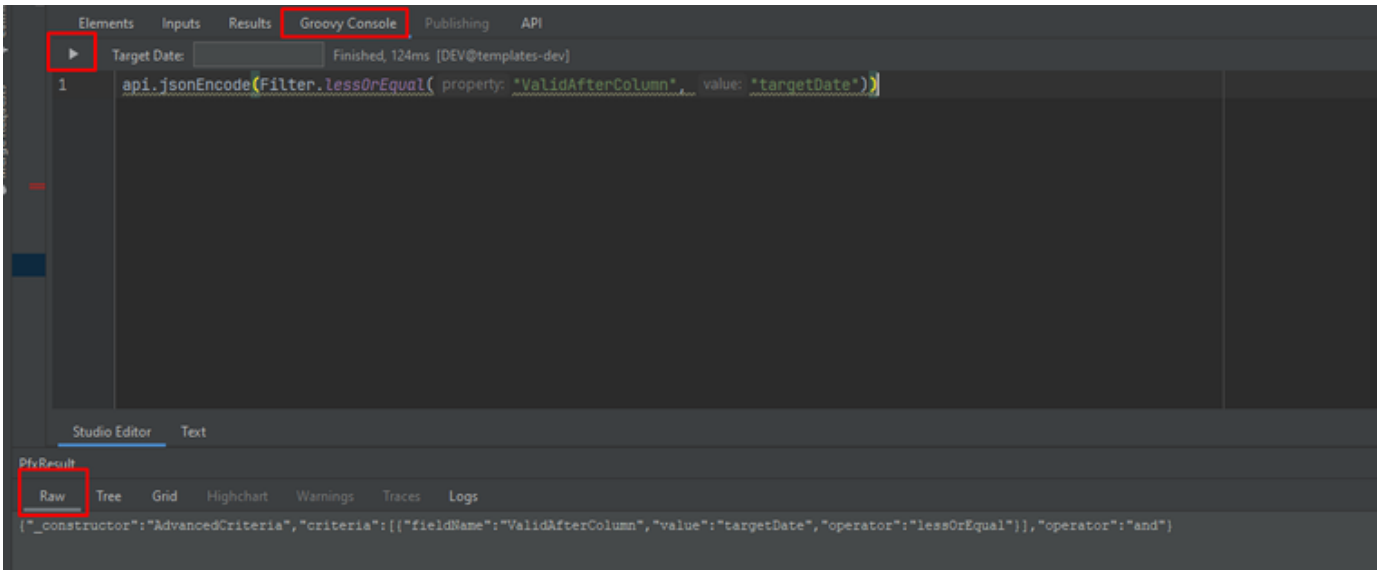
- Setting up "Custom Filter" which filters out all outdated entries.
- Setting up "Sort By Field" to choose the earliest among valid entries.

Configurable Lookups accept filters in the format of JSON encoded Pricefx Filter.

1. Open Pricefx Studio.
2. Open any logic.
3. Click **Groovy Console**.
4. Enter:

```
api.jsonEncode(Filter.lessOrEqual("ValidAfterColumn",
"targetDate"))
```

5. Click **Run**.
6. In the PfxResult window click **Raw**.
7. You will see the encoded filter.



You can put any working filter in the "customFilter" field in LookupConfiguration. However, the above filter is not correct:

- It does not have attributeId but the name of the column.
- "targetDate" would not dynamically resolve into date.

LookupConfiguration supports two tags which will solve this kind of issues.

Let's change:

- `"ValidFromColumn"` to `"<<FIELD_NAME(ValidFromColumn)>>"`
- `"targetDate"` to `"<<TARGET_DATE>>"`

These tags will be dynamically resolved at runtime.
Before:

```
{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"
ValidFromColumn","value":"targetDate","operator":"lessOrEqual"}],"operator":"
and"}
```

After:

```
{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_NAME
(ValidFromColumn)>>","value":"<<TARGET_DATE>>","operator":"lessOrEqual"}],"
operator":"and"}
```

Put that last filter with tags in the "CustomFilter" field.

Then put `-ValidFromColumn` in sortByField as shown below. We want sorting from the earliest date to latest.

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn"}

Checks Config

Custom Filter

{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_N

Sort By Field

-ValidFromColumn

Key 1 Field

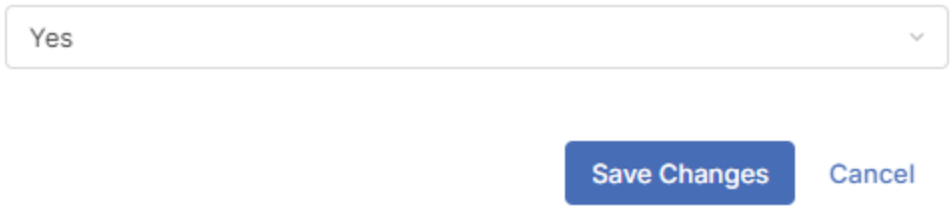
sku

Key 2 Field

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Fields:

- **customFilter** - Any filter which will be applied to data lookup. It supports some tags which are dynamically resolved at runtime. Usually utilized to implement validation of entries due at the pricing date.
- **sortByField** - How data should be sorted on the database level if more than one entry is returned. If you are interested in smallest/biggest/latest, fill in this field.

Step 2: Call Lookup Library

Calling lookup library has not changed from the previous cookbook.

Step 3: Read Results

Reading results has not changed since the previous cookbook.

4. ValidFrom and ValidTo

Prerequisites

- You already went through the previous cookbook 3. ValidFrom.
- Your CustomCost table already has a column with "ValidTo" date.

Step 0: Task

As data sanitation, I want to change my lookups to use lower and upper boundary for time filter. On algorithmical level, I need to know the following:

- What is the upper boundary for targetDate for an entry to be valid?

Step 1: Prepare Lookup Configuration

Using data from Step 0, do the following to fulfill the "ValidFrom - ValidTo" requirement:

- Set up "Custom Filter" which filters out all outdated entries.
- Warn users if multiple entries are found for a given timeframe.
- Theoretically, no sorting for this type of lookup is needed - only one entry is expected. However, for consistency between runs in Price Setting Package, we add "-validFrom" sorting anyway. It is up to users if they want this fallback mechanism or not.

We will go through these steps to produce a new custom filter. In Pricefx Studio Sandbox, put the following code:

```
api.jsonEncode(
    Filter.and(
        Filter.lessOrEqual("<<FIELD_NAME(ValidFromColumn)>>",
"<<TARGET_DATE>>"),
        Filter.greaterThan("<<FIELD_NAME(ValidToColumn)>>",
"<<TARGET_DATE>>")
    )
)
```

This will produce the below filter. Put that value as **customFilter**:

```
{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_NAME
(ValidFromColumn)>>","value":"<<TARGET_DATE>>","operator":"lessOrEqual"},
{"fieldName":"<<FIELD_NAME(ValidToColumn)>>","value":"<<TARGET_DATE>>","
operator":"greaterThan"}],"operator":"and"}
```

Then, we need to set up a warning in case more than one entry is in a specific time frame. LookupConfiguration supports multiple types of warnings:

- Unexpected errors
- Checking entries
- Checking values

Let's ignore the last one and configure the first two. Inside ChecksConfig, let's put this data:

```
{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING"}
```

This is a JSON Map. Keys of this map are error codes, values are ENUMs. We defined that in case of:

- Unexpected lookup error, throw an exception in a way that PSP does it.
- More than 1 entry found, add a warning to PSP warning manager.

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn"}

Checks Config

{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING"}

Custom Filter

{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_N

Sort By Field

-ValidFromColumn

Key 1 Field

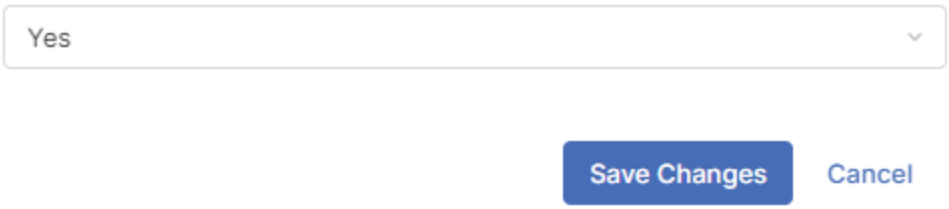
sku

Key 2 Field

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Fields:

- **checksConfig** - Defines behavior in case of corrupted data or incorrect lookup configuration.

Step 2: Call Lookup Library

Calling lookup library has not changed from the previous cookbook.

Step 3: Read Results

Reading results has not changed since the previous cookbook.

5. Data Validation and Currency

Prerequisites

- You already went through the previous cookbook 4. ValidFrom and ValidTo.
- Your CustomCost table already has a column with currency.

Step 0: Task

Currency is strictly connected to money value. I want to look it up together with any cost entry. I want specific behavior when the entry is missing data. On algorithmical level, I need to know the following:

- Name of the column with currency.
- What should happen if data is missing.

Step 1: Prepare Lookup Configuration

To implement the above information, we will modify two fields of our LookupConfiguration entry:

- We will change the **selectedFields** entry to also look up the currency:
  `{"cost":"CostColumn"}` -> `{"cost":"CostColumn", "currency":"CurrencyColumn"}`
- We will add 2 more entries to our **checksConfig** Map.
  - Possible keys for the checksConfig Map are dynamically generated for each looked up column. In this case, we will configure automatically performed checks:
    - `NO_FIRST_VALUE-cost`
    - `NO_FIRST_VALUE-currency`
  - We will set up our lookup in a way that an exception is thrown when the cost is missing. However, our flow will assume that the currency might be missing. While it will produce a warning, it will not interrupt the flow.
  - Our checks config will change:

- Before: `{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING"}`
- After: `{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING","NO_FIRST_VALUE-cost":"PSP_THROW","NO_FIRST_VALUE-currency":"PSP_WARNING"}`

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn", "currency":"CurrencyColumn"}

Checks Config

{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING","NO_FIR

Custom Filter

{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_N

Sort By Field

-ValidFromColumn

Key 1 Field

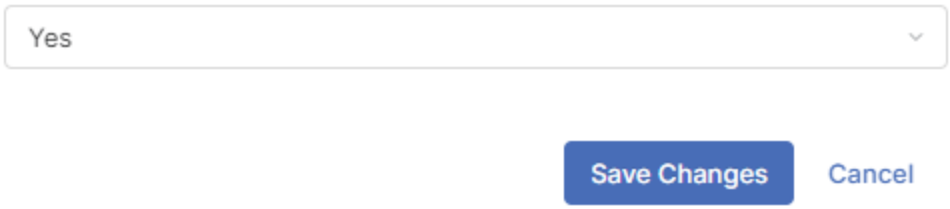
sku

Key 2 Field

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Fields:

- **checksConfig** - Below checks are available in Configurable Lookups:
  - `LOOKUP` Advisable to at least throw in that case
  - `MORE_THAN_1`
  - `NO_ENTRY`
  - `NO_FIRST_VALUE-${selectedField}`
  - `NO_ANY_VALUE-${selectedField}`

This type of behavior is available for each of these checks:

- `IGNORE`
- `LOG`
- `PSP_WARNING`
- `WARNING`
- `PSP_THROW`
- `THROW`
- `PSP_ABORT`
- `ABORT`

Step 2: Call Lookup Library

Calling lookup library has not changed from the previous cookbook.

Step 3: Read Results

Your return map has changed in this cookbook, now it will contain 2 entries:

```
BigDecimal customCost = lookupResult?.cost
def customCurrency = lookupResult?.currency // customCurrency is
nullable
```

6. Secondary Key

Prerequisites

- You already went through the previous cookbook 5. Data Validation and Currency.
- You already modified PSP in a way that utilizes secondaryKey as unique key. In our case it will be CustomerName.

Step 0: Task

I calculate cost differently for every customer. I want to look up the proper cost, based on secondaryKey of a line item. On algorithmical level, I need to know the following:

- Name of the column with secondaryKey in the data.

Step 1: Prepare Lookup Configuration

To implement the above information, we will fill **key2Field** with the name of the PX column where matching value is stored. In our example, it is `SecondaryKeyColumn`:

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn", "currency":"CurrencyColumn"}

Checks Config

{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING","NO_FIRS

Custom Filter

{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_N

Sort By Field

-ValidFromColumn

Key 1 Field

sku

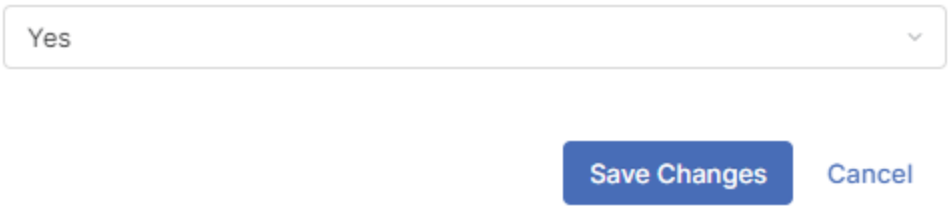
Key 2 Field

SecondaryKeyColumn

Key 1 Transformation

Key 2 Transformation

Is Using Dependency Level Hierarchy *



Fields:

- **key2Field** - Column name or attributeId. Value of that column needs to match key2 of the current context. In PLI and PGI, key2 is always secondaryKey. Take into consideration that while still batched, the constructed filter needs to contain `Filter.and` of 200 filters instead of one `Filter.in` with 200 elements. This is an expected trade-off when using secondaryKeys.

Step 2: Call Lookup Library

Calling lookup library has not changed from the previous cookbook.

Step 3: Read Results

Reading results has not changed since the previous cookbook.

7. Modify Line Item Key on the Fly

Prerequisites

- You already went through the previous cookbook 6. Secondary Key.

Step 0: Task

I calculate cost differently for every customer group. I have only a few customer groups and tens of customers. I do not want to create tens of entries for every product. On algorithmical level, I need to know the following:

- What is the PX column name where Customer Group is stored?
- How to match a specific Customer Name to a proper Customer Group?

Step 1: Prepare Lookup Configuration

To implement the above information, we will:

- Change **key2Field** with the name of the PX column where Customer Group is stored. We will be using Customer Group. In our case, the new column is called `CustomerGroupColumn`.
- Create a logic which handles transformation from Customer Name to Customer Group.
  - Create a Groovy library logic. Create a function within that logic which:
    - Accepts one parameter which will be secondaryKey of a line item. In our case it will be Customer Name.
    - Returns the transformed value - in our case Customer Group.
      Example:

```
def getCustomerGroup(String secondaryKey) {
    if (secondaryKey.startsWith("Cust-1")) {
        return "CUSTOMER_GROUP_1"
    }
    return "CUSTOMER_GROUP_GENERAL"
}
```

- Put the path to the above function in the **key2Transformation** field.
  - The format is: `libs.LibraryName.ElementName.FunctionName`
  - In our case it is: `libs.TransformationLibrary.Utils.getCustomerGroup`

Edit Value

Name *

CustomCost

User Source Type

PX

User Source Name

CustomCostTable

Selected Fields

{"cost":"CostColumn", "currency":"CurrencyColumn"}

Checks Config

{"LOOKUP":"PSP_THROW","MORE_THAN_1":"PSP_WARNING","NO_FIR

Custom Filter

{"_constructor":"AdvancedCriteria","criteria":[{"fieldName":"<<FIELD_N

Sort By Field

-ValidFromColumn

Key 1 Field

sku

Key 2 Field

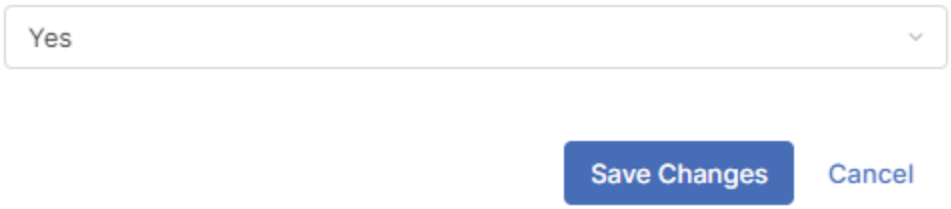
SecondaryKeyColumn

Key 1 Transformation

Key 2 Transformation

libs.TransformationLibrary.Utils.getCustomerGroup

Is Using Dependency Level Hierarchy *



Fields:

- **key1Transformation/key2Transformation** – Path to the function which can transform PGI/PLI key into a different lookup value. It is useful when the data has plain structure (not hierarchical), but it is less granular than line items.

Step 2: Call Lookup Library

Calling lookup library has not changed from the previous cookbook.

Step 3: Read Results

Reading results has not changed since the previous cookbook.

Configurable Lookup Columns Description

Column	Description
Name	Identifier of the Lookup operation. It will need to be referenced in the code.
userSourceType	Type of the table. LookupConfiguration has been tested with PX/PP /PCOMP tables.
selectedFields	Map in the JSON format. Keys of this map will be referenced in the code. After the lookup is performed, there will be record's data under these keys in Groovy Map. Values match either ColumnNames or atributeIds.
key1Field	Column name or attributeId. Value of that column needs to match key1 of the current context. In PLI and PGI, key1 is always SKU. If empty, the lookup will be cached rather than batched – because values will not differ for different products.
key2Field	Column name or attributeId. Value of that column needs to match key2 of the current context. In PLI and PGI, key2 is always secondaryKey. Take into consideration that while still batched, constructed filter needs to contain `Filter.and` of 200 filters instead of one `Filter.in` with 200 elements. This is an expected trade-off when using secondaryKeys.
key1Transformation /key2Transformation	Path to the function which can transform a PGI/PLI key into a different lookup value. Useful when the data has plain structure (not hierarchical), but it is less granular than line items. Example usage: In PG logic, secondaryKey is CustomerName. Product

	costs differ based on CustomerGroup. Here, you will pass the path to the function which translates from CustomerName to CustomerGroup, so you will not need to artificially generate new data entries.
customFilter	Any filter which will be applied to data lookup. It supports tags which are dynamically resolved at runtime: • **<<FIELD_NAME(value)>>** - Value will be expected to be the column's name which will be dynamically translated into attributeId. • **<<TARGET_DATE>>** - Will put translated into target date of the current calculation (in the format suitable for filtering on date fields).
sortByField	How data should be sorted on the database level if more than one entry is returned.
checksConfig	Map defining behavior in case something goes wrong. Keys in the map are types of error, while values define how to handle it. Cases when the check is performed: • No entry found: "NO_ENTRY" • More than 1 entry found: "MORE_THAN_1" • First entry does not have value x. x is a selected field, a key in the "selectedFields" config: "NO_FIRST_VALUE-x" • No entry has value x. x is a selected field, a key in the "selectedFields" config: "NO_ANY_VALUE-x" • Lookup engine returned an error, usually by incorrect configuration: "LOOKUP" • Possible behavior when an issue happens: • IGNORE - Ignores the issue. Default value for all checks. • LOG - Logs the issue to api.logInfo. • PSP_WARNING - Does not interrupt the flow, but registers the error to PSP WarningManager. • WARNING - Does not interrupt the flow, but registers the error by api. addWarning. • PSP_THROW - Throws an exception and registers the error to PSP WarningManager. • THROW - Throws an exception. • PSP_ABORT - Throws an exception and aborts the rest of the current PSP module. • ABORT - Calls api.abourtCalculation and throws an exception.
IsUsingDependencyLevelHierarchy	Allows you to turn on and off (by setting Yes/No) fallback on parent Dependency Levels. Integration to complex PSP mechanism.

Configuration (Price Setting)

A standard configuration in Price Setting includes:

- **PSP Configuration** - Represents the default config which allows for running Price Setting Package.

- **Pricing Configuration** – It is dynamically generated. Empty tables are populated based on user input during deployment via PlatformManager (for that reason product segmentation and pricing levels are handled in PlatformManager).
- **Data** – Mostly empty tables, which are fed by PSP Configuration: PXs, AnchorData PP, PricingExceptions PP, StockData PP, RoundingRules PP etc.

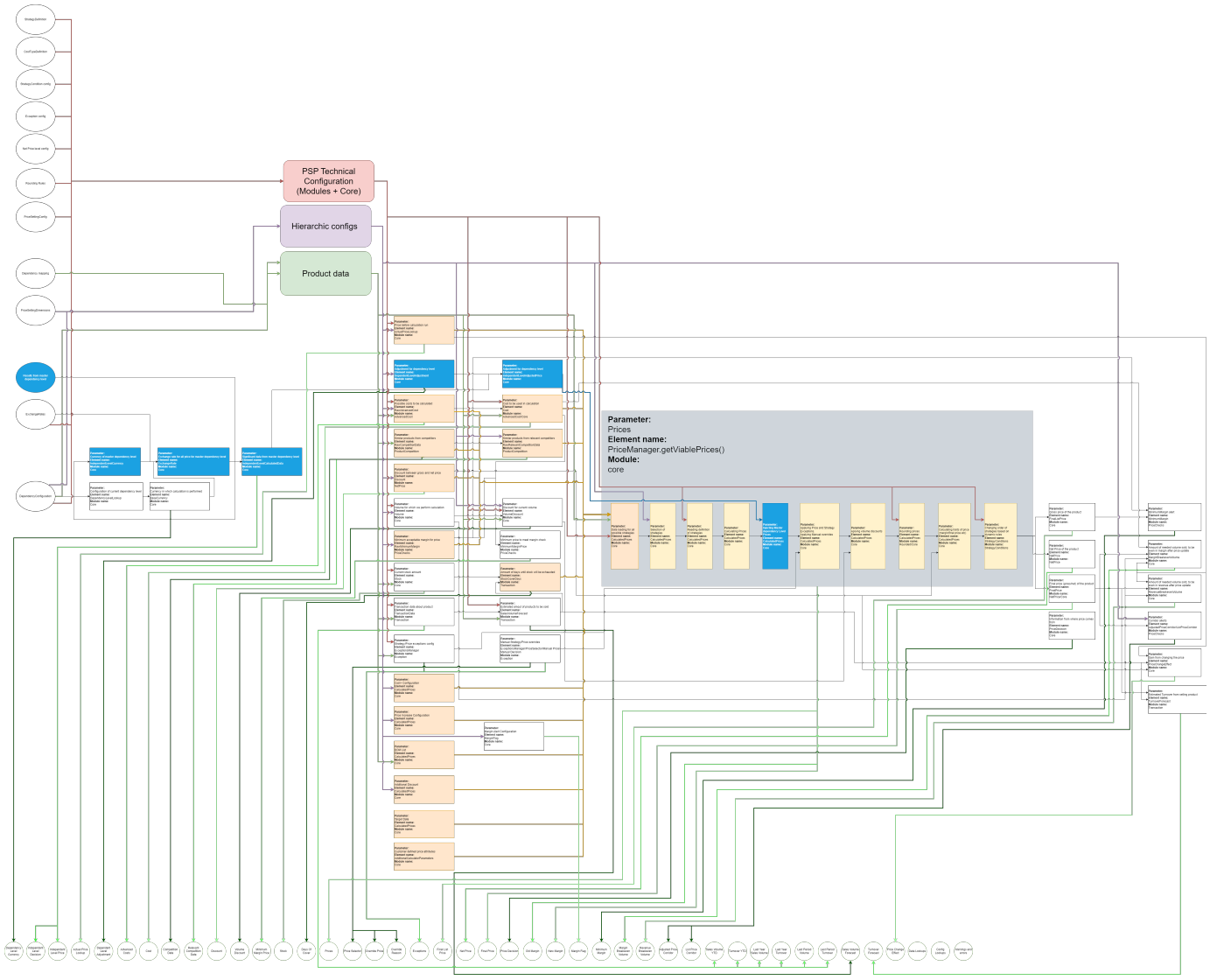
In this section:

Data Flow in Price Setting Package

The following diagram represents the data flow in the whole Price Setting Package (PSP). It looks intimidating, but it describes all the relations between different building pieces. It can help track where the results come from and how they are connected to each other.

Tips for reading the diagram:

- For simplification all configs are represented as 3 rectangles – Product Data, Hierarchical configs and PSP Technical Configuration.
- At the bottom, there are all elements visible to users from the PSP logic.
- Data only for Dependent Logic are shown in blue.
- Data used as input for strategy engines are shown in yellow.



Price Strategies

The main purpose of the Accelerate Price Setting Package is calculation and management of prices. That is why price calculation is one of the fundamentals in the package. Price Setting Package uses Price Strategies to calculate prices. From business perspective, price strategy represents the technical implementation of your pricing rules. When a price strategy is executed, it results in a price proposal for your product. You can have different price strategies for every product segment. We will calculate all of them and the one with the highest priority will end up as the final price proposal for your product.

Technically Price Setting Package uses Calculation Engines to calculate prices. A useable Pricing Strategy is a combination of Strategy Engine (some Groovy code) and StrategyDefinition PP (Price Parameter table combining the Strategy Engine with additional configuration parameters).

Pre-configured Price Strategies

The package comes with the following pre-configured Price Strategies:

Price Strategy	How it calculates prices	What data is used
	The minimum/average/maximum competition price. You can configure if you want to:	By default all competition data for the SKU is used. You can

Minimum Competition Based Price Average Competition Based Price Maximum Competition Based Price	• directly map to the competitor price or • reposition against it (using relative or absolute values).	limit it to only relevant competition data. The minimum margin and cost are passed to the engine, so you can configure it to skip competitors which you cannot afford to position against.
Recommended Retail Price	Recommended retail price coming from an external source.	Lookup in Product Extension with a list of Recommended Retail Prices.
Cost Plus	Calculation of Cost Plus Price. The provided example uses the relative plus factor (percentage) and applies it to the given cost base. It is possible to change it to an absolute value.	Product cost (or complex cost types), defined "plus" with absolute or relative values.
Price Increase	Increase of the previous price by a relative (percentage) factor. It is possible to change it to an absolute factor.	Actual price of the product and defined "increase" in absolute or relative values.
Kit Pricing	Kit Pricing calculates the price of a kit based on the prices of the sub-components. All of the sub-components have to be in the same PL/LPG.	BoM (Bill of Material) data to define sub-component relations.
Attribute Based Pricing	Attribute Based Pricing prices the products based on "value" of some product attributes. It takes the price from a defined reference product, and performs arithmetic operations (+, -, *, /) based on a defined formula and the values of some product attributes (e.g. "red" will result in + 3€, price will be multiplied by size, ...). These values can be direct numeric values (size, ...) or discrete values with assigned price impact.	Product reference to define the "base product" for a special product. You will also need: • List of attributes that should be considered in the price calculation • Translation of discrete attributes or ranges of numerical values to price impact values (if you have such) • Formula to calculate the result price based on reference price and the dedicated price impacts

Strategy Engines

These pre-configured examples are provided with the package.

If you need slightly different price strategies, please check how the strategy engines can be used and configured. They allow you to create various price strategies on your own. For details on the engines see:

- Adjustment Engine
- Attribute Based Engine

- Anchor Engine
- Competition Engine
- Kit Engine
- Lookup Engine
- Net Engine
- Custom Engines

Custom Strategies

When you need some other specific rule to calculate your price, you can "plug in" your own Custom Engines . A Custom Engine is basically some capsulated function that can be easily connected to the Price Setting Package. You can use the out-of-the-box engines and wrap them in your own engine or start completely from the scratch.

Dependent Price List Calculations

As you can see in Price Strategies, Price Setting Package can calculate different prices based on different pricing rules. You can define many pricing strategies; for each of them you have the following configuration options in StrategyDefinition PP:

- 'Level' describes where the definition is valid – for a dependent or parent price list. If you want a strategy to be valid in both scenarios, you need to create two entries.
- 'Overridable' describes if this strategy can be manually overridden by selecting other strategy or a manual price or using exception table.
- The remaining settings are used to order price strategies.

How Parent and Dependent Levels Are Calculated

Generally there is the following mechanism to calculate the prices. It is different for Parent and Dependent calculations.

Parent Price List Calculations

Parent price list:

- Calculates base strategies.
- Calculates standard strategies.
- Removes base strategies which returned no price.

The highest level strategy will be used as price proposal. All other strategies will be available in the "Prices" pop-up and in the strategy selection drop-down (when allowed, check PSP Override Module).

Dependent Price List Calculations

Dependent price list is more complex, since it has several configuration options which tweak the strategies order:

- The first strategy is "Parent Level Adjusted Price", which is a Final Price from the Parent price list for a given product adjusted by the dependency adjustment. It can be overridden by setting "No" for the "Prioritize Parent Level Price" column in the Strategy Selection Lookup PP. In this case, "Parent Level Adjusted Price" will be put at the end of strategies. It is a configuration on the product level.
- Base and standard product prices are calculated. However, they come in pairs with parent level adjusted prices, if the same strategy was calculated for a given product in the parent price list.
- Prices from the parent price list can be ignored on the dependent level by setting "Parent Level Only" to "Yes" in the StrategyDefinition PP.

- Dependent prices come in pairs with parent level adjusted prices (with dependent before parent), unless "Parent Level Priority" is set in the StrategyDefinition PP . Parent level priority is taken into consideration only when the strategy is defined for both dependent and parent level. It has to be set by an entry on the parent level.

Note:

- When strategy is calculated on the dependent level, it will use its local available data (as competition data, cost, ...).
- When strategy is taken from the parent level, it will take the parent level price and apply markup factor.

So you have to be careful to configure it correctly. For example, when you have Cost+ pricing strategy both for dependent and parent levels and you have the same cost, it will result in two different prices. One freshly calculated and one taken from the parent level and with an applied markup factor – which might seem unexpected.

Summary

These are your options to influence the Parent/Dependent behavior regarding importance of pricing strategies.

Flag	Where you find the flag	What it does
Prioritize Parent Level Price	PP Strategy Selection	Default is "Yes". When you change to "No", this will force the system to put the "Parent Level Adjusted Price" (= final approved price from parent level with the markup) to the end of the priority list.
Level	PP Strategy Definition	You can have "Parent" and "Dependent" as level in the definition of strategies. When you want to calculate them on both levels, you have to add them twice. Be aware that "calculate" means, that they are freshly calculated on the parent level. When you only want to take some price from the parent level and add a markup factor, you do not have to configure the strategy for the dependent level.
Parent Level Only		You can only set up this flag for "Level" = "Parent". When this is done like this, it will prevent the inheritance of this parent level price to the dependent level. So with this you force a price not to be taken over to the parent level and applied with the markup factor.
Parent Level Priority		When one strategy is calculated both on the parent and dependent levels, it will appear twice in the dependent level: 1. (Re)calculated on the dependent level base.

		2. Taken from the parent level and applied with markup factor.

Dependent Price Lists and Data Fallbacks

The Price Setting Package supports multiple price lists / price grids. Each of the lists is connected with one of the dependency levels: DependencyConfiguration PP. This is set on the PL/PG level as an input parameter, after choosing the proper logic. This structure is responsible for the following features:

- **Dependency mapping** – Allows having more than one data entry for each product, depending on the calculation context (e.g. different costs for web shop and brick and mortar shop),
- **Data fallback mechanism** – Allows incomplete data in case of granular pricing. This way only unique data needs to be put to the most granular dependency levels; the rest will fallback to the master dependencies.
- **Lookup keys config fallback mechanism** – Allows incomplete configurations for dimensional configurations. The mechanism is similar to the data fallback mechanism, with some more intuitive tweaks to configurations.
- **Master price adjustments** – Allows centralized approach to pricing where prices from the master price list are important for the dependent logic.
- **Grouping transaction data** – Allows the master price list to be completely valid. Transaction data of the dependent price lists will usually be (except for the HQ mode described below) also part of the master dependency.

In this section:

Introduction to Parent Selection Algorithm

This section explains how parent selection algorithm works and how to alter it using PSP Configuration, without needing to customize the algorithm's code, as that would be too low-level.

You need only to configure the hierarchy on the partition, and the algorithm will be applied automatically:

- Parent level prices are always selected from the parent PL/PG.
- Pricing Configuration Lookups automatically use fallback mechanism via the vLookupHierarchic function.
- Data Lookups may or may not use this feature, depending on the "Yes"/"No" value in the LookupConfiguration PP isUsingDependencyLevelHierarchy column.
- Transactions will always use this structure to include "children" transactions in the calculation.

Independent Logic

To set up independent price lists / price grids, the "IndependentPriceListLogic" logic must be used to run calculations. As the name suggests, these PGs/PLs are independent objects and all the calculation results depend only on the package configurations. This mode can be used to handle the simplest scenarios where only one dependency level exists or as a root for more complex independent-dependent hierarchies.

Dependent Logic

To set up dependent price lists / price grids, the "DependentPriceListLogic" logic must be used to run calculations. During PG/PL configuration you will be asked to select both current dependency level and parent dependency level. This connection will be used throughout the whole calculation to base prices on parent prices or to utilize hierarchical data fallbacks.

Dependency Mapping

To understand how to create distinct entries of the same product data which are unique in the context of a defined dependency level, see DependencyMappingConfig PP.

HQ Dependency Mode

If two dependencies have the same dimension and one depends on another, then we call such relationship an HQ relationship. HQ dependency levels will be skipped when looking for fallbacks but the master price might be looked up from such dependency.

Price Adjustments

Dependent price lists / price grids can use their master result prices as base for their own prices. Parent price is always taken from the first found master in the hierarchy but there are two exceptions:

- Parent logic does not expect a master price.
- If there is no master price list configured for the dependency level, then the master dependency is skipped. Such master dependency is called "virtual" and it exists only for fallback purposes.

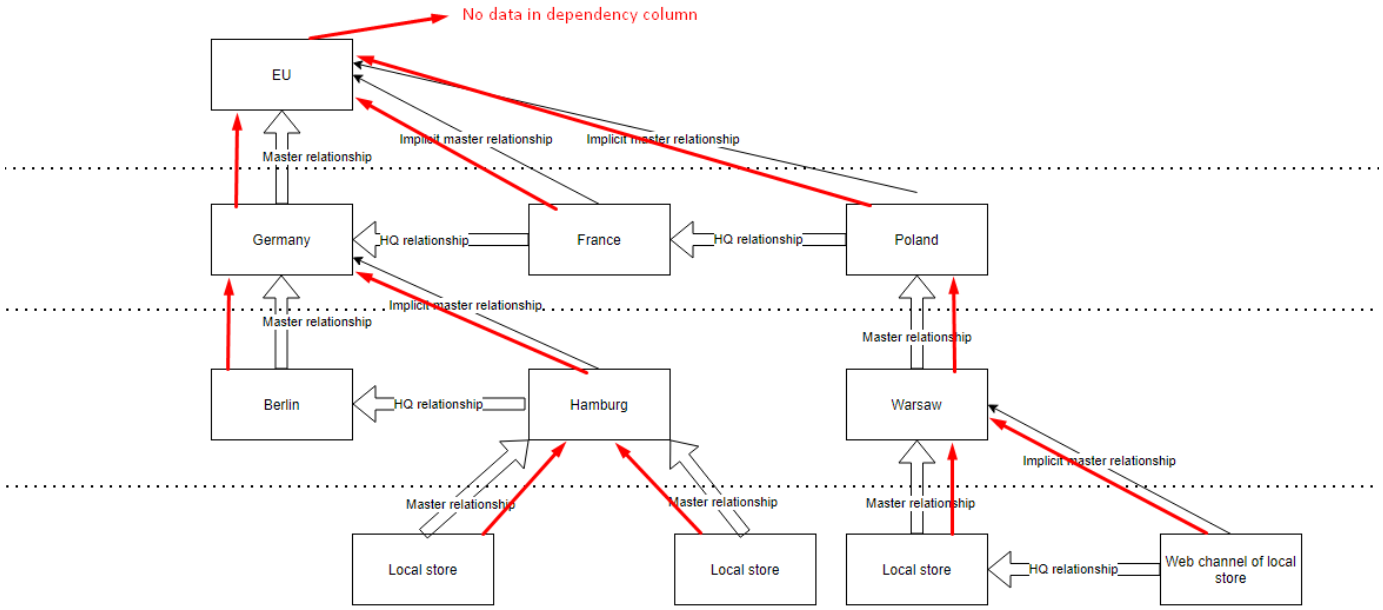
⚠ Currently, it is not possible to create a virtual dependency level on top of the hierarchy, as anything below the top uses the dependent logic – and expects to have a master price somewhere in the tree.

Data Lookup Fallbacks

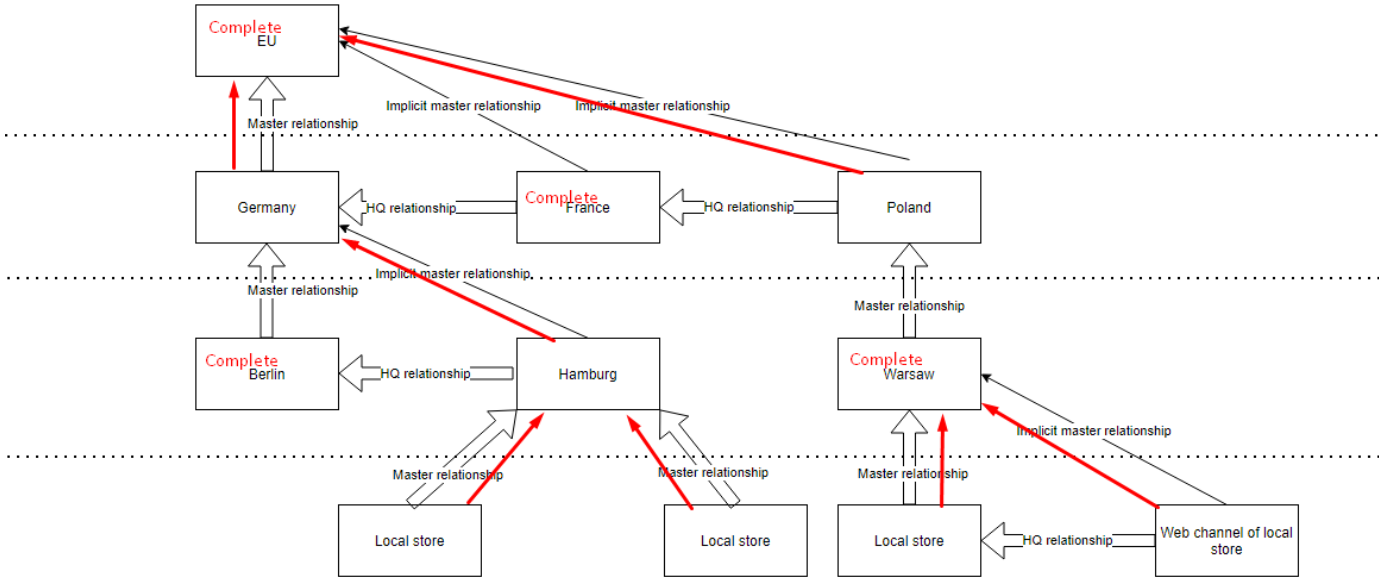
Fallbacks follow the following rules:

- Check if the current dependency level is complete.
  - If yes, abort the algorithm.
- Look up the master dependency level.
  - If the master dependency level does not exist, add empty mapping data as a possible fallback and abort the algorithm.
  - If the master dependency level is not in the HQ relationship with the current level, then add a master as a possible fallback.
- Repeat with the master dependency level as the current dependency level.

Example without the isComplete flags:



Example with the isComplete flags:

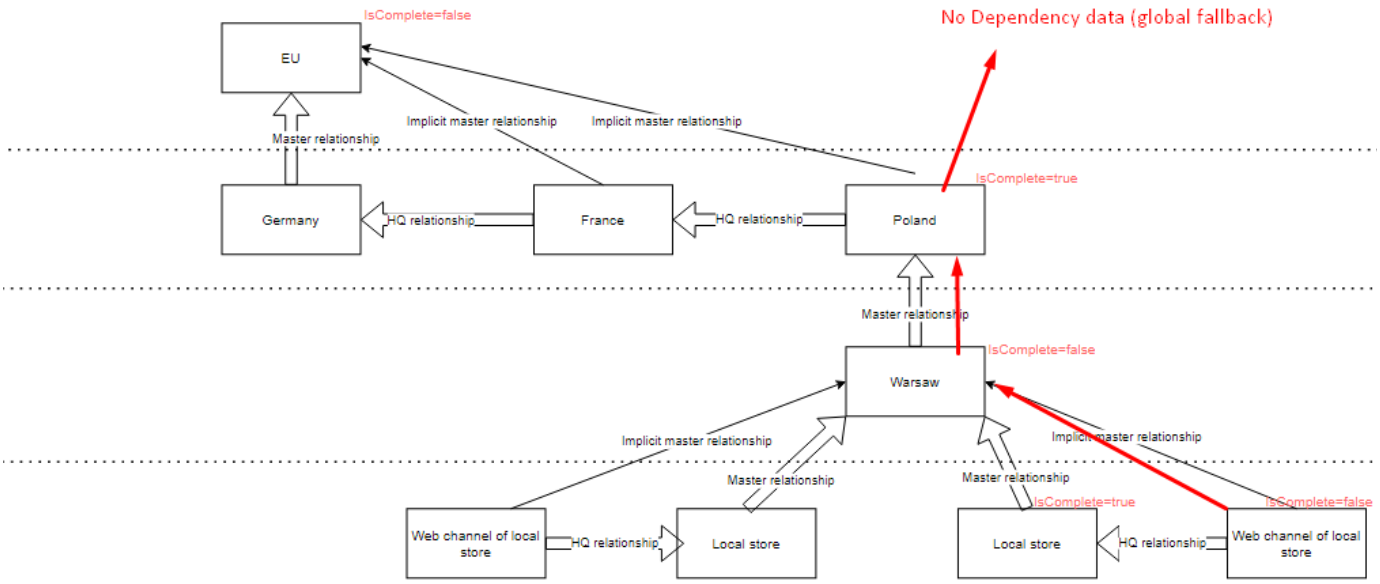


Lookup Keys Config Fallbacks

Comparison with data fallbacks:

- There is no need to configure dependency mapping for config tables.
  - Dependency mapping is always set to:
    - Dependency Property: DependencyLevelName
    - Mapping type: Table
  - Config tables have pre-generated price parameters.
- There is always a global, per-instance PP with fallback. It can be “turned off” by leaving it empty.

Example:



For a given configuration, when looking up the CostPlus config for a web channel of a local store, the algorithm will look for configs in PPs in this order:

- WebchanneloflocalstoreCostPlus
- WarsawCostPlus
- PolandCostPlus
- CostPlus

Transaction Data Fallback

Transaction data from PSP Transaction Module is a special case. More information can be found here.

Dependency Mapping

Dependency mapping defines how lookup data are filtered when loaded from a dependent price list.

- Dependency Mapping Types
- Dependency Mapping Mechanism

Dependency Mapping Types

Dependency mapping defines how different lookup data will be filtered when loaded from a dependent price list. It is available in the DependencyMappingConfig PP and it directly impacts configuration found in PriceSettingConfig PP.

Lookup

One source table for all dependencies.

SKU	Dependency	Cost
PD-0001	Global	10
PD-0001	Asia	11
PD-0001	EU	12
PD-0001	US	13
PD-0002	Global	14
PD-0003	Asia	15

Example: For a PL/PG which has a dependency level Asia, the PD-0001 SKU will have a value of 11.

Table

A source table for each dependency.

Global table

SKU	Cost
PD-0001	10
PD-0002	11
PD-0003	12

US table

SKU	Cost
PD-0001	11
PD-0002	12
PD-0003	13

Asia table

SKU	Cost
PD-0001	12
PD-0002	13
PD-0003	14

EU table

SKU	Cost
PD-0001	13
PD-0002	14
PD-0003	15

Example: For a PL/PG which has a dependency level Asia, the PD-0001 SKU will have a value of 12.

To use this type, change the name of the source table defined in PriceSettingConfig PP: the value should include the `<<DependencyPreference>>` placeholder. It will be replaced with the dependency property defined by the dependency mapping mechanism.

For example: for cost configuration which has dependency mapping type `Table`, the source table is `Product Costs <<DependencyPreference>>`, and the dependency value is `DE`, the dependency mapping mechanism will search for a table named `Product Costs DE` and then perform the lookup.

⚠ The Dependency Field in DependencyMappingConfig PP is omitted in this type.
Table dependency does not work for competitors because Pricefx has only 1 PCOMP table.

None

A source table for all dependencies.

SKU	Cost
PD-0001	10
PD-0001	11
PD-0001	12
PD-0001	13
PD-0002	14
PD-0003	15

Example: For a PL/PG which has a dependency level Asia, the PD-0001 SKU will have a value of 10.

When a data element has the mapping type = None, it has no dependency mapping or hierarchy, or fallback. In case there are multiple records, it will fetch the first found. And when a table name is dependency wildcard but it has the mapping type = None, it uses the table with no wildcard.

Example: <<DependencyPreference>>CostData => CostData.

⚠ The Dependency Field and Mapping Source Field in DependencyMappingConfig PP are omitted in this type.

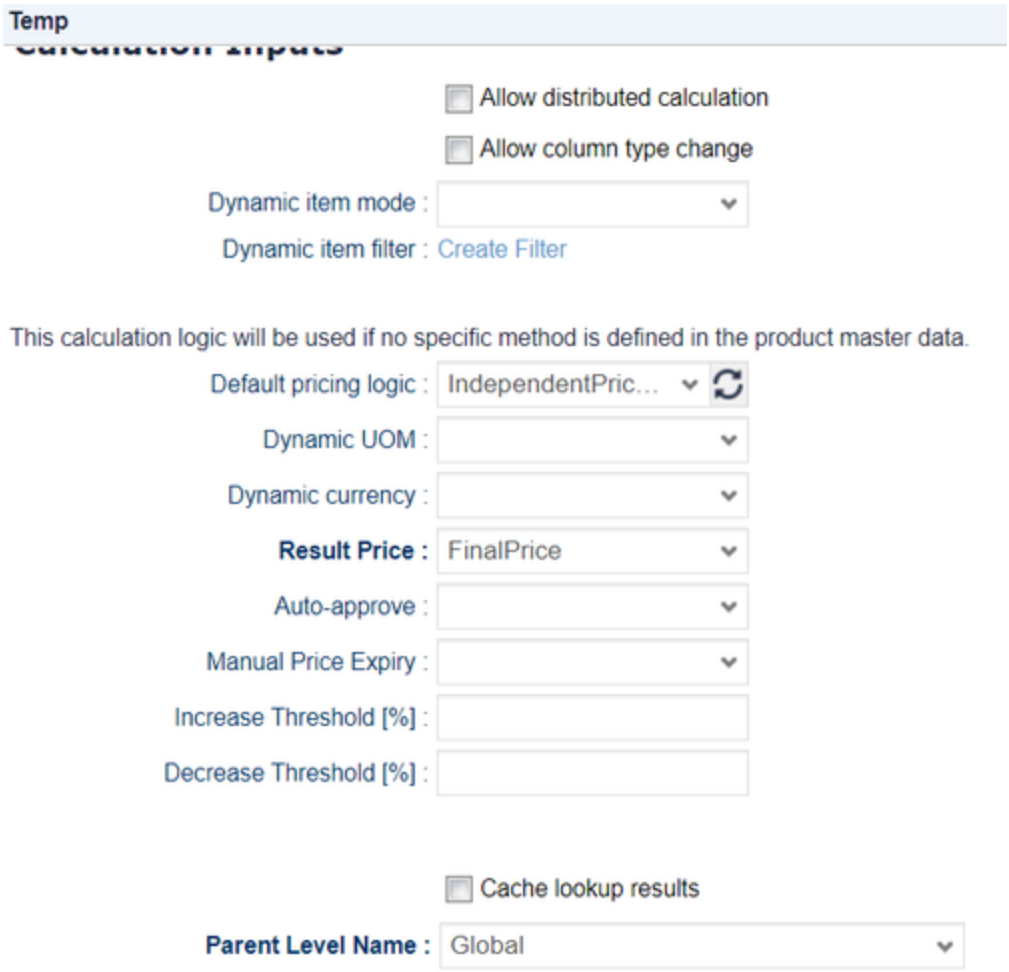
Dependency Mapping Mechanism

Here you will find an example of Cost mapping with all different dependency mapping types.

Example 1

Select Dependency

When creating a PL/PG with Price Setting Package, there is an input named **Parent Level Name** to select the dependency.



The selected dependency is *Global*. It will then search in DependencyConfiguration PP for the corresponding record.

Dependency Level Name	Depends On	Source Type	Source...	Dimension	Currency	Is Complete	ISO Code	Sales Org
Global	Independent	*	*	Area	USD	No	GG	SOOO

With "Lookup" Mapping Type

In DependencyMappingConfig PP set the following:

Name	Dependency Field	Type	Mapping Source Field
Cost	DependencyLevelName	Lookup	DependencyLevelName

- **Type** is set to *Lookup* to indicate the mapping type.
- **Dependency Field** points to DependencyLevelName. In the Select Dependency step, the selected dependency is *Global*, and in DependencyConfiguration PP, the corresponding row has the DependencyLevelName field = Global. Therefore, the dependency value will be *Global*.
- **Mapping Source Field** points to DependencyLevelName, it will search in the Data Source table for rows that have DependencyLevelName field value equaling to the dependency value (*Global* in this case).

The source table is defined in PriceSettingConfig PP:

Key	Condition	Type	Source	Source Table	Source Field	Source Field 2
Cost	*	Lookup	PX	CostData	Cost	Currency

It will search the PX CostData table for records that have DependencyLevelName field equaling to Global and get a value from the column referred in the Source Field as cost value (*Cost* field).

With "Table" Mapping Type

In DependencyMappingConfig PP set the following:

Name	Dependency Field	Type
Cost	DependencyLevelName	Table

- **Type** is set to *Table* to indicate the mapping type.
- **Dependency Field** points to DependencyLevelName. In the Select The Dependency step, the selected dependency is *Global*, and in DependencyConfiguration PP, the corresponding row has the DependencyLevelName field = Global. Therefore, the dependency value will be *Global*.

The source table is defined in PriceSettingConfig PP:

Key	Condition	Type	Source	Source Table	Source Field	Source Field 2
Cost	*	Lookup	PX	<<DependencyPreference>>CostData	Cost	Currency

It will search the the PX *Global* CostData table for records and get a value from the column referred in the **Source Field** as cost value (*Cost* field).

With "None" Mapping Type

If you would like to search the PX CostData table and get a value from the column referred in the **Source Field** as cost value (*Cost* field).

Providing we have the source table defined in PP PriceSettingConfig this way

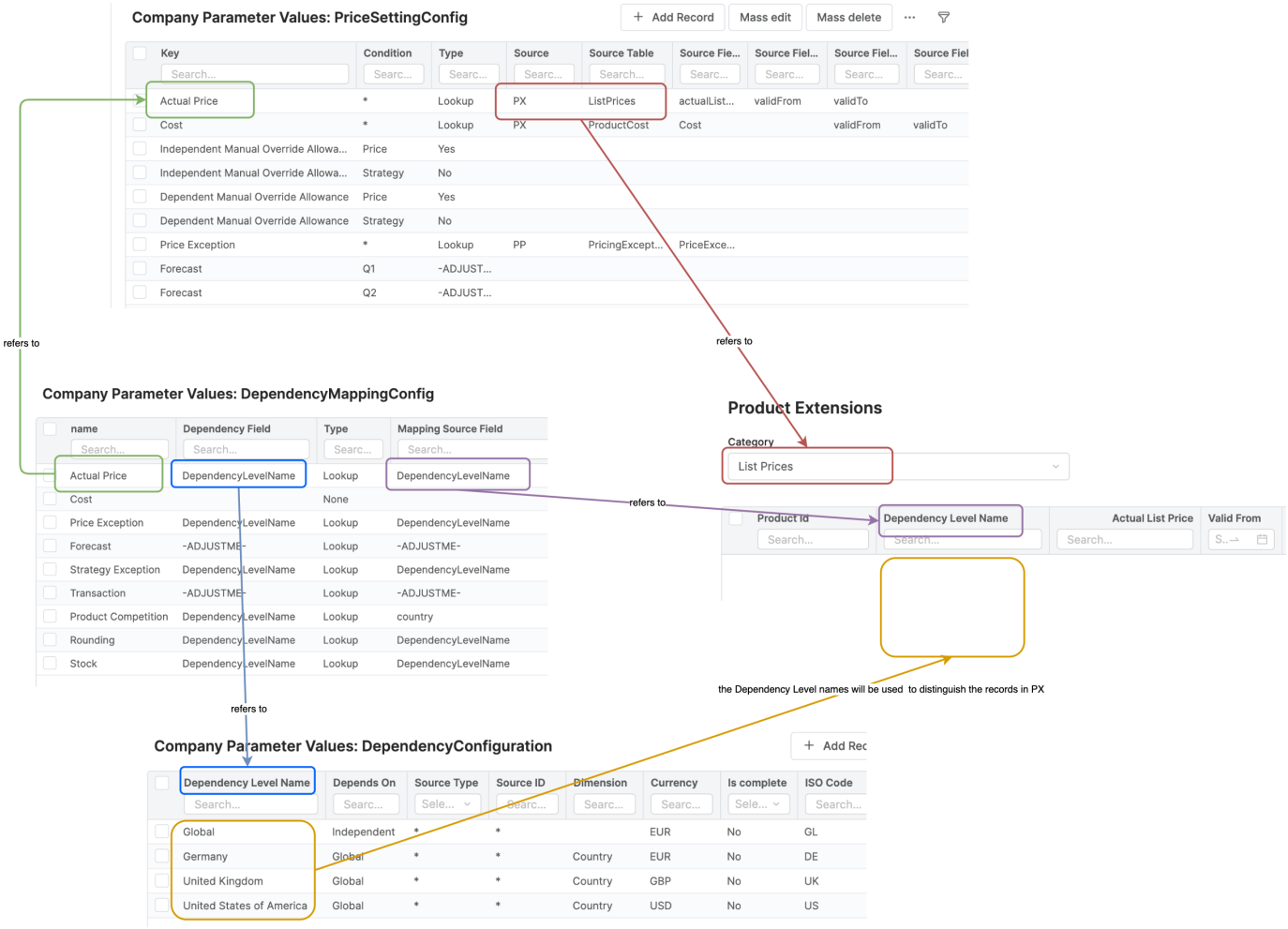
Key	Condition	Type	Source	Source Table	Source Field	Source Field 2
Cost	*	Lookup	PX	CostData	Cost	Currency

then in DependencyMappingConfig PP table, set the **Type** to *None*, in the following way:

Name	Dependency Field	Type
Cost		None

Example 2

Example of dependency mapping of the "Lookup" type.



Data Lookups

Price Setting Package uses ConfigurableLookupsUtil for data lookups. Most of them are "standardized", they are called in the same way. Some of them differ, due to different sources or data structure.

- Standardized Lookups
- Special Cases Lookups

Standardized Lookups

The flow of standardized lookups:

1. Lookup Configuration is parsed. It can either be read from PP or passed as a function parameter.
2. Table name (or names) are determined.
3. Table type is determined.
4. Once for a batch (by default 200 products), a DB lookup is performed.
    a. Custom Filter is parsed to a Filter object.
    b. SKU and secondary key (with optional transformation) are parsed into a batch Filter.
    c. If using PSP DependencyMapping, corresponding filters are applied.
    d. FieldNames are translated into attributesID.
    e. Lookup is performed.
        i. There is a check if the data lookup threw any exception. Any exceptions are propagated further for every product in the batch so that config issues are easier to debug.

    f. Data is grouped per SKU and secondary key (if the lookup did not fail).
    g. Data is saved into *api.global*.
5. Data for a current item is read.
6. If using PSP DependencyMapping, the best matching dependency level is selected.
7. Data is transformed to be under keys specified in selectedFields.
8. Data checks are performed, according to the checksConfig configuration.
9. Data lookup is registered for debugging (registering keeps the data for other products in the batch).
10. Data for SKU is read.

Special Cases Lookups

Actual Price

Technically, the Actual Price PX lookup is standardized. However, from the configuration point of view, the flow of data reading is non-standard when using PL and PG sources.

PL and PG do not use batching. PG is not even a lookup as such, since it reads data from the previous run of the same price grid.

Preference: Actual Price Lookup

Transaction

- Hierarchy works differently for Transaction data (it goes down for all sub-levels of the dependency, instead of working as a fallback).
- A lot of data will be returned for the given time period, there is a "date overlap" issue.
- Lookup manager supports `api.stream` calls, but transaction data might be stored in a Datamart or Data Source.

Preference: Transaction Lookup

Forecast

- Hierarchy works differently for Transaction data (it goes down for all sub-levels of dependency, instead of working as fallback).
- A lot of data will be returned for a given time period, there is a "date overlap" issue.
- Lookup manager supports `api.stream` calls, while transaction data might be stored in Datamart or Data Source.

Preference: Forecast Lookup

Competition Data and Strategy Conditions

While using Configurable Lookups, Competition Data and Strategy Conditions configurations are hardcoded in Groovy logic. These can be changed, but doing so will invalidate some of the wizards.

Calculation Engines

Calculation Engines provide plug-in/plug-out methods for price calculation which can be used in Price Setting Package and other projects as a standalone library.

Any given engine can be passed a simple Price Parameter (type MATRIX) with keys and values as specified in the Additional Configuration column in detailed documentation of the engine.

All available strategies are configured using the StrategyDefinition Price Parameter. For more details see Other Configs and documentation for each engine in subsequent sections. For additional documentation on the calculation engine, see Calculation Engine Library.

Here is a short explanation of how the engines work and what they can be used for:

Engine Name	Functionality	Sample Supported Strategies
Adjustment Engine	Takes one price as a base and applies a factor to it.	• Cost Plus • Price Increase
Anchor Engine	Calculates prices based on the price of another SKU. Note: This engine is deprecated.	• Anchor Pricing
https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/2517468375/Attribute+Engine	Calculates prices based on the price of another SKU and current SKU's attributes impact value.	• Anchor Pricing
Competition Engine	Calculates prices according to existing competition prices.	• Competition Based Pricing
Kit Engine	Calculates prices of a kit based on subcomponent prices.	• Kit Pricing
Lookup Engine	Looks up prices from an existing table.	• Recommended Retail Price • Promotion Pricing • Everything that works with a lookup of a stored price
Net Engine	Calculates a gross price of a product, based on a specific "pocket price" and discounts. The pocket price is always looked up using the LookupEngine.	• Net Pricing
Custom Engines	Any custom library method can be used as an engine as long as it takes proper parameters and returns a proper result.	• Basically anything

Currently, with our standard out-of-the-box configuration, the package comes preconfigured with the following strategies:

- Minimum Competition Based Price
- Average Competition Based Price
- Maximum Competition Based Price
- Recommended Retail Price
- Cost Plus
- Price Increase
- Kit Pricing
- Anchor Pricing

Adjustment Engine

Adjustment Engine takes care of simple "Value + Adjustment" calculations. It is used to implement strategies such as Cost Plus or Price Increase.

Input Parameters

Input	Type	Description
Value	BigDecimal	Used as a base for adjustments.
Adjustment	BigDecimal	Used as an adjustment. Can be absolute value or percentage – it depends on the mode selected in Additional Engine Configuration. For percentage based calculations the expected range is 0.0–1.0.

Additional Engine Configuration

This engine needs an additional simple configuration Price Parameter where all Configuration Options are present as keys.

Configuration Option	Expected value	Description
Calculation Mode	"Absolute"	Result is: `Value + Adjustment`
	"Percentage"	Adjustment is a percentage. Result is: `Value * (1 + Adjustment)`
	"SellingPrice"	Adjustment is a percentage. Result is: `Value / (1 - Adjustment)`

Default Strategy Calculation Parameters

For Cost Plus: `PRODUCT_COST`, `PLUS_FOR_PRODUCT`
For Price Increase: `BASE_PRICE`, `PRICE_INCREASE`

Attribute Based Engine

The idea of attribute-based pricing is to define the price based on product attributes like color, weight etc. It is based on the Anchor Follower approach: take the price of another SKU, then modify it to get the final price.

The engine supports only one level which means there can only be pairs like SKU A SKU B. A chain of anchors like SKU A SKU B SKU C ... is not supported.

Note: If the used Price List or Live Price Grid is of the Matrix type, the engine assumes that the second key is used for storing the volume information and will search for an anchor product with the secondary key equaling to 1.

In this section:

Understanding the Calculation Mechanism

The formula is defined in the AttributeBasedPricingRules PP. It takes the price of the anchor SKU as the basis, then calculates from left to right; there are no additional math operations. For each attribute in the rule:

- It gets the current product attribute value. The way to find it is defined in the PricingAttributes PP.
- It gets the impact of the above value. The impact value can be Value-Based or Interval-Based. It is defined in the ValueAttributesConversion PP or IntervalAttributesConversion PP accordingly.
- It calculates with the impact value.

Warnings

- A rule is considered to be invalid if it is not a continuous string or it is a continuous string but ends with an operator. Example:

- Divide by zero is not allowed.
- If it is configured to have validity periods and the data overlap, an exception is thrown.
- If it is configured to have dependency mapping but no data match the criteria, it gets the first one with the null value in the filter field.
- Any invalid field name / rule name / attribute name / ... is not allowed.
- This engine uses the re-run ("marked as dirty") functionality of Pricefx. You should not mix products that are used by other engines that mark items as dirty (e.g. Kit Engine) in one PL/LPG. It can lead to undefined behavior and wrong results.
- This engine only works correctly when all related products are in the current calculated Price List or Live Price Grid.

How to Use It

Input Parameters

Input	Type	Description
SKU	String	SKU of the calculated product.
Final List Price Element Name	String	Name of the element that keeps the Final List Price. It will be used for the Anchor price lookup if the calculation is in the Net mode.

Final Price Element Name	String	Name of the element that has the Final Price. It will be used for Anchor price lookup if the Final List Price element is empty. It usually happens during the Gross calculation.
Dependency Properties	String	Properties of current dependency.
Strategy Name	String	Name of the strategy. It is used to connect the name of the attribute based strategy (in PP AttributeBasedPricingRules*).* You can use the constant `STRATEGY_NAME` to connect it with the name of the strategy.

Additional Engine Configuration

This engine needs an additional simple configuration Price Parameter where all Configuration Options are present as keys.

Configuration Option	Expected Value	Description
Source Type	Allowed values: • PX • PP • P • PXREF	Defines where the anchor data is kept.
Source Name	ExampleTableName	Name of the data table. Expected only when PX or PP Source Type is used.
Anchor Field Name	ExampleAnchorColumn	Name of the column that contains SKU of the anchor product.
Sku Field Label	ExampleSkuColumn	Name of the column that contains SKU of a currently calculated product.

Default Strategy Calculation Parameters

```
SKU,FINAL_LIST_PRICE_ELEMENT_NAME,FINAL_PRICE_ELEMENT_NAME,
DEPENDENCY_PROPERTIES,STRATEGY_NAME
```

Define Attribute Data (with Sample Data)

AttributeBasedPricingRules PP

Name	Operator #1	Pricing Attribute #1	Operator #...	Pricing Attribute #...

Attribute-Based Simple	+	Color	+	Size
...	...	...	/	0

- Fields:
  - Name – Rule name
  - Operator #XX – Supported operators: +, -, *, /
  - Attribute #XX – Existing pricing attribute name in PricingAttributes PP

PricingAttributes PP

Pricing Attribute (Key)	Type	Source Type	Source Name	Source Field	Dependency Field	Dependency Mapping Type	Mapping Source Field	ValidFrom Field Name	ValidTo Field Name
Color	Value	P		Color					
Size	Interval	PX	Additional Product Data	Size	Country	Lookup	Country	ValidFrom	ValidTo
Weight	Direct Value	P		Weight					

- Fields:
  - Pricing Attribute – Name of the attribute
  - Type – Attribute value type
    - Value – Single value
    - Interval – Value in a specified range
    - Direct Value – Impact value is also the attribute value, no conversion for this type. The value data type can only be a number.
    - Direct Value and Interval Type only works for numeric values.
  - Source Type – Source table type
    - P / PX / PP
  - Source Name – Name of the source table
  - Source Field – Name of the field in the source table to take attribute value
  - Dependency Field – Name of the field in the dependency configuration table to take dependency value
  - Dependency Mapping Type – Type of dependency mapping
    - Lookup / Table
    - Fallback on dependency mapping – Returns one with “null” in the dependency field when there is no specific one
  - Mapping Source Field – Name of the field in the source table to take the matching value
  - Valid From Field Name – Name of the field in the source table to take the beginning date of the validity period
  - Valid To Field Name – Name of the field in the source table to take the end date of the validity period

ValueAttributesConversion PP

Pricing Attribute	Pricing Attribute Value	Price Impact Value
Color	red	3

Color	blue	2
Color	2	1
Color	<<fallback>>	1.5

Mapping one attribute value to one impact value. The attribute value can be both string and number.

IntervalAttributesConversion PP

Pricing Attribute	Pricing Attribute Value From	Pricing Attribute Value To (including)	Price Impact Value
Size	0	10	1.2
Size	10	50	1.4
Size	50	999999999	1.5

Mapping many attributes value to one impact value. The attribute value can only be number.

Anchor Engine

⚠ This engine is deprecated. Use the Attribute-based engine instead.

Anchor Engine calculates a price for a given product based on another product's price (Anchor) and anchor factor by which we multiply this price. The formula is: `Price = AnchorPrice * (1 + AnchorFactor)`

This engine supports only one level of connection. It means you cannot specify an anchor for an anchor, etc. In addition - this engine works properly only when all connected products are added to the same Price List or Live Price Grid.

Important notes:

- This engine uses the re-run ("marked as dirty") functionality of Pricefx. It means that you should not mix products that are used by other engines that mark items as dirty (e.g. Kit Engine) in one PL/LPG. It can lead to undefined behavior and wrong results.
- Because this engine doesn't return a price after the first calculation, if it is used as Base Strategy, it will not be shown in the "Prices" popup. The strategy will appear only when the second calculation run completes.
- If the used Price List or Live Price Grid are of the Matrix type, the engine assumes that the secondary key is used for storing the volume information and will search for an anchor product with the secondary key equaling to 1.

Input Parameters

Input	Type	Description
SKU	String	SKU of the calculated product.
Final List Price Element Name	String	Name of the element that keeps the Final List Price. It will be

		used for the Anchor price lookup if the calculation is in the Net mode.
Final Price Element Name	String	Name of the element that has the Final Price. It will be used for Anchor price lookup if the Final List Price element is empty. It usually happens during Gross calculation.

Additional Engine Configuration

This engine needs an additional simple configuration Price Parameter where all Configuration Options are present as keys.

Configuration Option	Expected value	Description
Source Type	Allowed values: • PX • PP • P • PXREF	Defines where the anchor data is kept.
Source Table	ExampleTableName	Name of the data table. Expected only when PX or PP *Source Type* is used.
Anchor Label	ExampleAnchorColumn	Name of the column that contains SKU of the anchor product.
Factor to Anchor Field Label	ExampleFactorColumn	Name of the column that contains the value used as factor multiplication.
Sku Field Label	ExampleSkuColumn	Name of the column that contains SKU of a currently calculated product.

Default Strategy Calculation Parameters

```
SKU,FINAL_LIST_PRICE_ELEMENT_NAME,FINAL_PRICE_ELEMENT_NAME
```

Competition Engine

Competition Engine supports various options for price calculation based on competitor prices (defined through the engine's Additional Configuration PP table).

Competitor price can be selected based on one of two approaches:

- Competitor Position - Select one competitor to align the price with.

    - Min/max - Select a minimum/maximum available competitor price.
    - min + X / max - Y - Select minimum/maximum competitor price position and adjust it by the given value.
    - 10%, 50%, 70% - Select the target competitor based on the provided percentage. The formula for the calculation is: `TargetCompetitorPosition = NumberOfCompetitors * Percentage`
- Price Position - Select a price at the given percentage point. A value of 0% matches the lowest competition price and value of 100% matches the highest competition price. The formula for calculation is: `Price = CompetitorMinPrice + (CompetitorMaxPrice - CompetitorMinPrice) * Percentage`

Note: Competitor Position and Price Position cannot be used at the same time.

After the competitor price has been selected, the engine will find the corresponding competitor name if in the "Competitor Position" mode.

Then you can additionally "reposition" the price by:

- Percentage - Modifies the price by a provided percentage. To make the price 5% cheaper, you use `-5%`; the same applies for a positive adjustment.
- Absolute value - Modifies the price by an absolute value. To make the price 10 units cheaper, you use `-10`; the same applies for a positive adjustment.

The engine supports Force Margin Check to verify that the selected competitor price is affordable. You can set values "Yes/No" in the table to turn the functionality on or off.

If the new competitor price is affordable after applying Force Margin Check, the engine will find the corresponding competitor name again and calculate the total of skipped competitors counting from the old competitor price to the new one. Finding the corresponding competitor name is not relevant for the "Price Position" mode. In such case, the lowest affordable price will override the current price if in range of available prices.

Note: Order of operations is:

1. Price calculation by selected mode
2. Margin check
3. Reposition

Input Parameters

Input	Type	Description
Competitor and Prices	List	List with prices and competitor name from competitors that should used for processing.
Minimum Margin Price	BigDecimal	Price used for affordability check, nullable.

Additional Engine Configuration

This engine needs an additional simple configuration Price Parameter where all Configuration Options are present as keys.

Configuration Option	Expected value	Description

Competitor Position	Allowed values: • `min` - Selects the competitor with the lowest price. • `max` - Select the competitor with the highest price. • `min + X` - Selects the competitor with the lowest price and adjusts the position by X. • `max – X` - Selects the competitor with the highest price and adjusts the position by X. • `40%` - Selects the competitor whose position is at 40th percentile of the whole competitor range.	Selects the target competitor to compare to. Case insensitive.
Price Position	Value in range: `0% - 100%`	Directly selects the competitor price based on the percentage provided and the range of competitor prices.
Repositioning %	Value in range: `0% - 100%`	Adjusts the selected competitor price by the given percentage.
Repositioning Abs	Absolute value. Can be negative.	Adjusts the selected competitor price by the given absolute value.
Force Margin Check	Allowed values: • Yes • No	Checks whether the selected competitor price is affordable based on Minimum Margin Price. If it is not, the next competitor price / affordable price is selected until an affordable price is found. If no such price is found, the exception is thrown.

Relevant Competitors Definition

You can decide if you want to use all existing competition data, or if you want to define relevant competitors. Definition of relevant competitors can be done on the Lookup Key Level. We support definition of a list of competitors. You can decide if these competitors should be used or excluded.

You can define relevant competitors in PP "RelevantCompetitionData". It has to be filled as followed:

Configuration Option	Expected Value	Description
Lookup Keys	Values of the Lookup Key	Keys in PP are the selected Lookup keys.
Relevant Competitors	"yes" or "no"	• yes - relevant competitors are defined

		• no - the list of competitors is excluded
Competitor #1 ... Competitor #29	Name of Competitor	You can define up to 29 competitors to be considered as relevant or excluded.

Default Strategy Calculation Parameters

`COMPETITOR_PRICES (or RELEVANT_COMPETITOR_PRICES), MINIMUM_MARGIN_PRICE`

Kit Engine

Kit Engine calculates a price for a given product based on subcomponents and quantities defined in a standard BOM Data table.

A Kit Price is a sum of all of its subcomponent prices multiplied by provided quantities. There is no limit on how many levels of "subcomponent of subcomponent" are defined. If more than one level is present, we sum all the prices "at the bottom of the tree" using proper quantity factors.

The engine runs a cycle detection algorithm on the input BOM data. It throws an exception if a cycle is found.

This engine works properly only when all connected products are added to the same Price List or Live Price Grid.

⚠ Important notes:

- This engine uses the re-run ("marked as dirty") functionality of Pricefx. It means that you should not mix products that are used by other engines that mark items as dirty (e.g. Anchor Engine) in one PL /LPG. It can lead to undefined behavior and wrong results.
- Because this engine doesn't return a price after the first calculation, if it is used as Base Strategy, it will not be shown in the "Prices" popup. The strategy will appear only when the second calculation run completes.
- If the used Price List or Live Price Grid are of the Matrix type, the engine assumes that the secondary key is used for storing the volume information and will search for a subcomponent product with the secondary key equaling to 1.

Input Parameters

Input	Type	Description
SKU	String	SKU of the calculated product
BOM List	List	BOM List for the currently calculated product as returned by `api.bomList()` or in the same format.
Final List Price Element Name	String	Name of the element that keeps the Final List Price. It is used for subcomponent price lookups if the calculation is in the Net mode.
Final Price Element Name	String	

		Name of the element that has the Final Price. It is used for subcomponent price lookups if the Final List Price element is empty. It usually happens during Gross calculation.

Additional Engine Configuration

This engine does not have any additional configuration.

Default Strategy Calculation Parameters

`SKU,BOM_LIST,FINAL_LIST_PRICE_ELEMENT_NAME,FINAL_PRICE_ELEMENT_NAME`

Lookup Engine

Lookup Engine can be used for any strategy that has a static price that needs to be fetched from an existing table, e.g. Promotion Price or Recommended Retail Price.

It utilizes additional filtering based on:

- Target Date – Applied to "Valid From Field"/"Valid To Field" fields from the input configuration.
- Additional Filter Values – Passed values are OR'ed together and applied to the "Additional Filter Field" field from the input configuration. By default in the Price Setting package it uses all not-null.

At the end, we get a result for a given SKU, within the given validity dates and matching additional filter.

Input Parameters

Input	Type	Description
SKU	String	SKU of the calculated product.
Target Date	Date	Date used for the ValidFrom /ValidTo filtering.
Additional Filter Values	List	Used to pass values that will be used as additional filters. Empty by default.

Additional Engine Configuration

This engine needs an additional simple configuration Price Parameter where all Configuration Options are present as keys.

Configuration Option	Expected value	Description
Source Type	Allowed values: • PX • PP	Defines where the data is kept. For the PP type the engine expects a MATRIX table (Value Type: MATRIX, MATRIX2, MATRIX3, or MATRIX4) with SKU in the "name" column.
Source Table	ExampleTableName	Name of the data table.

Source Field	ExampleSourceField	Name of the column that contains the price within the table.
Valid From Field	ExampleFromColumn	Name of the Date type field.
Valid To Field	ExampleToColumn	Name of the Date type field.
Additional Filter Field	ExampleAdditionalFilterColumn	Name of the additional filter column. It can be any string that can match some data passed in the Additional Filter Values list.

Default Strategy Calculation Parameters

`SKU,TARGET_DATE,DEPENDENCY_INFORMATION_VALUES`

Net Engine

Net Engine calculates a gross price of a product based on a specific "pocket price" and discounts. The pocket price is always looked up using the Lookup Engine, so what this engine does is basically reverting any discounts that were applied to it.

Input Parameters

Input	Type	Description
SKU	String	SKU of the calculated product.
Target Date	Date	Date used for the ValidFrom/ValidTo filtering.
Discounts	List	Discounts that you want to "reverse".
Additional Filter Values	List	Used to pass values that will be used as additional filters. Empty by default.

Additional Engine Configuration

This engine needs an additional simple configuration Price Parameter where all Configuration Options are present as keys.

Configuration Option	Expected value	Description
Source Type	Allowed values: • PX • PP	Defines where the data is kept.
Source Table	ExampleTableName	Name of the data table.
Source Field	ExampleSourceField	Name of the column that contains the price within the table.

Valid From Field	ExampleFromColumn	Name of the Date type field.
Valid To Field	ExampleToColumn	Name of the Date type field.
Additional Filter Field	ExampleAdditionalFilterColumn	Name of the additional filter column. It can be any string that matches some data passed in the Additional Filter Values list.

Default Strategy Calculation Parameters

`SKU,TARGET_DATE,DISCOUNTS,DEPENDENCY_INFORMATION_VALUES`

Custom Engines

In addition to using the predefined engines, you can also define your own strategies.

All you need to do is to place a function path to the function in some external Groovy Library (instead of an engine name in the PP StrategyDefinition), e.g. `libs.MyLib.MyElement.MyFunction`.

This function should return a calculated price or throw an exception. Its message will be shown in the PL /LPG.

Users may add their own parameters for the custom engine in the `AdditionalCalculatorParameters` element, according to the Groovy Documentation. All parameters should be closures that will be executed dynamically when needed. If the user data are already fetched and used in the PSP flow, we recommend using simple Closure, for example `{return myVariable}` to avoid fetching data again.

Possible return values of the engine-like function:

- Price as BigDecimal
- Thrown exception (will be handled and shown in Prices popup)
- Map with keys:
  - "price" - Price as BigDecimal.
  - "message" - Message shown in the Prices popup.
  - "messageType" - How message/price will be shown. Possible values are "Info", "Warning" and "Critical".

See also;

- How to Create and Set up New Price Calculation Strategy
- Sample configuration of custom engine (Pricefx staff only)

Volume Breakdown

It is possible to run a PL/LPG with a volume breakdown. This feature allows you to apply different configurations per volume, depending on the quantity. To use this feature, utilize the secondary key in Matrix PL/LPG. The secondary key set will be the list of volumes.

Also, a calculation item in the PL/LPG with the volume equaling to 1 will always be added. It is to keep the default price without the volume discount applied and it will be used as the parent item price in dependent PL/LPG calculations.

Discount

The calculation logic takes the secondary key, looks for the additional discount, and applies the adjustment to list prices. There will be the `Volume Discount` element to show the discount per volume value. If there is no volume breakdown defined for an SKU, the PL/LPG shows 1 row of the SKU with `Secondary Key` value set to 1, and the `Volume Discount` 0%.

The volume breakdown configurations are defined in the Product dimension, <dependency>VolumeBreakdown PP table. In order to use the feature properly, the table has to be configurable on the level of the lookup key and has one PP per dependency level. Example:

Price Parameter Values : GlobalVolumeBreakdown [3]

Business Unit	Product Gr...	Product Cl...	Volume 1	Discount 1	Volume 2	Discount 2	Volume 3	Discount 3
Food	Meatball	A	0	10.00 %	20	20.00 %	50	50.00 %
Others	*	*	10	10.00 %	20	20.00 %		

Exceptions on the SKU level are defined in the VolumeBreakdownExceptions PP. With this, the configuration in the corresponding <dependency>VolumeBreakdown will be ignored. Example:

Price Parameter Values : VolumeBreakdownExceptions [4]

SKU	Dependenc...	Volume #01	Discount #01	Volume #02	Discount #02	Volume #03	Discount #03	Volume #04
MB-0001	France	20	0.50 %	60	1.00 %			
MB-0001	Germany	20	0.50 %	60	1.00 %	100000	20.85 %	
MB-0002	Germany	5	5.00 %	10	10.00 %			

⚠ For <dependency>VolumeBreakdown and VolumeBreakdownExceptions PP tables:
If there is a space between the volume-discount pairs, then discount values cannot be parsed.
Volume discount does not have an impact on Manual Override Price.

Margin Break-even and Revenue Break-even

When making pricing changes in the price list, there will be two added fields:

- Break-even volume for revenue
- Break-even volume for margin

This requires three fields:

- New Margin (%)
- Previous Margin (%)
- Previously approved price

The calculation formula:

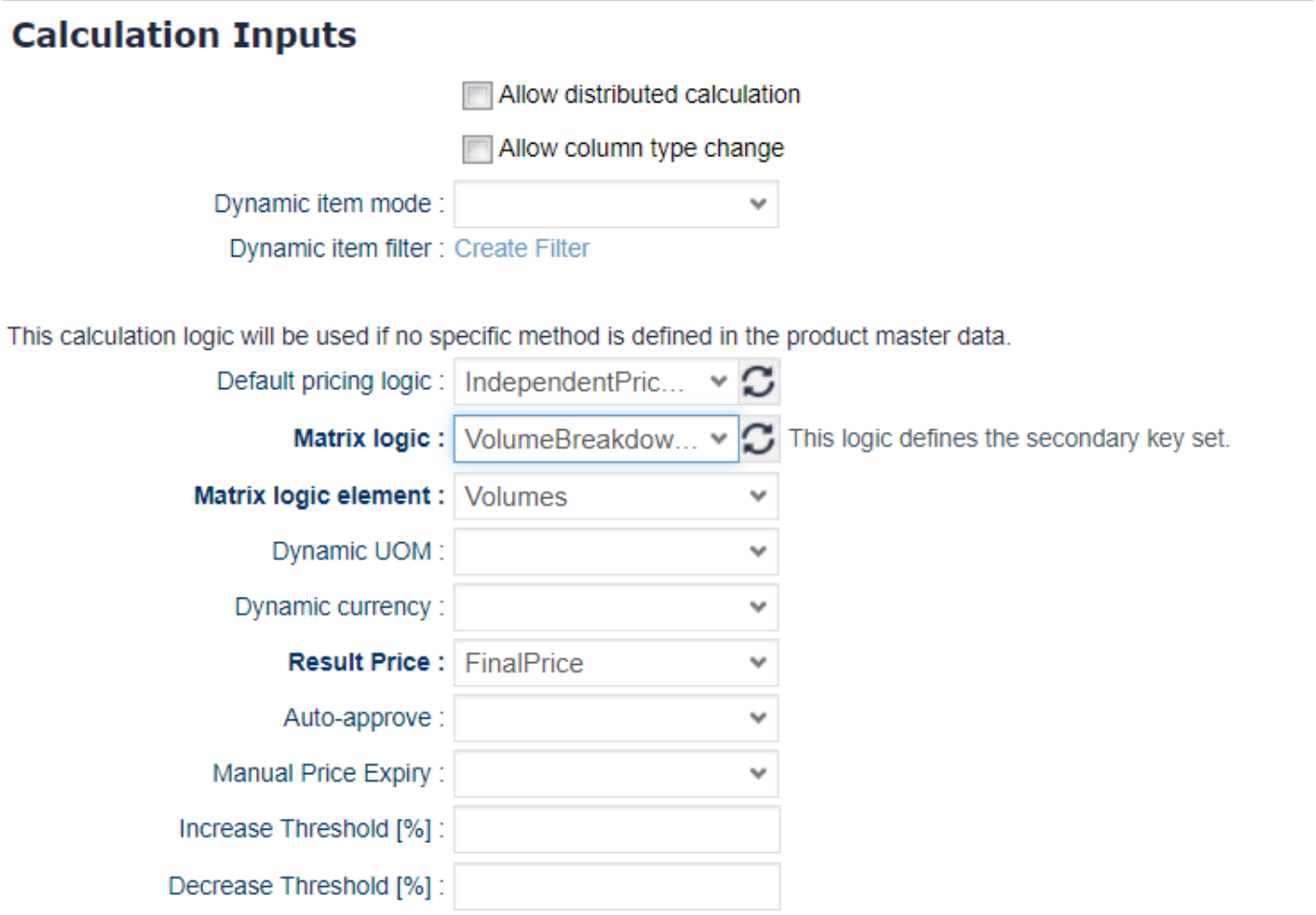
- Break-even volume for revenue %: (New Price minus Previous Price) / Previous Price) * -100%
- Break-even volume for margin %: (((Old Price - Cost) * Volume) / (New Price - Cost)) / Volume) - 1

Example:

Old price	New Price	Cost	Volume	Old Revenue	Old Margin	Breakeven volume increase %	Old margin %	New Margin %	Breakeven margin %	New Margin breakeven volume	New Revenue breakeven volume	New margin	New Price
100	110	70	100	10000	3000	-9,0909091	0,3	0,36363636	-0,21212	75	90,90909	3000	10000
100	90	70	100	10000	3000	11,111111	0,3	0,22222222	0,35	150	111,1111	3000	10000

Usage

To use this feature, set proper data and create a matrix PL/LPG and use the VolumeBreakdownMatrixLogic as the Matrix logic:



Pricing Configuration

- Hierarchical Lookups
- Product Segmentation per Feature

Hierarchical Lookups

- Configuration Of Bootstrapping
- Hierarchical Config Lookup
- Lookup List

Configuration Of Bootstrapping

Configuration tables for Hierarchical Lookups are created dynamically by bootstrapping. Bootstrapping receives 4 inputs:

- There is 1-6 hierarchical attributes configured for any lookup in PriceSettingDimensions PP. These are the keys of newly created Price Parameters. Hierarchical lookups are configs not intended to be configured for a product, but for a group of products. However, the decision how to split products into groups is up to the user. Splitting products per SKU into 1-element group will work just fine.

- There are 13 features to be configured and one general fallback. It decides which configs (from above) are put in which Price Parameters as keys.
- Most of the Hierarchical Lookups have one table per Dependency Level + 1 (universal fallback). Dependency Configuration should be prepared before bootstrapping: DependencyConfiguration PP
- Bootstrapping expects to find on the partition the below listed Price Parameters. These PPs will be removed during the run. Since the amount of PPs might be very big (for a lot of Dependency Levels), each Price Parameter is placed where related PPs should be generated.
  - AdditionalDiscountTempHook
  - AdjustedPriceCorridorTempHook
  - BaseStrategySelectionTempHook
  - CostPlusTempHook
  - CostSelectionTempHook
  - DependencyLevelAdjustmentTempHook
  - DiscountTempHook
  - ListPriceCorridorTempHook
  - MinMarginTempHook
  - PriceIncreaseTempHook
  - RelevantCompetitionDataTempHook
  - StrategySelectionTempHook
  - VolumeBreakdownTempHook

All of these are handled by PlatformManager in the standard deployment scenario. For edits after deployment, follow Change Product Segmentation.

Hierarchical Config Lookup

After generating hierarchical tables, these should be filled with data:

Attributes

Attributes of hierarchical tables are described on their pages.

Keys

Each hierarchical table has 1-6 keys. Each key is a product attribute. If there was no name to the product column, "ProductColumn-attributeXX" name is used. The user should describe groups of products, with the ability of using "*" fallback. Order of entries is irrelevant - the most detailed config is chosen.

If no entry has been chosen, the user might create a general fallback with only an asterisk ("*").

If no "asterisk fallback" is used, hierarchical fallback will be utilized. It means there is no need to create configs for very detailed dependency levels on which we do not perform segmentation: https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/2962817818/Dependent+Price+Lists+and+Data+Fallbacks#Lookup-Keys-Config-Fallbacks.

Example:

Business Unit	ProductColumn-attribute10	Product Class
Food	*	*
Food	Meatball	C
Others	*	*
*	*	*
Beverages	*	*
Food	Meatball	B
Others	Toppings	C
Food	Apple	A
Food	Meatball	tempA
Food	Sausage	A
Food	Sausage	B
Test PLCM	*	*
Food	Apple	B
Food	Meatball	D
Food	Meatball	A
Beverages	Non-Alcoholic	C

Lookup List

The "TempHooks" PPs listed above each correspond to a single lookup.

- AdditionalDiscount
- AdjustedPriceCorridor
- BaseStrategySelection
- CostPlus
- CostSelection
- DependencyLevelAdjustment
- Discount
- ListPriceCorridor
- MarginAlertsForPriceLists
- MinMargin
- PriceIncrease
- RelevantCompetitionData
- StrategySelection
- VolumeBreakdown

Product Segmentation per Feature

⚠ Product Segmentation controls how some Price Parameters will be generated during the deployment. It means that changes to this configuration require additional administrative actions described in Change Product Segmentation.

We support a "general" lookup key that is applicable for every lookup where you do not define a specific one. You can define a specific set of lookup keys for different tables and features in Price Setting Package during deployment.

There is the option to define other specific lookup keys per feature. It is possible to define the lookup keys for following features:

Feature	Description
Fallback	Used whenever there is no specific lookup key per feature.
StrategySelection	Selection of the strategies and importance.
MinMargin	Minimum margin used for warnings and margin checks.
DependencyLevelAdjustment	Markup factor between the Parent Price List and the Dependent one.
VolumeBreakdown	Quantities and volume discounts for price lists.
CostPlus	Factor for the Cost Plus Price strategy that is applied to the cost base to get a price.
PriceIncrease	Factor for (periodic) price increase that is applied to the old price to get a new increased price.
AdditionalDiscount	Predicted additional discount. This is used in Target Price Strategy to anticipate influence of additional On-Invoice and Off-Invoice Price conditions.
BaseStrategySelection	Selection of "Base Strategies". Base Strategies are usually defined on more generic level to have some basic pricing rules across the complete product portfolio.
AdjustedPriceCorridor	Corridor used for price harmonization checks. It is used to check how strong the consistency in the general pricing rules is. The smaller this KPI is, the more the Dependent Pricing is aligned with the overall rules set. When strictly following the Parent Level Prices and the defined markup factor, it is zero.
ListPriceCorridor	Corridor used for price harmonization checks. This is used to check the harmonization of the prices themselves between Parent Level Price in Dependent Level Price. When AdjustedPriceCorridor is zero, this will exactly mirror the markup factor between the two.
RelevantCompetitionData	Parameter to define the relevant competitors. You can decide if your competition based strategies will use all competition data or only the set of relevant competition data.
CostSelection	When you have more than one cost type (e.g. Cost with freight, Average Warehouse Cost, ...) you can decide per product segment which of them is used (for calculation of margin, for Cost Plus pricing strategy, ...).
Discount	Discount you have in your discount structure. It is used in the Gross/Net mode to calculate the net price based on the calculated List Price.

Each of the lookups will have generated multiple tables, based on dependency hierarchy. For details see Dependent Price Lists and Data Fallbacks.

To learn more about Hierarchical Lookups, visit Hierarchical Lookups.

Before deployment it is important to consider what features will be used. For every feature you have to think about the granularity which will be required for data later. This is generally a business decision.

Price Setting Modules

Modularization is one of main concepts in Price Setting Package. It means that the package is split into a single required Core Module and multiple independent feature modules. This separation allows the package to stay fairly simple for small installations. while also allowing for more complex feature rich configurations.

Since modules are mostly independent, most of them have warnings and errors on the module level. In case something goes wrong in the module, the rest of them will still work.

Modules can be configured through our Configuration Wizard or by enabling them manually and configuring them according to individual module's configuration page.

Available modules:

Module Name	Description	Configuration key name
Core Elements	Basic Module of Price Setting Accelerator. In the module you can do the basic configuration of the package.	
Transaction	Displays transaction and forecast data about products.	PSP_TRANSACTION_MODULE
Stock	Provides stock data and the StockCoverDays calculation. This module depends on the Transaction module.	PSP_STOCK_MODULE
Net Price	Allows you to calculate a net price (with a proper discount taken into consideration). This is usually used in B2B(2C) Business.	PSP_NET_PRICE_MODULE
Overrides	Handles exceptions in pricing. It allows you to manually override product prices in the Price List / Price Grid or store exceptions per SKU.	PSP_OVERRIDES_MODULE
Price Checks	Checks if the user margin is within a suitable range and if not, it issues alerts. Also, for dependent price lists, it checks if the difference between a dependent and parent price is within a suitable range.	PSP_PRICE_CHECKS_MODULE
Price Flexibility	Provides integration with Price Flexibility Package. It adds a new element to the independent price grid which describes why a product has been automatically added to a price grid.	PSP_PRICE_FLEXIBILITY_MODULE

Product Competition	Gathers and displays product competition data. This can be used for any competition based strategy.	PSP_PRODUCT_COMPETITION_MODULE
Strategy Conditions	Performs additional checks if prices meet certain conditions. Strategies can be skipped or used as fallback to ensure proper pricing rules.	PSP_STRATEGY_CONDITION_MODULE
Rounding Rules	Rounds prices to user friendly values.	PSP_ROUNDING_RULES_MODULE
Advanced Cost	Calculates additional cost types. These will be used for pricing strategies and margin calculations.	PSP_ADVANCED_COST

PSP Core Elements

Price Setting Package Core Elements consist of:

- PSP Cost Element
- PSP Actual Price Element

PSP Cost Element

Lookup

1. Search for records of the current dependency level in the selected source table. If none is found, use the dependency fallback mechanism.
   - Details about the dependency fallback mechanism can be found at Dependent Price Lists and Data Fallbacks.
   - The selected source table depends on the dependency mapping configuration.
2. Filter records with current valid time configuration.

Calculation with Cost

1. Convert the cost currency to the current currency.
   - Details about currency conversion can be found at ExchangeRates PP
2. Perform the calculation based on the selected price strategies.

Configuration

1. Set the data source configuration in the PriceSettingConfig PP table. Details can be found at Cost Config (PriceSettingConfig).
2. Set the dependency mapping in the DependencyMapping PP table with the key `Cost`. Details can be found at DependencyMappingConfig PP.

PSP Actual Price Element

Lookup

Product Extension Source Type

1. Search for records of the current dependency level in the selected source table. If none is found, use the dependency fallback mechanism.
   - Details about the dependency fallback mechanism can be found at Dependent Price Lists and Data Fallbacks.
   - The selected source table depends on the dependency mapping configuration. Details can be found at DependencyMappingConfig PP.
2. Filter records with current valid time configuration.

Price List Source Type

1. Search for the latest approved price list which contains the record for the current SKU. The price list must also have the same dependency level name and calculation logic name as the current PL /PG.
2. Get the final price of the found item.

Live Price Grid Source Type

1. Search in the current PG for the latest approved record of the current SKU. If there is no approved price yet, it will take the price from previous calculation.
2. Get the final list price of the found item. If the final list price is not available, get the final price.

Calculation with Actual Price

1. Convert the price currency to the current currency.
   Details about currency conversion can be found at ExchangeRates PP.
2. Perform the calculation based on the selected price strategies.

Configuration

1. Set the data source configuration in the PriceSettingConfig PP table. Details can be found at Actual Price Lookup.
2. Set the dependency mapping in the DependencyMapping PP table with the key `Actual Price`.
   Details can be found at DependencyMappingConfig PP.

PSP Override Module

The Override module allows you to create custom price behavior that does not follow the default rules.

Visible Elements

Visible elements of the Override module are `Override Price`, `Override Reason`, `Price Selector`, and `Exceptions`. These elements override the calculated prices.

Element name	Label	Parent PL /PG	Dependent PL/PG	Description
PriceSelector	Price Selector	Yes	Yes	Dropdown list of calculated prices to choose from.
ManualPrice	Override Price	Yes	Yes	To enter a price manually.
ManualPriceReason	Override Reason	Yes	Yes	To enter a comment manually.
Exceptions	Exceptions	Yes	Yes	Information about the override values (if any).

Override Levels

> ⚠ Changing this configuration adds or hides some visible fields and because of the way Pricefx treats such changes, all used PLs and PGs should be recreated from scratch. Otherwise there will be "zombie columns" which will make the impression that exceptions do not work correctly.

- The current line (LineLevel) - Can only set the override values on the current SKU line in PL/PG. The ExceptionTable values are excluded.
- Configuration tables (ExceptionTable) - Can only set the override values through the configuration tables. The LineLevel values are excluded.
- Both (Yes) - Uses both Current line and Exception tables methods.
- None (No) - Override is not allowed.

More details about override levels can be found at Exceptions and Manual Override Allowance Config (PriceSettingConfig).

Override Mechanism

Depending on the Override level configuration, the user can override a product price or a product strategy in different ways. The `Price Selector` element contains the calculated prices based on specified strategies and exceptional values.

- To set up a strategy, see StrategyDefinition PP.
- For a list of built-in calculation engines, see Calculation Engines.
- To specify the strategies used to calculate the price for an SKU, see Exception Lookup and Strategy Selection Lookup.

Type	Usage	Current line (Manual override)	Configuration tables (Exception table)	Both	None
Price	To override an SKU price with a specific price	Type the price in the `Manual Price` field	Set the price in a configuration table	Can do both Current line and Exception tables methods	Override is not allowed
Strategy	To use a specific strategy in the calculated prices as the selected strategy	Select a strategy from the `Price Selector` dropdown list	Set the strategy in a configuration table	Can do both Current line and Exception tables methods	Override is not allowed

Override Order (Highest to Lowest)

Order	Name	Price to calculation	Price Decision	Necessary Action
1	Manual Price Override	Price from the Manual Price field	Default comment is inserted if none given. It can be manually overridden.	Type a price in the Manual Price field
2	Manual Strategy Override	Price from PriceSelector	Default exception message with the	Choose a strategy from the PriceSelector drop-down

			name of the price strategy chosen in the exception.	
3	Price Exception	Price from an exception table	Default exception table message.	Set up an exception for the product in table
4	Strategy Exception	Price from the price strategy chosen in the exception	Default exception message with the name of the price strategy chosen in the exception.	Set up an exception for the product in table
5	Normal	Price from the first strategy in the calculated prices (first/bold row in the Prices popup)	Name of the price strategy.	N/A

Some levels of this hierarchy can be skipped by changing the Manual Override Allowance configuration. For example, when setting the Parent Manual Override for a ˇprice to "ExceptionTable", it will disable the Manual Price Override from this hierarchy.

Exception Data Sources

Price Exception

Lookup

1. Search for records of the current dependency level in the selected source table. If none is found, use the dependency fallback mechanism.
   - Details about the dependency fallback mechanism can be found at Dependent Price Lists and Data Fallbacks.
   - The selected source table depends on the dependency mapping configuration.
2. Filter records with current valid time configuration.

Calculation with Price Exception

1. Convert the exception value currency to the current currency.
   - Details about currency conversion can be found at ExchangeRates PP.
2. The exception value is in the `Price Selector` element for selection. If this is the default final price or not depends on the level configuration and override orders.

Strategy Exception

Lookup

1. Search for records of the current dependency level in the selected source table. If none is found, use the dependency fallback mechanism.
   - Details about the dependency fallback mechanism can be found at Dependent Price Lists and Data Fallbacks.
   - The selected source table depends on the dependency mapping configuration.
2. Filter records with current valid time configuration.

Calculation with Strategy Exception

The exception value is in the `Price Selector` element for selection. If this is the default final price or not depends on the level configuration and override orders.

Configuration

Set Module Status

Set the module status in the PriceSettingModules PP table whose the module name is `PSP_OVERRIDES_MODULE`.

Set Override Levels

1. Set the product price override level for parent pricing in the PriceSettingConfig PP table with the keys `Parent Manual Override Allowance | Price`.
2. Set the product strategy override level for parent pricing in the PriceSettingConfig PP table with the keys `Parent Manual Override Allowance | Strategy`.
3. Set the product price override level for dependent pricing in the PriceSettingConfig PP table with the keys `Dependent Manual Override Allowance | Price`.
4. Set the product strategy override level for dependent pricing in the PriceSettingConfig PP table with the keys `Dependent Manual Override Allowance | Strategy`.

The override level options:

- Yes
- No
- LineLevel
- ExceptionTable

Set Price Exception Data Source

1. Set the data source configuration in the PriceSettingConfig PP table whose key is `Price Exception`. Details can be found at Exception Lookup.
2. Set the dependency mapping in the DependencyMapping PP table whose key is `Price Exception`. Details can be found at DependencyMappingConfig PP.

Set Strategy Exception Data Source

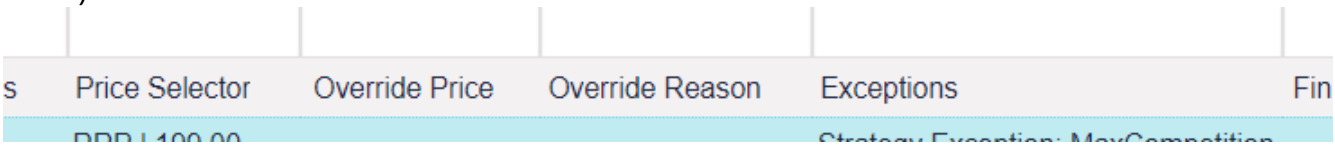
1. Set the data source configuration in the PriceSettingConfig PP table whose key is `Strategy Exception`. Details can be found at Exception Lookup.
2. Set the dependency mapping in the DependencyMapping PP table whose key is `Strategy Exception`. Details can be found at DependencyMappingConfig PP.

PSP Rounding Rules Module

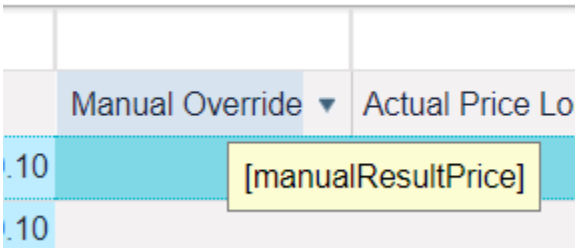
The Rounding module allows users to round prices to business-friendly values. Rounding is done by using a defined set of rules. This set can be expanded if needed.

⚠ **Note**:

- Values from PSP Exception Module will not be rounded (manually overridden prices and Exception Prices).



- Default Manual Override of Pricefx will be rounded.



- Only **List Price** is rounded for each calculated strategy. So when using the Net Calculation Level, it only rounds the Gross Price.

Mechanism

This module rounds prices to business-friendly targets. Manually overridden prices and Exception Prices will not be rounded.

Targets

- To49Cents: XXX.YYY => XXX.49
- To50Cents: XXX.YYY => XXX.50
- To95Cents: XXX.YYY => XXX.95
- To99Cents: XXX.YYY => XXX.99
- ToWhole: XXX.YYY => XXX
- To5Whole: XXX.YYY => XX5
- To49Whole: XXX.YYY => X49
- To99Whole: XXX.YYY => X99
- RawPrice*: XXX.YYY => XXX.YY
- NoRounding: do not round

Modes

- UP - Round away from zero.
- DOWN - Round towards zero.
- HALF_UP - Round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up.
- HALF_DOWN - Round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down.

More information about rounding mode definitions and examples can be found at Rounding Modes.

Configure Rules

In the rounding rules configuration table, specify these fields:

- From (must be PP key1) - The price to be rounded should be greater than or equal to this field value.
- To (must be PP key2) - The price to be rounded should be less than this field value.
- Dependency Mapping Field - The value to be used for dependency mapping. Not required if using the table mapping mode.
- Rounding Rule - The target to be applied.
- Rounding Mode - The mode to be applied.
- Valid From - Valid start date. Optional.
- Valid To - Valid end date. Optional.

Examples:

From	To	Country	Rounding Rule	Rounding Mode
5.00	10.00	Global	To49Cents	HALF_UP
10.00	20.00	Global	To50Cents	UP
20.00	30.00	Global	To95Cents	HALF_DOWN
30.00	40.00	Global	To99Cents	UP
40.00	49.00	Global	ToWhole	DOWN
50.00	55.00	Global	To99Whole	DOWN

Rule Data Source

Lookup

1. Search for records of the current dependency level in the selected source table. If none is found, use the dependency fallback mechanism.
   - Details about the dependency fallback mechanism can be found at https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/2962817818/Dependent+Price+Lists+and+Data+Fallbacks#Data-Fallbacks.
   - The selected source table depends on the dependency mapping configuration.
2. Filter records with current valid time configuration.

Configuration

Set Module Status

Set the status in the PriceSettingModules PP table which the module name is `PSP_ROUNDING_RULES_MODULE`

Set Rule Data Source

1. Set the data source configuration in the PriceSettingConfig PP table with the key `Rounding Rules`. Details can be found at Rounding Rules Lookup.
2. Set the dependency mapping in the DependencyMapping PP table with the key `Rounding`. Details can be found at DependencyMappingConfig PP.

PSP Net Price Module

This module calculates net prices (with proper discounts taken into consideration). This is usually used in B2B or B2C businesses.

Module Related Elements

Technical Name	Label	Available in Parent PL/PG	Available in Dependent PL/PG	UI Visible	Output Type	Description
NetPriceLevel		Yes	Yes	No	Int (bool)	Indicates if the net price will be calculated.
NetPrice	Net Price	Yes	Yes	When pricing	BigDecimal	List price with a discount applied.

Discount	Discount	Yes	Yes	mode is Gross/Net and the module is turned on	BigDecimal	Discount (%) to be applied when transitioning from a gross to net price.
FinalListPrice	Final List Price	Yes	Yes		BigDecimal	See the Mechanism section.
FinalPrice	Final Price	Yes	Yes	Yes	BigDecimal	

Net Price Module and Pricing Mode

Pricing mode	Module status	Visible	Final list price	Net price	Final price
Gross	On	No	Not available	Not available	Strategy calculated price
Gross	Off	No	Not available	Not available	Strategy calculated price
Gross/Net	On	Yes	Strategy calculated price	`Final list price` with `Discount` applied	`Net price`
Gross/Net	Off	No	Not available	Not available	Strategy calculated price

To set the pricing mode, see PriceSettingLevel PP and Adjustments after Changing the PP PriceSettingLevel.

Discount Data Sources

- Discount data tables – Used for the regular PL/PG calculation.
  Table name convention: `<<dependency>>Discount`.
  Examples: Discount, AsiaDiscount, GlobalDiscount,...
- Additional d'iscount price parameter tables – Used for the Net engine.
  Table name convention: `<<dependency>>AdditionalDiscount`.
  Examples: AdditionalDiscount, AsiaAdditionalDiscount, GlobalAdditionalDiscount,...
- Lookup – Search for the table which matches the current dependency level. If none is found, use the dependency fallback mechanism.
  Details about the dependency fallback mechanism can be found at Dependent Price Lists and Data Fallbacks.

⚠ When using the dependency fallback mechanism for <<dependency>>Discount /<<dependency>>AdditionalDiscount, if a dependency level in the tree has a complete state `Yes` but the value cannot be found, it will get the value from the root table, which is the Discount table / AdditionalDiscount table.

Net Price Module and Net Engine

The Net engine takes a target price and performs calculations to produce the net price. The data source for the `target price` is defined in the corresponding additional engine configuration table. For details see Net Engine - Additional Engine Configuration.

When using the strategy running on the Net engine, the following mechanism is applied:

Net Price module	Discount (%)	Additional discount (%)	Strategy calculated price	Net price	Final price
On	Some value	Some value	`target price / (1-Additional Discount) / (1-Discount)`	`calculated price * discount`	`net price`
On	Null	Null	`target price`	Null	Null
On	Some value	Null	`target price / (1-Discount)`	`calculated price * discount`	`net price`
On	Null	Some value	`target price / (1-Additional Discount)`	Null	Null
Off	N/A	Some value	`target price / (1-Additional Discount)`	N/A	`calculated price`
Off	N/A	Null	`target price`	N/A	`calculated price`

Configuration

To turn the module on or off, update the module status in the PriceSettingModules PP table whose name is `PSP_NET_PRICE_MODULE`.

PSP Transaction Module

This module displays transaction and forecast data about products.

Module Visible Elements

Visible elements of the Transaction module are `Sales Volume YTD`, `Turnover YTD`, `Last Year Sales Volume`, `Last Year Turnover`, `Last Period Volume`, `Last Period Turnover`, `Sales Volume Forecast`, and `Turnover Forecast`. These elements display transaction and forecast data of the product, which are sales data for the last year and forecast data for the next year.

Type	Technical Name	Label	Independent PL/PG	Dependent PL/PG	Description
Historical	SalesVolumeYTD	Sales Volume YTD	Yes	Yes	The sum of sales volume from the beginning of the current year to the calculation date.
	TurnoverYTD	Turnover YTD	Yes	Yes	The sum of turnover from the beginning of the current year to the calculation date.
	LastYearSalesVolume	Last Year Sales Volume	Yes	Yes	The sum of sales volume from the whole last year.
	LastYearTurnover	Last Year Turnover	Yes	Yes	The sum of turnover from the whole last year.
	LastPeriodVolume	Last Period Volume	Yes	Yes	The sum of sales volume in a specified time range in the past.

	LastPeriodTurnover	Last Period Turnover	Yes	Yes	The sum of turnover in a specified time range in the past.
Forecast	SalesVolumeForecast	Sales Volume Forecast	Yes	Yes	The sales volume forecast.
	TurnoverForecast	Turnover Forecast	Yes	Yes	The turnover forecast.

Last Period Calculation

The last period configuration allows flexible data lookup. The time units used in this calculation (days, weeks, months, and years) are not days counting from the beginning. In other words, when specifying the last period time as `1 week`, it does not mean taking 7 days from the calculation day backward. The calculation takes only periods that are finished and a week starts on Sunday, ends on Saturday.

For example, the calculation date is 22 September 2020, and the last period configuration "1 week". The result will be the data from 13 September to 19 September.

Forecast Types

Last year	Linear	Lookup
The sum of sales volume /turnover from the whole last year	ytdData = sum of sales volume/turnover from the beginning of the current year to the calculation date ytdDaysCount = number of days from the beginning of the current year to the calculation date currentYearDaysCount = number of days of the calculation year result = ytdData / ytdDaysCount * currentYearDaysCount	The sum of sales volume /turnover from the beginning of the current year to the calculation date. The data is obtained from a defined data source.

To configure the forecast type, see Forecast Config (PriceSettingConfig).

Currency Exchange

Transaction data may point to Product Extensions, Datamart or Data Source. In the Price Setting Package, the PL/PG logic will ignore the "currency" field in Datamart source tables as the exchange rate has been applied during the data transition from Data Source to Datamart. The exchange rate is only applied when the Datamart's currency and the calculation's currency are different.

When the transaction data source points to Data Source, the "currency" field of the pointed Data Source table will be utilized. For details, see How to Set up Currencies.

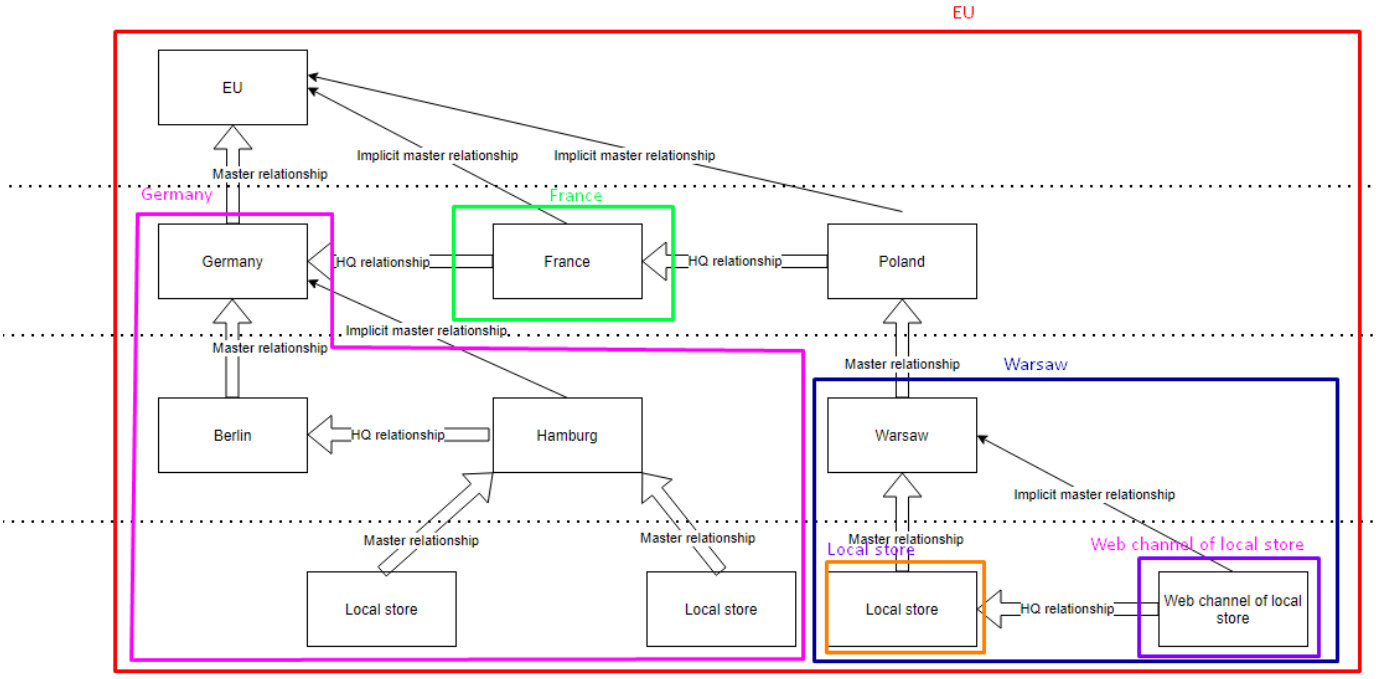
When the transaction data source points to Product Extensions, the currency is converted using the ExchangeRate PP. For details, see ExchangeRates PP.

Lookup

Transaction data lookups return aggregated data for the current and all children dependency levels and there is no fallback like other lookups. Transactions are records of anything being sold. If something has been sold in a city, it means that these transactions should also appear as being sold in the country.

If there is no dependency defined, e.g. a small company working only in a country, without changing their prices depending on the regions, it will grab all transactions.

Example:



It collects everything from Germany and levels defined as children, and children's children, etc. In this example, Germany data = Germany + Berlin + Hamburg + Local store + Local store. It does not include nodes at the same level, which are France and Poland in this case.

Configuration

Set Module Status

Set the module status in the PriceSettingModules PP table whose module name is `PSP_TRANSACTION_MODULE`

Set Transaction Data Source

- For transaction data source, set it in the PriceSettingConfig PP at the key `Transaction Source`. For details, see Transaction Lookup.
- For transaction dependency mapping, set it in the DependencyMapping PP table at the key `Transaction`. Details can be found at DependencyMappingConfig PP.

Configure Forecast Calculation

- For forecast setting, set it in the PriceSettingConfig PP at the key `Forecast`. For details, see Forecast Config (PriceSettingConfig).
- For transaction dependency mapping, set it in the DependencyMapping PP table at the key `Forecast`. Details can be found at DependencyMappingConfig PP.

Configure Last Period Calculation

To configure the last period calculation, set it in the PriceSettingConfig PP at the key `Last Period Transaction`. For details, see Last Period Config (PriceSettingConfig).

Warning

Technically, if there are too many rows (1 million by default) for a given batch of products (200 by default), the batch will be split in two, a warning will be raised and the SQL query will be executed again. If there are too many rows for only 1 SKU, then the Pricefx restriction has been met, an error will be raised and no transaction data will be read.

In Product Extensions, however, we do not expect to have tens of thousands of rows. We expect data to be pre-aggregated. Having a lot of rows in PX is possible, but counterintuitive.

PSP Stock Module

Stock module provides stock data and the StockCoverDays calculation. This module depends on the Transaction module.

Lookup

1. Search for records of the current dependency level in the selected source table. If none is found, use the dependency fallback mechanism.
   - Details about the dependency fallback mechanism can be found at Dependent Price Lists and Data Fallbacks.
   - The selected source table depends on the dependency mapping configuration.
2. Filter records with current valid time configuration.

Calculation with Stock

To calculate Days Of Cover, we have to take Sales Volume Forecast from the transaction data and divide it by 365. Then we get average selling volume per day.

Days Of Cover = Stock / Avg selling volume per day

The result is rounded without decimal places, half up.

If the Sales Volume Forecast value is invalid (null, 0,...), there will be no Days Of Cover forecast as well.

Configuration

1. Set the data source configuration in the PriceSettingConfig PP table. Details can be found at Stock Lookup.
2. Set the dependency mapping in the DependencyMapping PP table with the key `Stock`. Details can be found at DependencyMappingConfig PP.

PSP Price Checks Module

The Price Checks module verifies if a product margin is within a suitable range. For dependent price lists, it also checks the delta between the parent price and the dependent price in a dependent PL/PG.

Module Elements

Technical name	Label	Available in Parent PG/PL	Available in Dependent PG/PL	Description
MinimumMargin	Minimum Margin	Yes	Yes	The minimum margin specified for the product.
ListPriceCorridor	List Price Corridor	No	Yes	Delta between the parent price (*) and final list price.

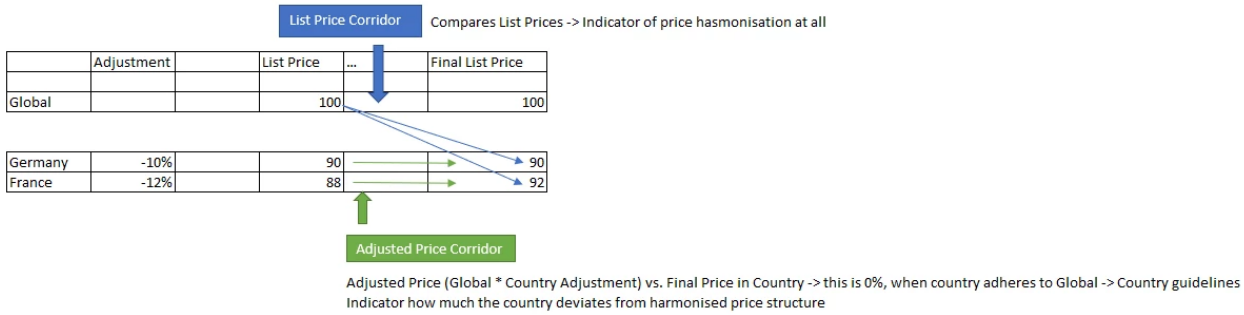
				On parent price, refer to Element Data Sources > List Price Corridor section below.
AdjustedPriceCorridor	Adjusted Price Corridor	No	Yes	Delta between the adjusted parent price and final list price. On parent price, refer to Element Data Sources > Adjusted Price Corridor section below.
MinimumMarginPrice	Minimum Margin Price	Yes	Yes	A price calculated based on `MinimumMargin` (%), `Cost` and `Discount` (%).

This module contains also Margin Alert Lookup which means that you can configure the MarginAlertsForPriceLists lookup and there is an additional "MarginFlag" element.

Element Data

Values for price checks are configured in respective price parameter tables or calculated results.

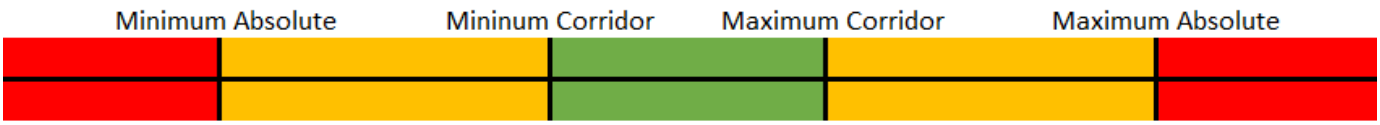
- **Minimum Margin – MinMargin PP table**
  From table with naming convention: `<<dependency>>MinMargin`.
  Examples: MinMargin, AsiaMinMargin, GlobalMinMargin,...
- **List Price Corridor (Dependent PG/PL only)**
  The value is taken from the final list price of the referred parent PG/PL.
  To set the referred parent PG/PL for a dependency level, see DependencyConfiguration PP.
- **Adjusted Price Corridor (Dependent PG/PL only)**
  The value is taken from the final list price of the referred parent PG/PL with the dependent adjustment applied.



  To set the dependent adjustment, fill the data in the corresponding `<<dependency>>DependencyLevelAdjustment` Price Parameter table.
- **Minimum Margin Price** – The value is calculated with the formula:
  - At Gross pricing mode: result = Cost / (1 - MinimumMargin)
  - At Gross/Net pricing mode: result = Cost / (1 - Discount - MinimumMargin + MinimumMargin * Discount)

Checking Mechanism

The List Price Corridor and Adjusted Price Corridor elements will be colored based on the corridor ranges which they fall into.



- When the element value is inside the Corridor range, it will be green.
  - Minimum Corridor <= value <= Maximum Corridor
- When the element value is inside the Absolute range, it will be yellow.
  - Minimum Absolute <= value < Minimum Corridor
  - Maximum Corridor < value <= Maximum Absolute
- When the element value is outside the Absolute range, it will be red.
  - value < Minimum Absolute
  - value > Maximum Absolute
- When the element value is zero, it will be blue.

Configuration

Set Module Status

Set the module status in the PriceSettingModules PP table whose module names is `PSP_PRICE_CHECKS_MODULE`

Set Minimum Margin

To specify the minimum margin for product segments, do it in the corresponding `<<dependency>>MinMargin` PP table.
Examples: MinMargin, AsiaMinMargin, GlobalMinMargin,...

Set List Price Corridor

To validate the delta between the parent price and the final list price of a product, the corresponding boundaries are taken from the ListPriceCorridor PP table.

Set Adjusted Price Corridor

To validate the delta between the adjusted parent price and the final list price of a product, the corresponding boundaries are taken from the AdjustedPriceCorridor PP table.

PSP Strategy Conditions Module

The Strategy Conditions module lets users define special conditions based on which some prices will be ignored or taken with lower priority. Conditions are applied at the very end of the calculation, after exceptions. This feature does not, however, differentiate between parent and dependent prices in a dependent price list. Parent prices are fetched and paired with dependent ones after applying conditions.

To set a condition, there are these requirements: condition order, condition expression, and the rule to be applied.

Condition Expression Syntax

In general, a condition expression has the form of `left-hand side operand operator right-hand side operand`. For example, Cost+.margin<MINIMUM_MARGIN_PRICE. A suggestion is to think of the left-hand side strategy as the main strategy for the given condition.

Left-hand Side Operand

The left-hand side of the expression must always be a property of a given strategy. The supported properties are:

- Gross price of the strategy: `<some strategy name>.price`.
  - Example: Cost+.price, RRP.price, etc.
- The margin of the strategy: `<some strategy name>.margin`.
  - Example: Cost+.margin, RRP.margin, etc.

The strategy can be any calculated strategy or exception.

⚠ Do **not** use parentheses with strategy names like this: (Cost+.price), (RRP).price. The correct format is Cost+.price or RRP.price.

Operator

Supported operators are: "<" (less than), ">" (greater than), "=" (equals).

Right-hand Side Operand

The right-hand side of an expression may be either property of a strategy or PriceCalculator's parameter.

- Gross price of the strategy: `<some strategy name>.price`.
  - Example: Cost+.price, RRP.price, etc.
- The margin of the strategy: `<some strategy name>.margin`.
  - Example: Cost+.margin, RRP.margin, etc.
- PriceCalculator's parameter: `parameter name`
  - Example: MINIMUM_MARGIN_PRICE, DISCOUNT, etc.

In addition, the right-hand side may be modified by a certain multiplier. The syntax is to wrap the right-hand side operand into parenthesis "(...)", add an asterisk character and a string parsable to BigDecimal. Example of a condition expression using multiplier:

- Cost+.price<(RRP.price * 2)
- Cost+.price<(MINIMUM_MARGIN_PRICE * 2)

Also, custom values from additionalParameters and additionalOptionalParameters are accepted.

⚠ The strategy name needs to be written explicitly. This is the case when using Dependent PL/PG and "Cost+ (Parent Type Strategy)" and "Cost+" are two different strategies.

Rules

Name	Description	Syntax
Skip	Remove the strategy on the left-hand side from the strategy list.	(skip)
Fallback		(fallback)

	Move the left-hand side strategy to the last position of the strategy list.	
Move behind	Move a strategy after another strategy.	$strategyName1<$strategyName2 Example: "Cost+<RRP", which mean moving Cost+ strategy after RRP strategy.

Wildcards

To manage a large number of strategies in one go, there is the wildcard "{any}". The condition with wildcard will be applied to every strategy and exception in the strategy list. At the place of a wildcard, strategy name will be inserted.

Example: "{any}.margin<MINIMUM_MARGIN"

Configuration

Set Module Status

Set the module status in the PriceSettingModules PP table where the module names is `PSP_STRATEGY_CONDITION_MODULE`.

Set Condition

To set a strategy condition, set it in the StrategyConditions PP. To learn more about it, see StrategyConditions PP.

Notes

Some explanation for condition

"{any}.price < (Cost+.price *5) " : I don´t want to have any price that is smaller (Cost+.price * 10)... So basically Cost+ price is smaller than (Cost+.price * 10)

PSP Advanced Cost Module

This module provides additional cost strategies which are used for pricing strategies and margin calculations. Each cost strategy has its own dependency mapping configuration and data source configuration (including optional valid dates or currency).

Mechanism

Similarly to pricing strategies, there can be many cost strategies. In a product segment, several cost strategies can be selected. The cost strategies will be calculated and their values are presented in the `Advanced Costs` element of the PL/PG. However, only the first valid value (top-down order) is used for margin calculation.

Example: If there are Cost1=null, Cost2=45, Cost3=50, then Cost2 value will be used to calculate the margin.

Advanced Cost and Simple Cost

By enabling the AdvancedCost module, the Cost configuration in PriceSettingConfig PP and DependencyMappingConfig will become unused and can be removed.

Cost Aggregation Types

Currently, there are three types supported:

- SINGLE - Lookup for only one cost.
- AVG - Lookup for costs in the targeted Product Extension, then return the average of all found values.
- SUM - Lookup for costs in the targeted Product Extension, then return the sum of all found values.

Cost Strategy and Pricing Strategy

Naming Convention

A cost strategy value can be passed to a pricing engine as a parameter. When defining a cost strategy, a proper engine suffix has to be provided. The base name of any engine parameter is always "COST", so if a cost strategy has the "_EXAMPLE" engine suffix, then its engine parameter name is "COST_EXAMPLE".

Usage

In terms of pricing engine parameters, only costs from the CostSelection PP for the current product are loaded + "PRODUCT_COST".

Example:

In CostTypeDefinition PP, there are COST1, COST2, COST3, COST4, COST5, COST6.

In CostSelection PP, it is set to use COST1, COST2, COST3, COST4, COST5.

During the calculation process, COST1, COST3, and COST4 were not calculated properly. It means that the product's cost will be COST2 (first valid calculated cost value) and 3 parameters will be loaded to engines, which are "PRODUCT_COST", "COST2", and "COST5". "PRODUCT_COST" represents the first valid cost and it has the same value as COST2.

Configuration

Set Module Status

Set the module status in the PriceSettingModules PP table whose module name is `PSP_ADVANCED_COST`

Set Cost Strategy Definition

To set up a cost strategy, create a new data row in the CostTypeDefinition PP with details in CostTypeDefinition PP.

Set Cost Strategy Selection

Set selections in the CostSelection PP table which correspond with the current dependency level.
Table naming convention: `<<dependency>>CostSelection`.
Examples: AsiaCostSelection, GlobalCostSelection,...

Using Cost Strategy Engine Parameter (Optional)

For each SKU in a PL/PG run, only its selected cost strategies will be loaded. To pass a cost strategy value as a parameter to a pricing engine, the cost strategy must be selected in the corresponding row in the CostTypeDefinition PP table. Then, add the parameter name in the desired pricing strategy, in the StrategyCalculationParameters property. For details, see StrategyDefinition PP.

⚠ The parameters are passed as inputs to engines and the input order is important, so do not change the default engine's parameters order.

PSP Product Competition Module

This module gathers and displays product competition data. This can be used for any competition-based strategy.

Module Related Elements

Technical Name	Label	Available in Parent PL/PG	Available in Dependent PL/PG	UI Visible	Output Type	Description
RawCompetitionData		Yes	Yes	No	List	The list of competition data obtained from the Competition Data master table
CompetitionData	Competition Data	Yes	Yes	Yes	Matrix	Formatted competition data
RawRelevantCompetitionData		Yes	Yes	No	List	The list of target competitors data obtained from the Competition Data master table
RelevantCompetitionData	Relevant Competition Data	Yes	Yes	Yes	Matrix	Formatted relevant competition data

Mechanism

	Competition data	*Relevant competition data*
Dependency mapping	Set the dependency mapping in the DependencyMapping PP table with the key `Product Competition`. Details can be found at DependencyMappingConfig PP.	
Lookup	Search for records of the current SKU which are at the corresponding dependency level in the CompetitionData master table.	1. Search for target competitors in the corresponding RelevantCompetitionData PP table. • Table name convention: `<<dependency>>RelevantCompetitionData`. Examples: RelevantCompetitionData, AsiaRelevantCompetitionData, GlobalRelevantCompetitionData,... 2. Filter the `Competition data` list to get only data from target competitors.
Data fallback	If none is found, use the dependency fallback mechanism with the mapping type `Lookup`.	

Product Competition Module and Competition-based Strategies

If the module is turned off, null will be passed to the pricing engine. The resulting price will then be null as well and there will be a warning message indicating that there is no competition price to be calculated.

For more information about the competition pricing engine, see Competition Engine.

PSP Price Flexibility Module

This module provides integration with Price Flexibility Package (PFP). It adds the Changes element (technical name: ChangesFromMonitor) to the independent LPG which describes why a product has been automatically added to a price grid.

Currently, multiple features from PFP v1.2 are not supported. Price Flexibility Module will work properly only with configuration using a single monitor instance (or with older versions of PFP).

About Price Flexibility Package

The package is designed to notify a user group of the status when a product is added or updated. It means that changed products will be added to a Live Price Grid and wait for approval/denial by members of the assigned user group. Once the decision is made, approved products will be moved to a configured Price List and denied products will be removed from the Live Price Grid.

Price Flexibility Package monitors all products available in the system. Currently, it is not possible to filter them.

For more information about the Price Flexibility Package, see Accelerate Price Flexibility Package.

Common Configuration Procedures (Price Setting)

The following procedures are often required:

- Add New Dependency Level
- Rename Dependency Level
- Adjustments after Changing the PP PriceSettingLevel
- Change Product Segmentation
- Configure or Add Data Lookup

Add New Dependency Level

Business Requirements

When a new dependency level is required:

- Decide:
  - Is this going to be a “Virtual Dependency Level” (used only as a fallback for other dependency levels which are lower in hierarchy) or a standard one?
  - Is this going be “Parent” or “Dependent on already existing dependency” Level?
    - If Dependent, should it be in master relation or HQ relation?
  - Is this is going be “Complete Dependency Level” (no fallback to more general Dependency Levels, nor general fallback)?
- Prepare data for Dependency Level (or agree to utilize a fallback to some of them).
- Prepare Hierarchical Config for Dependency Level (or agree to utilize a fallback to some of them).

New Dependency Level Check List

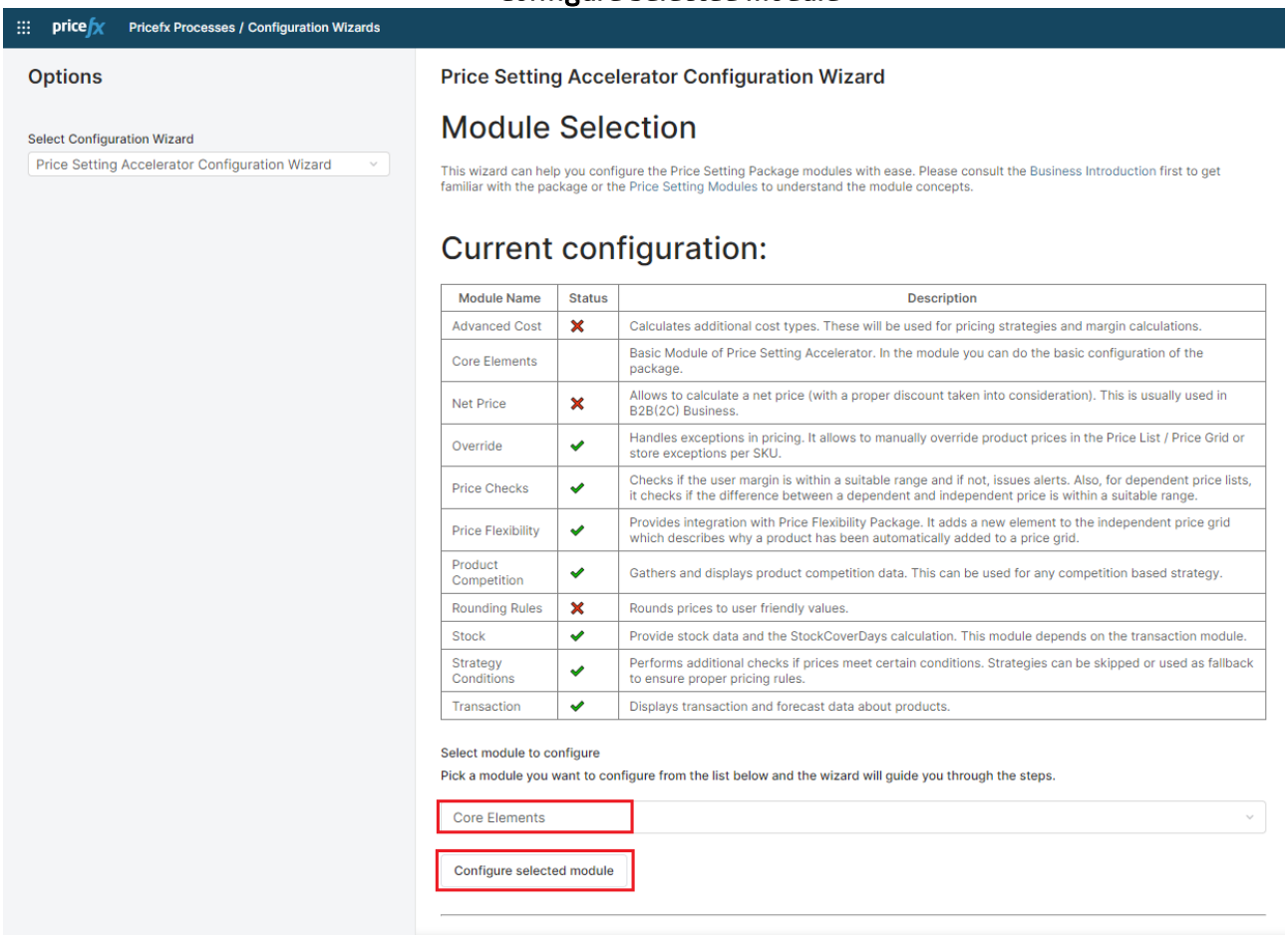
1. Add a new entry to Dependency Configuration. All values will come from business requirements described above.

- Remember to enter PL/PG ID of Master Dependency Level, not just Dependency Level just being added.
- Remember that Preferences are used for Dependency Mapping DependencyMappingConfig PP. PSP does not support different Dependency Mapping per Dependency Level, so it needs to be similar to data entered for other Dependency Levels.

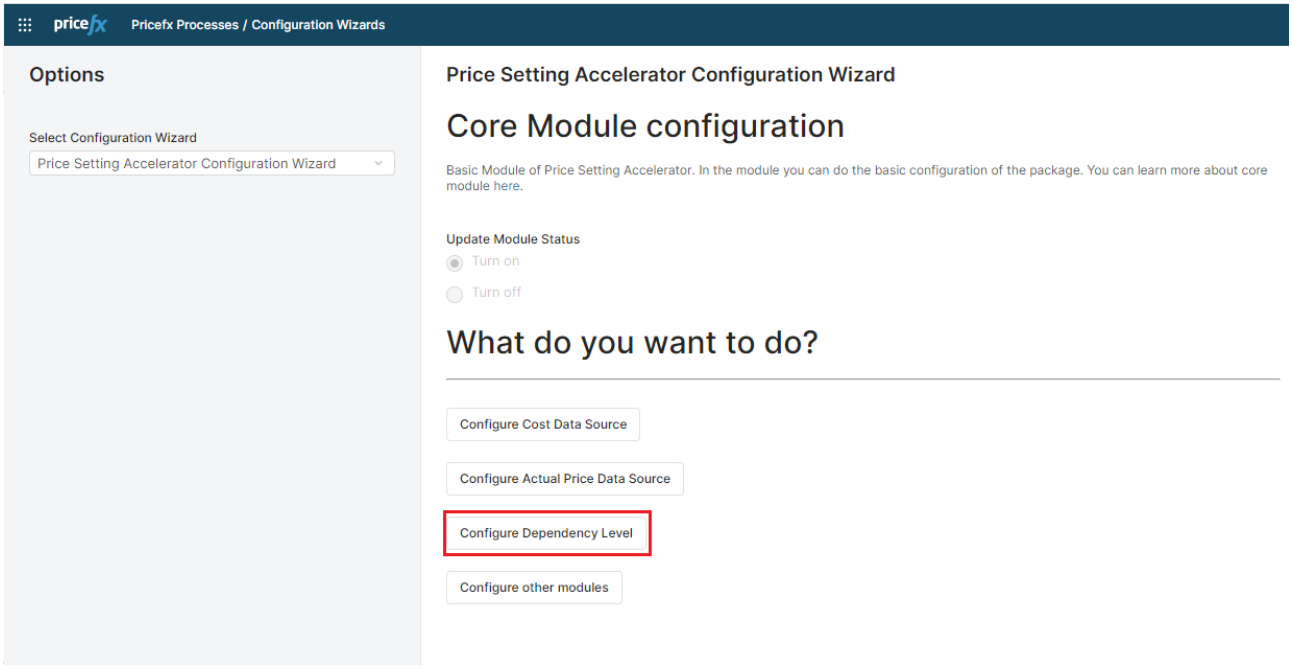
2. Run bootstrapping, according to this guide: Change Product Segmentation
3. Enter data in newly created Hierarchical tables (if needed, fallback might be utilized if Dependency Level is not complete – a general fallback is always utilized). List of all Hierarchical tables: Hierarchical Lookups.
4. Enter data for each Data Lookup (if needed, fallback/general fallback might be utilized if Dependency Level is not complete). List of all Data Lookups: Data Lookups
5. Create a new PL/PG, according to the second part of this section: https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/2321809516/How+to+Deploy+Price+Setting+Accelerator#Configuration-and-Price-List%2FGrid-Creation

Using Price Setting Accelerator Configuration Wizard

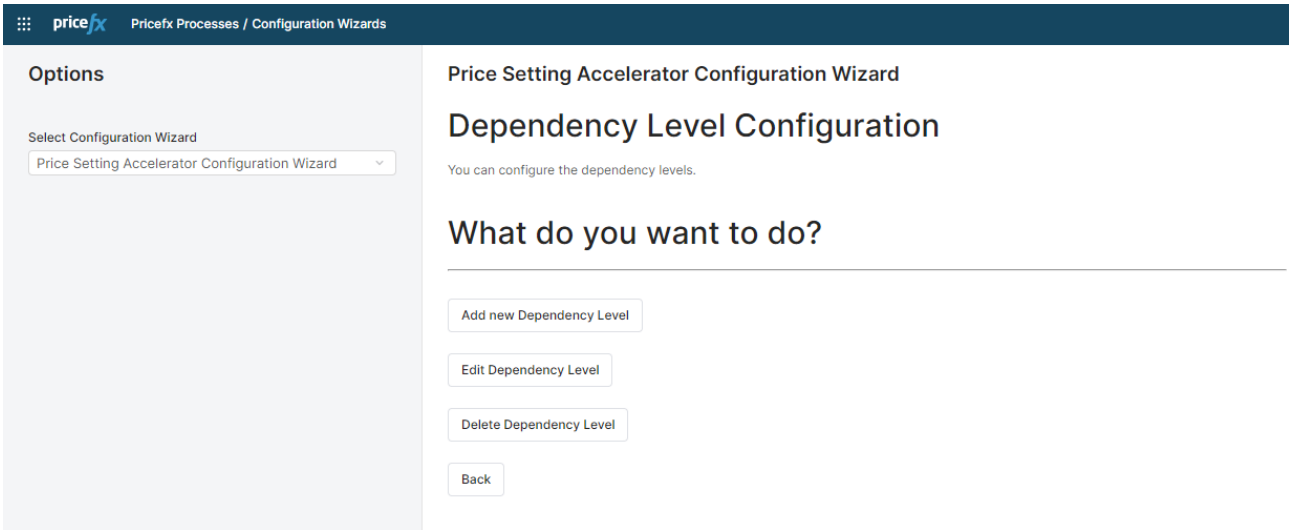
1. Go to **Pricefx Processes** > **Price Setting Accelerator Configuration Wizard**.
2. Select Core Elements module and click **Configure selected module**.



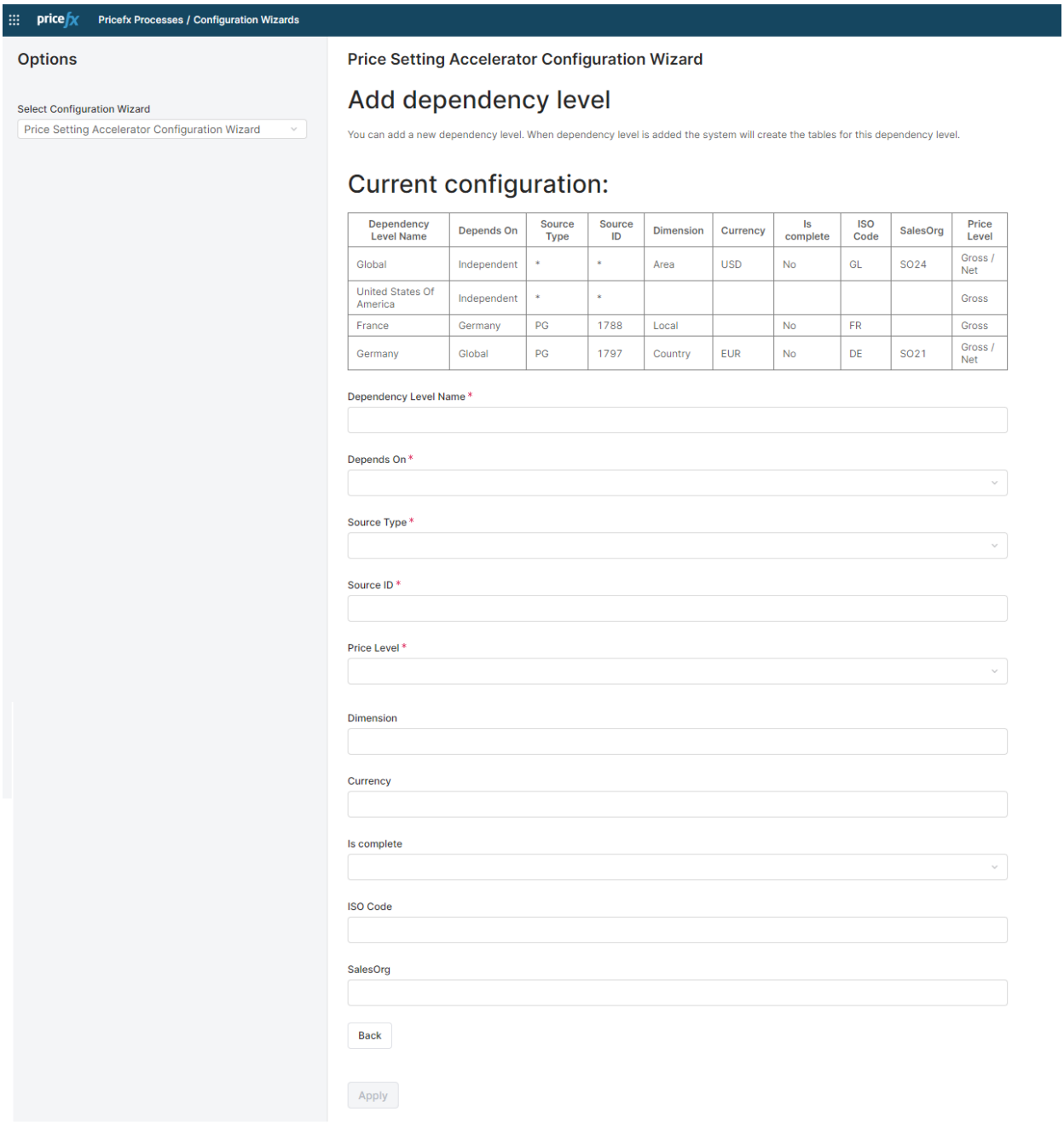
3. Select **Configure Dependency Level**.



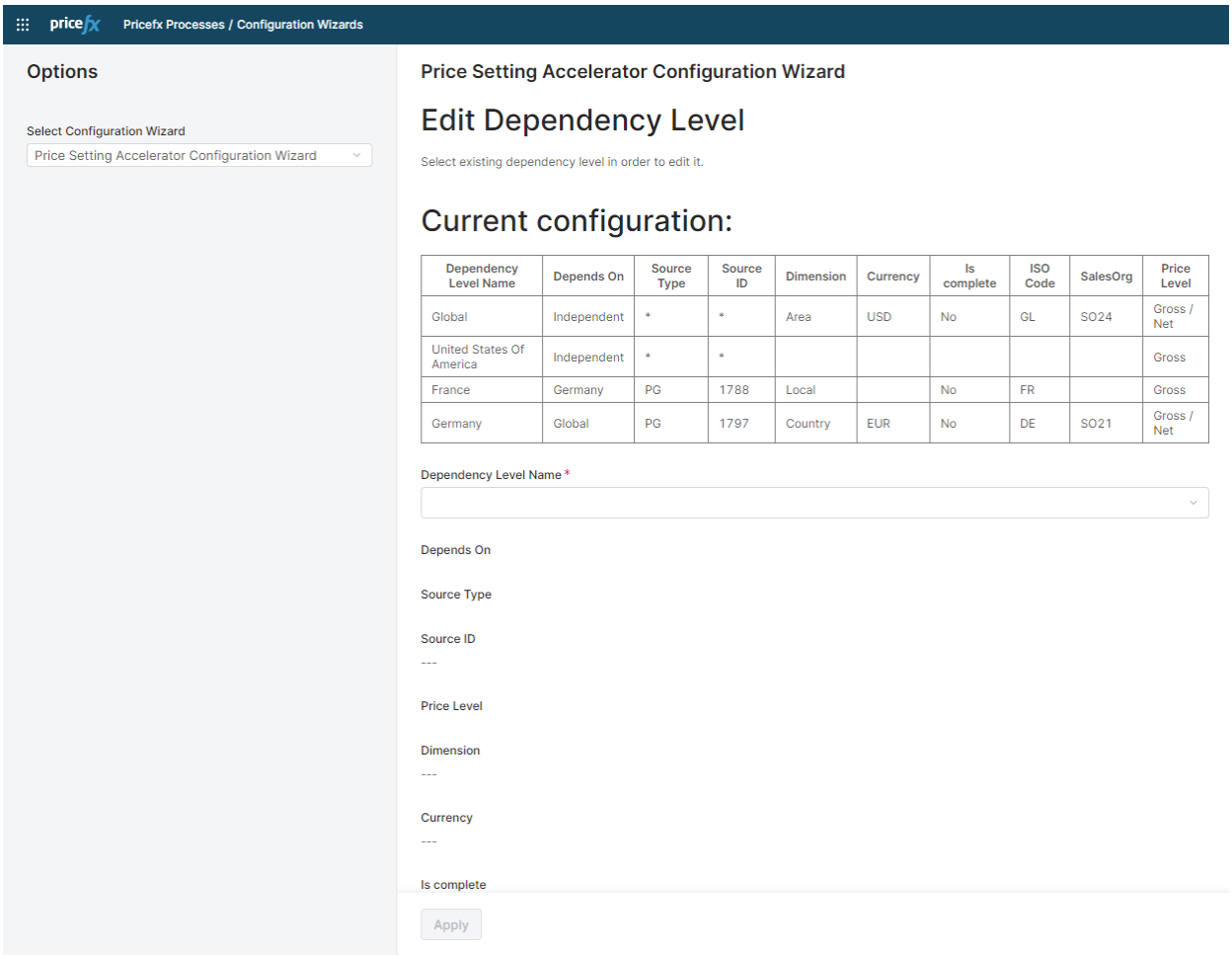
4. Select an action.



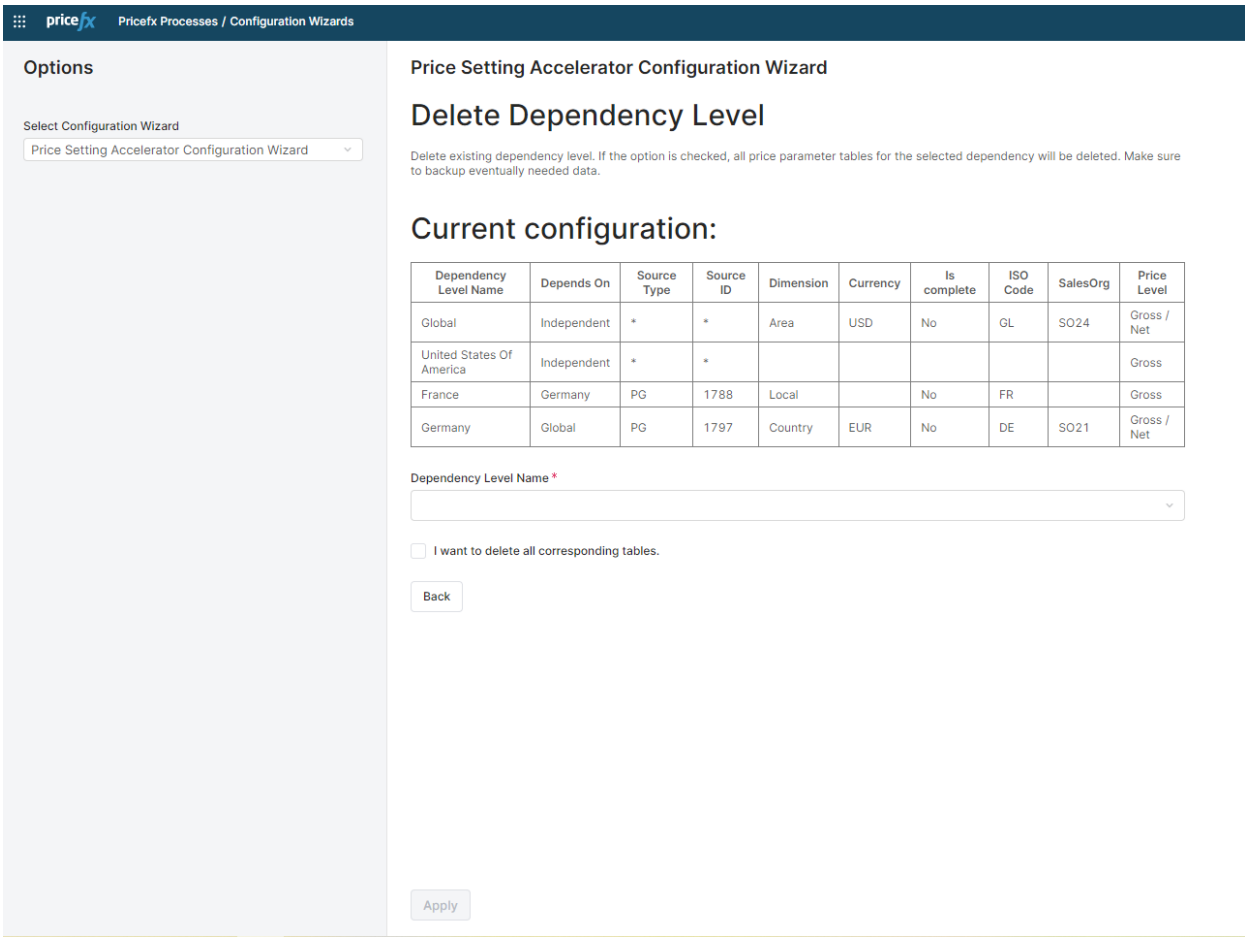
a. Add new dependency level



b. Edit dependency level



c. Delete dependency level



Rename Dependency Level

> ⚠ It is not recommended to make these changes when the system is running in a production environment if you need to have fully functional historical calculation data.

Sometimes it is necessary to rename Dependency Level Name defined in DependencyConfiguration PP. To do that, you have to:

1. Rename it in DependencyConfiguration PP - in the Dependency Level Name column.
2. During the bootstrapping process, we automatically generate PP names based on a new name. Search for all PPs starting with the old name and change it to new one.
3. Rename it in PriceSettingLevel PP.
4. Delete all active LPGs and PLs that were created for this Dependency Level Name and create new ones with the new name selected in the input configuration.
   Because the input changed, old approved PLs will **not be considered as the same dependency level** after the renaming.

Adjustments after Changing the PP PriceSettingLevel

Issues

After changing Price Setting Levels, there can be the following issues:

- Changing Price Setting Level will result in starting/stopping calculation of conditional elements. Calculation will always be performed with same result, but there might be some columns missing or appearing and never changing their values.
- Not only results will become obsolete at the moment of changing Price Setting Level. Running calculation of dependent Dependency Level will result in issues with reading Final Prices of master Dependency Level calculation.

Solutions

1. If possible, create a new PL/PG for all Dependency Levels and update values in Dependency Configuration (PG/PL ID).
2. If not possible, rerun the whole PL/PG. Hide "zombie fields" (never updated) by preferences.
3. Rerun all PLs/PGs starting from Parent Dependency Level and going through hierarchy tree.

Change Product Segmentation

Technical Requirements for Bootstrapping

Detailed description of requirements can be found at https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/3252061372/Hierarchical+Lookups#Configuration-Of-Bootstrapping.

Change Segmentation Check List

1. These configurations need to be updated:
   - PriceSettingDimensions PP
   - DependencyConfiguration PP
2. Choose one of these two options:
   - Run "Price Setting Package - Upgrade" in PlatformManager Marketplace if you have the newest version of Accelerator. If you do not have the newest version, but you want to upgrade it anyway, follow Upgrade Steps (Price Setting).
   - Do it manually:
     - Deploy all Temporary Hooks. These are added to this page as attachments. You can change their folder, if you want new PPs to be created in another place.
     - Check for any changes in "CF_BuildPricingTables" in Accelerator repository, if there was any fixed bug.
     - Run logic named "CF_BuildPricingTables". It can be done through CF or as a debug (with "Allow object modification" checked in).
3. Clean up.
   - Remove Price Parameters which are not used anymore.
     - Price Parameters from old Dependency Configuration are not removed automatically.
     - There are a few restrictions on changing Dimensions, so always check if column names have been updated correctly. If the number of keys decreased and one of the keys moved to a previous position, then PP could become corrupted. Then it needs to be removed and added manually.
   - Add values to new Hierarchical Tables.

Troubleshooting

- If you run bootstrapping during calculation, calculation is almost sure to fail. Most configurations are kept in cache for all products in PL/PG. Be sure to rerun after bootstrapping is finished.
- If bootstrapping did not work properly for any reason, a bug should be reported. However, wrong generation of Hierarchical Price Parameters will damage the whole Accelerator. If you are going to fix any issues manually, before Accelerator Team can respond:

- Make sure that you have the newest, not modified version of CF_BuildPricingTables.
- Remember to deploy TemporaryHooks every time before you run bootstrapping.
- Do not be afraid to run bootstrapping multiple times. The worst thing that can happen is that config is lost (which should be backed up anyway) or there are more leftover PPs on the partition (if the config is different every time).
- Bootstrapping just creates Hierarchical Price Parameters (and removes Temporary Hooks). It can be done manually, but it will be time consuming. Config describing how data is read is in Price Parameters PriceSettingDimensions and DependencyConfiguration.

Configure or Add Data Lookup

Data Lookups are managed using the ConfigurableLookups feature. Lookups are defined by configurations either placed in the LookupConfiguration Company Parameter or directly in the code. For adding a new Configurable Lookup, refer to the page linked above. For changing an existing Configurable Lookup, you have the following approaches:

- **Product Competition or StrategyConditions** - These lookup definitions are described in Groovy code. To modify them, search for the `libs.ConfigurableLookupsLib.LookupInstanceFromPP.prepareConfiguredLookup` function with the parameters "ProductCompetition" and "StrategyConditions."
  - Be aware that these are placed in Groovy because changing the configuration will break related wizards.
  - Remember that Groovy configurations take priority over Company Parameter configurations. You can adjust every lookup in Groovy, or you can move the ProductCompetition and StrategyConditions configurations to the Company Parameter by deleting the Groovy config.
- **Modify lookups in PSP Wizards** - This option is more limited than changing the Company Parameter itself. Currently, the UI supports all use cases that were possible before PSP v2.3.0 (pre-Configurable Lookups).
- **Modify Company Parameter LookupConfiguration** - To learn more about LookupConfiguration, refer to the link above. Note that all custom changes made to LookupConfiguration might be "fixed" by a wizard that targets the same table.

Technical Information (Price Setting)

This is the main section for documentation on Price Setting Package architecture. You can find here information about inner workings of the package and more advanced configuration concept descriptions.

- Warning Handling
- Error Handling Deep Dive
- Caching Lookup Results
- Batching
- Price Setting Configuration Wizard - Technical Design
- Logic Element Names for Approval Workflow (Price Setting)
- Module Dependencies

Standard Configuration

The Price Setting Package comes with some standard configuration. This includes some structures (PX, PP) as well as pre-configured data in configuration PPs.

The following items are included in a standard configuration:

- Product Extensions
  - ProductCosts - Used by core Cost lookups
  - ListPrices – Used by Actual Price lookups
  - PromotionPrices – Used by sample engine configuration
  - RecommendedRetailPrices – Used by sample engine configuration
- Price Parameters
  - Sample configurations for various pre-configured engines
    - AnchorAdditionalConfig
    - AttributeBasedPricingRules
    - AvgCompetitionAdditionalConfig
    - CostPlusAdditionalConfig
    - MaxCompetitionAdditionalConfig
    - MinCompetitionAdditionalConfigMin
    - PriceIncreaseAdditionalConfig
    - PromotionLookupEngineConfig
    - RRPLookupEngineConfig
  - Data containers configured for different features:
    - AnchorData
    - PricingExceptions – Data container for strategy and price exceptions
    - StockData – Data container for product stock data
- Pre-Configuration for Price Setting Package that **has to be completed**:
  - Configuration for lookups in included data containers (LookupConfiguration PP)
  - Configuration for dependency mapping in some of the included data containers and product extensions
  - Sample of complete configuration that has to be completed (mostly PriceSettingConfig PP)
- Sample for Pricing Strategy Definition
- Sample for Rounding Rules Configuration
- Sample for Pricing Exceptions

Configuration Company Parameters (Price Setting)

- Bootstrapped Tables
- Price Setting Config PP
- Module Configuration Tables
- Other Configs

Bootstrapped Tables

- Additional Discount Lookup
- Adjusted Price Corridor Lookup
- Base Strategy Selection Lookup
- Cost Plus Lookup
- Cost Selection Lookup
- Dependency Level Adjustment Lookup
- Discount Lookup
- List Price Corridor Lookup
- Margin Alert Lookup
- Minimum Margin Lookup
- Price Increase Lookup
- Relevant Competition Data Lookup
- Strategy Selection Lookup
- Volume Breakdown Lookup

Additional Discount Lookup

Name

{nameOfDependency}AdditionalDiscount"DependencyConfiguration"

Attributes

All default values are empty.

- Discount %

Description

Value of minimum Discount in percent for the Target Price engine. It is one of the supportive parameters coming with pre-configured strategies. This value is a difference between Final Price (at the end of calculation) and selling price (e.g. rebates).

Adjusted Price Corridor Lookup

Name

- AdjustedPriceCorridor

Attributes

All default values are empty.

- Minimum Absolute
- Minimum Corridor
- Maximum Corridor
- Maximum Absolute

Description

Value of Adjusted Price Corridor thresholds in percent for the Price Checks module. Difference between the price from Master Dependency Level and current price is compared and quotient is showed as an alert, colored depending on this config.

Base Strategy Selection Lookup

Name

{nameOfDependency}BaseStrategySelection

Attributes

All default values are empty.

- Price Strategy #1
- Price Strategy #2
- Price Strategy #3
- Price Strategy #4
- Price Strategy #5
- Price Strategy #6
- Price Strategy #7
- Price Strategy #8
- Price Strategy #9
- Price Strategy #10
- Price Strategy #11
- Price Strategy #12
- Price Strategy #13
- Price Strategy #14
- Price Strategy #15
- Stop After First Price Found
  - Default value is empty (= false).

Description

Order of used base strategies. Base strategies appear before the standard strategies. However, if they fail, they are removed from “Prices” popup.

Cost Plus Lookup

The out-of-the-box pricing strategy Cost+ needs to store the “plus” value. This table stores such “plus” value in the form of percent or money value.

Whether the percentage or money value are used depends on an additional configuration of the Cost+ strategy.

The value of “plus” can be configured for a combination of:

- **Pricing Level** - given by the table name
- **Product Segment** - set up as values of the keys

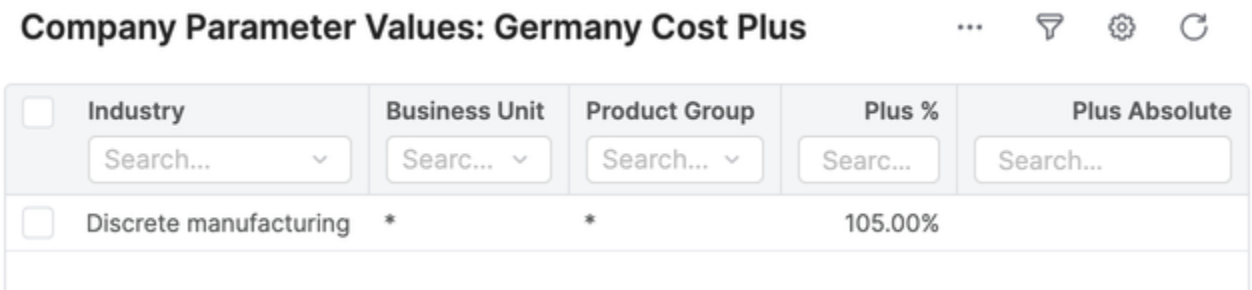


Table Name

- {nameOfDependency}CostPlus
- CostPlus - fallback table for cases when no relevant records are found in the relevant {nameOfDependency}CostPlus tables

Keys

All default values are empty.

- Product Segment attributes

Attributes

Default values of attributes are empty.

- Plus %
- Plus Absolute

Cost Selection Lookup

Name

- {nameOfDependency}CostSelection

Attributes

All default values are empty.

- Cost Type #1
- Cost Type #2
- Cost Type #3
- Cost Type #4
- Cost Type #5

Description

Order of Cost Types calculated for the Advanced Cost module.

Dependency Level Adjustment Lookup

One of the strategies to calculate dependent price lists is to take a price from other price list (master price list) and adjust it by % adjustment.

This table stores the size of such adjustment per:

- **Pricing Level** - given by nameOfDependency in the table name
- **Product Segment** - given by the keys of the table

This table is valid only for Dependent PL/PG.

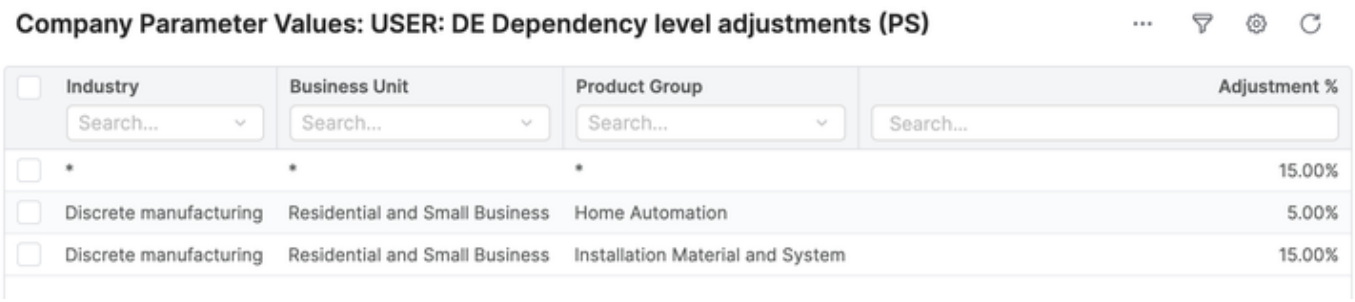


Table Name

- {nameOfDependency}DependencyLevelAdjustment

Keys

All default values are empty.

- Product Segment attributes

Values/Attributes

All default values are empty.

- Adjustment %

Discount Lookup

Name

{nameOfDependency}Discount

Attributes

All default values are empty.

- Discount %

Description

Value of minimum Discount in percent for Net Price module. This value is a difference between Gross and Net (Final) Price.

List Price Corridor Lookup

Name

- ListPriceCorridor

Attributes

All default values are empty.

- Minimum Absolute
- Minimum Corridor
- Maximum Corridor
- Maximum Absolute

Description

Value of List Price Corridor thresholds in percent for the Price Checks module. Difference between the price from Master Dependency Level and current price is compared and quotient is showed as an alert, colored depending on this config.

Margin Alert Lookup

Name

- MarginAlertsForPriceLists

Attributes

All default values are empty.

- Critical Threshold
- High Threshold
- Medium Threshold

Description

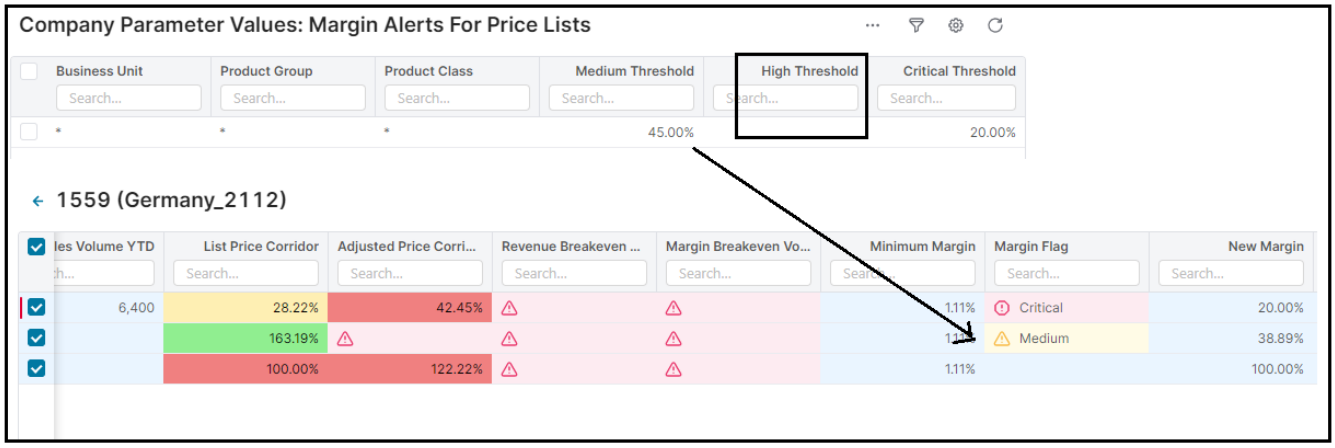
It is the value of Margin Alerts thresholds in percent for the Price Checks module. The `Margin Flag` element in the PL/PG has the values Critical, High, or Medium, depending on what margin falls into what range. For Critical and High ranges, it will color the row in red. For the Medium range, it will color the row in yellow.

The checking order is: critical threshold => high threshold => medium threshold (too small margin => small margin => medium margin). It is not in data order (not from the smallest to biggest or vice versa).

Example: Given medium threshold = 45, high threshold = 100, critical threshold = 20, margin = 38.89, the checking process will be:

- 38.89 <= critical (20)? No
- 38.89 <= high (100)? Yes

Therefore, the margin falls into the high range.



Minimum Margin Lookup

Name

{nameOfDependency}MinMargin

Attributes

All default values are empty.

- Min Margin %

Description

Value of the minimum margin in percent for the Price Checks module.

Price Increase Lookup

The out-of-the-box pricing strategy Price Increase needs to store the “increase” value. This table stores such “increase” value in the form of percent or money value.

Whether the percentage or money value are used depends on the additional configuration of the Price Increase strategy.

The value of “increase” can be configured for a combination of:

- **Pricing Level** - given by the table name
- **Product Segment** - set up as values of the keys

Company Parameter Values: USER: 904 Price Increase setup (PS)

Industry	Business Unit	Product Group	Increase %	Plus Absolute
Search...	Search...	Search...	Searc...	Search...
Discrete manufacturing	Residential and Small Business	Electrical Protection and Control	200.00%	10.00
Discrete manufacturing	Residential and Small Business	Home Automation	0.00%	10.00
Discrete manufacturing	Residential and Small Business	Light Switches and Electrical Sockets	50.00%	10.00
Discrete manufacturing	Residential and Small Business	Network Infrastructure and Connectivity	130.00%	10.00
Discrete manufacturing	Residential and Small Business	Services	80.00%	100.00
Discrete manufacturing	Residential and Small Business	Uninterruptible Power Supply (UPS)	25.00%	12.00

Table Name

- {nameOfDependency}PriceIncrease
- PriceIncrease - used as a fallback if no related records are found in the relevant {nameOfDependency} PriceIncrease tables

Keys

All default values are empty.

- Product Segment attributes

Attributes

All default values are empty.

- Plus %
- Plus Absolute

Relevant Competition Data Lookup

Name

- {nameOfDependency}RelevantCompetitionData

Attributes

All default values are empty.

- Competitor #01-#29

Description

List of relevant competitors. It filters out values from Competition Data Lookup to create a list of Relevant Competitors, used for a popup display and pricing.

Strategy Selection Lookup

The StrategySelection table defines order of strategies used for a certain Product Segment.

When exceptions / overrides / base strategies are used, they will be used before the strategies defined here.

Name

{nameOfDependency}StrategySelection

Keys

All default values are empty.

- Product Segment attributes

Attributes

All default values are empty.

- Price Strategy #1
- Price Strategy #2
- Price Strategy #3
- Price Strategy #4
- Price Strategy #5
- Price Strategy #6
- Price Strategy #7
- Price Strategy #8
- Price Strategy #9
- Price Strategy #10
- Price Strategy #11
- Price Strategy #12
- Price Strategy #13
- Price Strategy #14
- Price Strategy #15
- Reverse Calculation Order – It defines if the calculation should be done from bottom to top, instead of the default top to bottom. This is only a technical change, important if you are utilizing the ENGINE_RESULTS parameter, which contains the result of all previous calculations.
- Prioritize Parent Level Price – It defines if a price coming from the master Dependency Level should be placed before the strategies defined here or after them. This setting is relevant only for Dependent price list/grid.
  - Default value is empty (= false).
- Stop After First Price Found
  - Default value is empty (= false).

Volume Breakdown Lookup

Name

- {nameOfDependency}VolumeBreakdown

Attributes

All default values are empty.

- Volume #01
- Discount #01
- ...
- Volume #15
- Discount #15

Description

Pairs consisting of Volumes in integers and Volume Discounts in percent. These values will be used in generating Matrix PG and discounts will be applied to prices.

Price Setting Config PP

Each of the subpages represents a single row in the PriceSettingConfig Price Parameter.

- Actual Price Lookup
- Debug Mode
- Exceptions and Manual Override Allowance Config (PriceSettingConfig)
- Forecast Config (PriceSettingConfig)
- Forecast Lookup
- Last Period Config (PriceSettingConfig)
- Transaction Lookup

Actual Price Lookup

This configuration represents a single row of the PriceSettingConfig Price Parameter.

Column	Value	Description
Key	Actual Price	Defines how to lookup values for the ProductListPrice lookup.
Condition	*	Condition is not needed here. Leave it at the default value (= *).
Type	Lookup	Hardcoded value.
Source	• ConfigurableLookup • PL • PG	• ConfigurableLookup performs lookup based on the data in LookupConfiguration PP. The entry to use will have “ActualPrice” as its key. • PL/PG perform lookups based on a predefined algorithm.

Why This Lookup Is Not Standardized

Technically, Actual Price PX lookup is standardized. However, from the configuration point of view, flow of data reading is non-standard when using PL and PG sources.

PL and PG do not use batching. PG is not even a lookup as such, since it reads data from the previous run of the same price grid.

Debug Mode

This configuration represents several rows of the PriceSettingConfig Price Parameter. It allows users to add their own log messages and thus makes it easier to troubleshoot issues and better understand the package. All messages should be well formatted and as informative as possible for the users.

Feature Setting

Allows user enable or disable debug mode

Column	Value	Description
Key	Debug Mode	
Condition	Debug	
Type	OFF/ON	• OFF - Disables the debug mode. This is the default value. • ON - Enables the debug mode.

Default Setting

This is the default setting for all modules, it is used if the module is missing or the value Type is empty.

Column	Value	Description
Key	Default	
Condition	Debugger	
Type • NONE • LOG • TRACE • ALL		• NONE - Debug mode is off. • LOG - Shows logs using `api.logInfo()`. • TRACE - Shows logs using `api.trace()`. • ALL - Shows logs using `api.logInfo()` and `api.trace()`. This is the default value.

Module Setting

Column	Value	Description
Key	Module name • Warning Manager • Lookup Tracer • Price Calculator • Custom • Config Manager	
Condition	Debugger	
Type • NONE • LOG • TRACE • ALL		• NONE - Debug mode is off, it is not overridden from Default Setting. If we set Type = null, it will be overridden by Default Setting. • LOG - Shows logs using `api.logInfo()`. • TRACE - Shows logs using `api.trace()`. • ALL - Shows logs using `api.logInfo()` and `api.trace()`. This is the default value.

Exceptions and Manual Override Allowance Config (PriceSettingConfig)

This configuration represents a single row of the PriceSettingConfig Price Parameter.

One configuration row should exist for every key-condition pair mentioned in the table below.

Column	Value	Description
Key	• Parent Manual Override Allowance • Parent Manual Override Allowance	Holds the configuration of exception/override allowance. One row should be configured for every entry.
Condition	• Strategy • Price	Defines which exception type is configured. There is one row for each condition.
Type	• Yes • No • LineLevel • ExceptionTable	Defines if and what kind of exception is allowed: • LineLevel - Only line level manual overrides are available. Users can define them within PL/PG per product. Exception table records are not checked at all. • ExceptionTable - Only exceptions through the exception tables are available (defined in ExceptionConfig). It also means that line level manual overrides in elements "ManualPrice", "ManualPriceReason" and "PriceSelector" are cleared. • Yes - Both exception types are allowed. The importance hierarchy is as follows: 1. Price Manual Override 2. Strategy Manual Override 3. Price Exception 4. Strategy Exception • No - Exceptions are not allowed. Elements "ManualPrice", "ManualPriceReason" and "PriceSelector" are hidden from the user.

Forecast Config (PriceSettingConfig)

This configuration represents a single row of the PriceSettingConfig Price Parameter.

There are rows for forecasts, one for every quarter, so there will be key pairs "Forecast-Q1", "Forecast-Q2" etc.

Each quarter can be configured with one of these three types:

- Last Year
- Linear
- Lookup

Last Year

Column	Value	Description
Key	Forecast	Defines how to calculate the forecast.
Condition	• Q1 • Q2 • Q3 • Q4	Logic checks the current date. If it matches this configuration row, the row is used.
Type	LastYear	The forecast will be equal to last year's sales (based on the transaction source).

Linear

Column	Value	Description
Key	Forecast	Defines how to calculate the forecast.
Condition	• Q1 • Q2 • Q3 • Q4	Logic checks the current date. If it matches this configuration row, the row is used.
Type	Linear	The forecast will be calculated with a linear growth based on the year to date values from the transaction source.

Lookup

See Forecast Lookup.

Forecast Lookup

This configuration represents a single row of the PriceSettingConfig Price Parameter.

⚠ There are also other ways of reading Forecast; Lookup is only one of them.

Column	Value	Description
Key	Forecast	Defines how to calculate the forecast.
Condition	• Q1 • Q2 • Q3 • Q4	If the current date matches this configuration row, the row is used. This is checked by the logic.
Type	Lookup	The logic will perform a lookup for the forecast from an external source.
Source	• Datamart • Datasource • PX	Source type where the forecasts are stored.
Source Table	{name of the table with forecasts}	
Source Field	{name of the column with the turnover}	
Source Field 2	{name of the column with the quantity}	
Source Field 3	{name of the column with SKU}	Only PX has a default SKU, so we require SKUs for all tables for consistency.
Source Field 4	{name of the column with the invoice date}	

		The forecast will always be fetched using only data with the "next year" filter applied on this column. ⚠ The column must be of the Date type.
Source Field 5	{name of the column with currency of turnover}	Turnover currency
Source Field 6	{name of the column with minimum value of aggregated data}	Only for pre-aggregated PX Source
Source Field 7	{name of the column with maximum value of aggregated data}	Only for pre-aggregated PX Source
Source Field 8	{name of the column with average value of aggregated data}	Only for pre-aggregated PX Source

Why This Lookup Is Not Standardized

- Hierarchy works differently for Transaction data (it goes down for all sub-levels of dependency, instead of working as fallback).
- A lot of data will be returned for a given time period, there is a "date overlap" issue.
- Lookup manager supports api.stream calls, while transaction data might be stored in Datamart or Data Source.

Last Period Config (PriceSettingConfig)

This configuration represents a single row of the PriceSettingConfig Price Parameter. It is not related to lookup sources. It is a candidate to be moved to the module-specific table. For now, ignore column names in that config.

Column	Value	Description
Key	Last Period Transaction	Defines from which period data should be looked up
Codition	*	Condition is not needed here. Leave it at the default value (= *).
attribute1	• Years • Months • Weeks • Days	Time unit of lookup configuration
attribute2	{any integer}	Amount of time units of lookup configuration

Transaction Lookup

This configuration represents a single row of the PriceSettingConfig Price Parameter.

Column	Value	Description
Key	Transaction Source	Defines where data about transactions are stored.

Condition	*	Condition is not needed here. Leave it at the default value (= *).
Type		Leave blank.
Source	• Datamart • Datasource • PX	Type of the table where transactions are stored.
Source Table	{name of the table with transactions}	
Source Field	{name of the column with the invoice price}	Used for turnover calculation.
Source Field 2	{name of the column with the quantity field}	Used for volume calculation.
Source Field 3	{name of the column with SKU}	Only PX has a default SKU, so we require SKUs for all tables for consistency.
Source Field 4	{name of the column with the date for the invoice}	⚠ The column must be of the Date type.
Source Field 5	{name of the column with the currency}	Invoice price currency
Source Field 6	{name of the column with minimum value of aggregated data}	Only for pre-aggregated PX Source
Source Field 7	{name of the column with maximum value of aggregated data}	Only for pre-aggregated PX Source
Source Field 8	{name of the column with average value of aggregated data}	Only for pre-aggregated PX Source

Why This Lookup Is Not Standardized

- Hierarchy works differently for Transaction data (it goes down for all sub-levels of the dependency, instead of working as a fallback).
- A lot of data will be returned for the given time period, there is a "date overlap" issue.
- Lookup manager supports api.stream calls, but transaction data might be stored in a Datamart or Data Source.

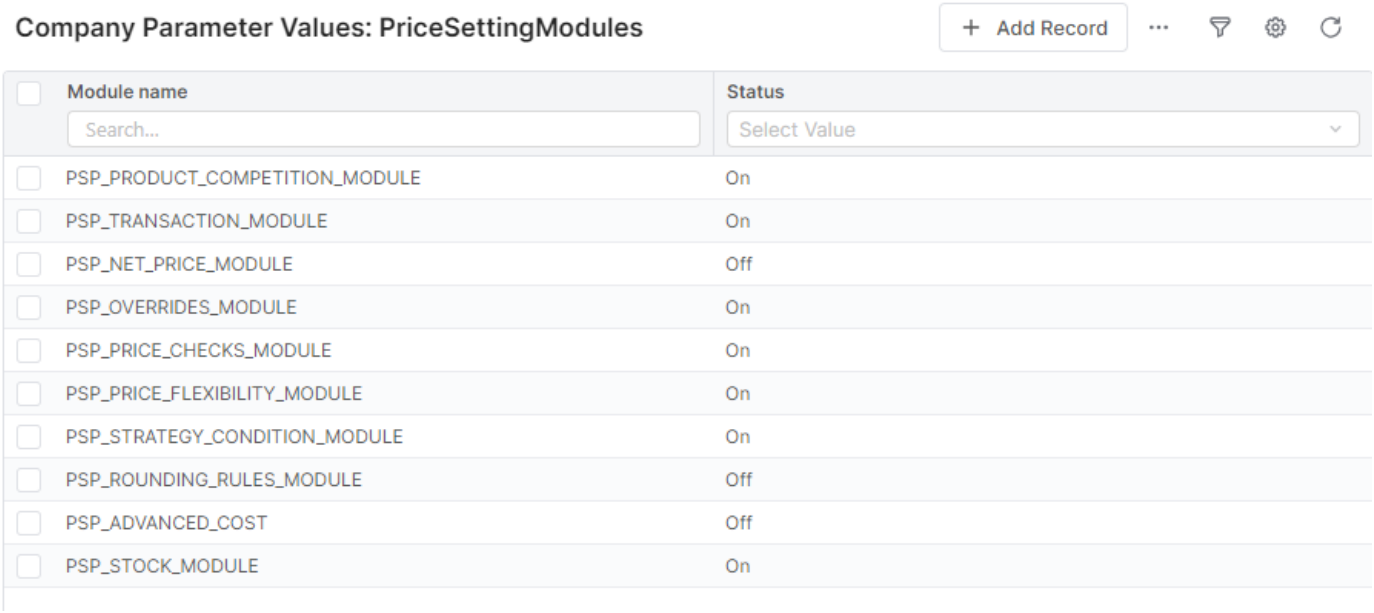
Module Configuration Tables

- PriceSettingModules PP
- For PSP Advanced Cost Module
- For PSP Strategy Conditions Module
- For PSP Override Module

PriceSettingModules PP

Column Name	Values	Description
Name	• PSP_ADVANCED_COST • PSP_NET_PRICE_MODULE • PSP_OVERRIDES_MODULE • PSP_PRICE_CHECKS_MODULE • PSP_PRICE_FLEXIBILITY_MODULE • PSP_PRODUCT_COMPETITION_MODULE • PSP_ROUNDING_RULES_MODULE • PSP_STRATEGY_CONDITION_MODULE • PSP_TRANSACTION_MODULE • PSP_STOCK_MODULE	Technical name of the module. ⚠ All values needs to be present in the PP for the package to work correctly.
Status	• On (default) • Off	Defines whether any given module is enabled or disabled.

Example:



For PSP Advanced Cost Module

- Advanced Cost Lookup (CostTypeDefinition PP)

Advanced Cost Lookup (CostTypeDefinition PP)

Column Name	Values	Description
Cost Type	Any string	Name of the definition, used in cost selection.
Calculation Engine Suffix	Any string	String concatenated with "COST" and passed to engines as a parameter.
Type	"Lookup"	Other types will be added in the future.
Lookup Name		Specifies a name of a lookup; the whole LookupConfiguration is then done through Configurable Lookups.

Calculation Method	• SINGLE • AVG • SUM	For details see Advanced Cost.
Dependency Field	Name of the column in Dependency Configuration (country mapping feature)	
Dependency Type	• Table • Lookup (country mapping feature)	
Mapping Source Field	Name of the PX column (country mapping feature)	

For PSP Strategy Conditions Module

- StrategyConditions PP

StrategyConditions PP

For additional details see also PSP Strategy Conditions Module.

Column Name	Values	Description
Order	{incrementing integer}	Defines the order of conditions. Conditions are checked one by one and in case of multiple conditions and strategies, the result may vary based on the order.
Condition	{condition}	Expression to be evaluated (Boolean value to activate the rule). For details see PSP Strategy Conditions Module.
Rule	{rule}	Defines what to do with the strategy when the condition is evaluated to true. For details see PSP Strategy Conditions Module.
CheckException	• Yes • No	Determines if this condition should be applied to exception prices (both strategy and price exceptions). The check is performed only for strategies on the left side of the condition field. Default value: empty (=No)
Rule Explanation	{text field with explanation}	Shows a reason when a Strategy Condition was applied so that users understand what rules influenced the pricing structure. E.g. “Skipped because price was outside allowed corridor”.
Dependency Level		Refer to dependency configuration.

For PSP Override Module

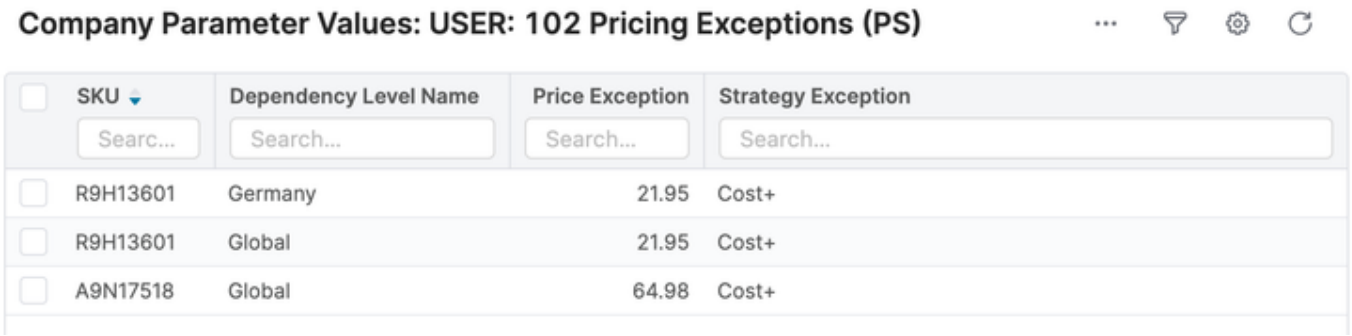
- PricingExceptions PP

PricingExceptions PP

> ⚠ This table is used for 2 separate (i.e. not anyhow related) use-cases: 1) Price Exception and 2) Strategy Exception.

Column Name	Description	Example Value
SKU*	{SKU from Product master}	MB-0001
Dependent Level Name*	{name of dependency level} as defined in DependencyConfiguration PP	Global
Price Exception	Price used for this product in the price list calculated for specified pricing level	20.0
Strategy Exception	Defined strategy (as stated in StrategyDefinition PP) exception for given parameters	Cost+

Examples:



Other Configs

The configuration options described here may be required to set up to enable individual Price Setting modules and features.

- DependencyConfiguration PP
- DependencyMappingConfig PP
- PriceSettingDimensions PP
- ExchangeRates PP
- CompetitionAdditionalConfig PP
- StrategyDefinition PP
- VolumeBreakdownExceptions PP

- WarningConfig PP
- PriceSettingLevel PP

DependencyConfiguration PP

Company Parameter DependencyConfiguration defines:

- All Pricing Levels used for price setting
- Dimension of Pricing Levels
- Relations (dependencies) between the Pricing Levels
- Properties (preferences) of each Pricing Level which can be used when looking up data in tables

This is one of the core tables whose content is uploaded during the accelerator installation process.

Field Name	Description	Data Type	Example
Dependency Level Name	Name of this Pricing Level. Unique ID, it should represent name of the pricing environment.	Text value defined by business user	Germany
Depends On	The pricing level can depend on another level. Here is the name of another Pricing /Dependency Level from this table, on which it depends.	Text, with allowed values: • "Independent" – for independent level • <name of another dependency level listed in this table> – for dependent levels	Global
Source Type	Type of a price list. In some scenarios we need to read prices from the parent (i.e. from a price list which this one depends on). The referenced price list contains prices which will serve as a starting point for price calculation.	Text, with allowed values: • * (used for independent pricing levels) • PG • XPG • PL • XPL	PG
Source ID	ID of a price list. In some scenarios we need to read prices from the parent (i.e. from a price list which this one depends on). The referenced price list contains prices which will serve as a starting point for price calculation.	Text, with allowed values: • * (used for independent pricing levels) • <id of the source PL/PG>	123
Dimension	Name of dimension where the level belongs (for details on dependencies see Pricing Levels).	Text • *empty* – used for independent level • <name of dimension> – this dimension name will be the same for all pricing levels from the same dimension	Country Warehouse
Currency	Currency Code used for calculations of prices on this pricing level. If we read prices from the parent price list and currencies differ, the accelerator will perform a currency conversion.	Text	EUR

IsComplete	Flag indicating if all data is supposed to be filled/available at this point when doing data lookups on this level or dependent levels. • "Yes" – Fallback hierarchy will be stopped at this point. • "No" – When looking up data, and they are not found on this level, we continue to "fall back" to higher levels to find the data.	Boolean, with allowed values: • Yes • No (default)	No
Preference #01-27	Custom parameters, used in mapping the dependency level to the appropriate rows of a given source table. The mapping is managed via the DependencyMappingConfig PP table. Currently, only one field can be used to map at a time (see also Dependency Mapping Mechanism).		ISO Code SalesOrg

Example:

Price Parameter Values : DependencyConfiguration [8]

Dependency ...	Depends On	Source Type	Source ID	Dimension	Currency	Is complete	ISO Code	SalesOrg
EU	Independent	*	*	Area	EUR	Yes	EU	SO00
Germany	EU	PG	155	Country	EUR	No	DE	SO20
Poland	Germany	PG	381	Country	PLN	No	PL	SO21
Asia	Independent	*	*	Area	EUR	Yes	GL	SO10
Warehouse1	Asia	*	*	Warehouse	EUR	No	UK	SO15
Warehouse2	Asia	*	*	Warehouse	EUR	No	UK	SO15
Webshop	Germany	PG	381	Channel	EUR	No	DE	SO20
StationaryShop	Germany	PG	381	Channel	EUR	No	DE	SO20

DependencyMappingConfig PP

Dependency mapping defines how different lookup data will be filtered when loaded from a dependent price list. It is available in the DependencyMappingConfig PP and it directly impacts configuration found in PriceSettingConfig PP. Entries in this table have to be present even if you intend to use only the independent level price list.

Column	Value	Description
Key	• Cost • Discount Group • Transaction • Projection • Price Exception • Strategy Exception • Actual Price • Product Competition • Rounding • Stock • Strategy Conditions	Defines how to filter out values from different dependencies. There must be one row for every key defined in this table.
Dependency Field		The logic will lookup the Dependency Configuration PP for a dependency being calculated and take a value from the

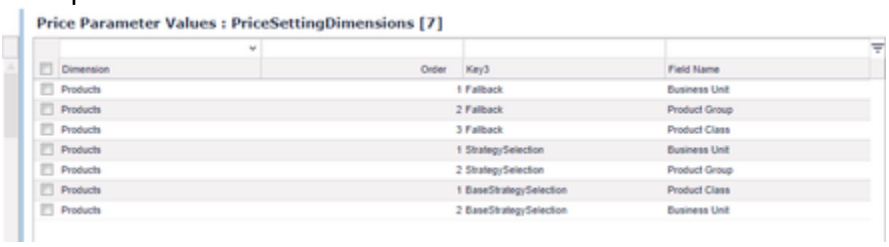
	{name of the label in the DependencyConfiguration PP}	column configured in this field. It will be used as a filter for source data defined in the Source Table row. This filter will be applied to a column configured in Source Field of this configuration row. In the start, it has the "ADJUSTME" value and it must be changed. Special use cases: • Price Exception / Strategy Exception Dependency Mapping – We allow to use MATRIX typed PPs with only one key when the configured type is "PP". In this case, the Condition field value has to be "-" and dependency mapping will not be applied to these results.
Type • Lookup • Table • None		We can do the dependency mapping mechanism in two ways: • **Lookup** – Assumes that you have a Data Source for all countries. e.g. one PX for the product cost for different products and values are defined by a label of the Mapping Source field. So the relationship is defined within the Data Source. • **Table** – Assumes that you have multiple Data Sources for all dependencies. Each dependency has its own data source for a specified dependency mapping mechanism. When you use this type of search, you should also change the value of sourceTable in PriceSettingConfig: the value should include a placeholder that will be swapped with your dependency property defined by the dependency mapping mechanism. The placeholder has this format: <<DependencyPreference>>. In this type of value, the Mapping Source field is ignored because you do not need to filter results inside the data source. Table dependency does not work for competitors because Pricefx has only 1 PCOMP table. An example of usage: • Dependency Level Name: Germany • Preference1 (from Dependency Configuration): DE • Dependency Field: Preference1 • Source Type: PX • Source Table (from PriceSettingConfig): Product Costs <<DependencyPreference>> With this data, the dependency mapping mechanism will search for a PX named Product Costs DE and then perform the lookup. • **None** – To use a Data Source with no dependency info.
Mapping Source Field	{name of the column with the dependency information in the source data}	The label should be taken from the table defined in the lookup on the row from Source Table with the "Lookup" type. It filters results this way: Filter.equal("${Source Field}", "${valueRetrieviedFromCondition}") It is important to double-check if this field was set up correctly, as most data lookups in the package utilize `api.stream()` calls and they will return nothing if you request a field that does not exist on a target object type. For

		example, "Country" for type PCOMP will not return anything even though a given product has competition data, because the name of the field is "country".

PriceSettingDimensions PP

Column Name	Values	Description
Dimension Type	• Products • Custom	The dimension value will be taken from: • Products: Product master data • Custom: Custom value provided by Groovy code
Order	• 1 • 2 • 3 • 4 • 5	Hierarchy order where "1" is the most important attribute, the following ones are used for more specific lookups.
Feature Name	• Fallback • StrategySelection • BaseStrategySelection • MinMargin • CostPlus • PriceIncrease • AdditionalDiscount • RelevantCompetitors • DependencyAdjustment • VolumeBreakdown • AdjustedPriceCorridor • ListPriceCorridor • CostSelection • Discount	Name of the affected feature for which PP will be generated with given keys. Only "Fallback" is mandatory, features with no specified keys will use "Fallback".
Dimension	• Products: {name of the column from the products table} • Custom: {name of the custom value provided in Groovy code}	

Example:



Custom Dimension

In hierarchy lookup, it is not only the Product attribute that can be used as the lookup dimension. A calculated value by Groovy can also be used as a dimension. What is available:

- `CustomDimensionManager` that can be used to provide the custom dimension value in Groovy code.

```
libs.PriceBuilderCommonElementUtils
        .CustomDimensionManager
        .getInstance()
        .addCustomDimensionValue("CustomDimensionName", "Custom
Dimension Value")
```

- `Custom` dimension type that can be used to configure the lookup dimension. The custom dimension value is provided by `CustomDimensionManager`. If the custom dimension value is not provided by the manager, the `Universal` wildcard will be used instead.

For more details see How To Set up Custom Dimension Lookup.

ExchangeRates PP

Column Name	Values	Description
From	{code of the base currency matching the DependencyConfiguration data}	e.g. "PLN"
To	{code of the target currency matching the DependencyConfiguration data}	e.g. "EUR"
ValidDate	{from which date the currency is valid}	In case of multiple entries with the same currency, the first entry after the valid date is chosen.
Rate	{exchange rate value}	e.g. "5.05"

Example:

Price Parameter Values : ExchangeRates [5]

From	To	ValidDate	Rate
VIR	EUR	28/08/2019	215
VIR	PLN	13/08/2019	84.15
VIR	PLN	27/08/2019	86
PLN	EUR	28/08/2019	4.1
EUR	PLN	28/08/2019	0.25

CompetitionAdditionalConfig PP

Column name	Values	Description
Name	{Setting Name}	To learn about the built-in pricing engines, see Calculation Engines.
Mode	{Setting Value}	

⚠ This is just a sample strategy. You can make as many copies of this PP as you want. You insert the used PP name in a PP called "StrategyDefinition". You can even skip this step if you do not use Competition Based Pricing. Configuring other calculation engines will work in a similar fashion.

Example:

Price Parameter Values : CompetitionAdditionalConfig [5]

Name	Mode
Competitor Position	min
Repositioning %	
Force Margin Check	no
Repositioning Abs	+10
Price Position	

StrategyDefinition PP

Colum Name	Values	Description
Strategy Name	{user friendly name of the strategy}	e.g. "CostPlus"
Level	• Independent • Dependent	Determines if the strategy can be calculated at independent/dependent level. If you need it for both, then create two entries with different values here.
Calculation Engine	• {name of the predefined engine} or • {path to the function in groovyLibrary which will perform calculations} To learn about the built-in pricing engines, see Price Strategies.	Path should look like this: "libs.MyLib.MyElement.myFunction"
Additional Engine Configuration	{name of PP with the engines customization}	Not all engines are customizable, so this field is nullable.
StrategyCalculationParameters	{list of parameter names which are sent to calculation}	Default parameters are: • SKU • TARGET_DATE • PRODUCT_COST • BASE_PRICE • LOOKUP_KEYS • PLUS_FOR_PRODUCT • COMPETITION_PRICES • MINIMUM_MARGIN_PRICE • PRICE_INCREASE • DEPENDENCY_INFORMATION_VALUES

		• BOM_LIST • DISCOUNTS If additional parameters are necessary, they need to be added from the code to the CalculatedPrices element in the "additionalParameters" map. ℹ The parameters are passed as an input to engines, so the order is important and should not be changed for default engines.
Parent Level Only	• Yes • No	If Yes, parent level results of this strategy will not be shown on dependent level PG/PL.
Parent Level Priority	• Yes • No	If Yes, this strategy will be above the dependent level results on a dependent level PG/PL.
Overridable	• Yes • No	If No and this strategy is the most important for the product represented by the given lookup keys, it is not possible to override the price through exception tables and manual overrides, regardless of exception configuration.
Mandatory Strategy	• Yes • No	Default value: No If yes, calculate price for this strategy even if stop flag found in (Base)StrategySelection

Example:

Price Parameter Values : StrategyDefinition [18]

Strategy Name	Level	Calculation Engine	Additional Engine Configuration	Strategy Calculation Parameters	Independent Level Only	Independent Level Priority
RRP	Independent	LookupEngine	RRPLookupEngineConfig	SKU,TARGET_DATE,COUNTRY_INFORMATION_VALUES		
RRP	Dependent	LookupEngine	RRPLookupEngineConfig	SKU,TARGET_DATE,COUNTRY_INFORMATION_VALUES		
PriceIncrease	Independent	AdjustmentEngine	PriceIncreaseAdditionalConfig	BASE_PRICE,PRICE_INCREASE		
PriceIncrease	Dependent	AdjustmentEngine	PriceIncreaseAdditionalConfig	BASE_PRICE,PRICE_INCREASE		
MinCompetition	Independent	CompetitionEngine	MinCompetitionAdditionalConfig	COMPETITION_PRICES		
MinCompetition	Dependent	CompetitionEngine	MinCompetitionAdditionalConfig	COMPETITION_PRICES		
MaxCompetition	Independent	CompetitionEngine	MaxCompetitionAdditionalConfig	COMPETITION_PRICES		

VolumeBreakdownExceptions PP

For additional details see also Volume Breakdown.

Column Name	Values	Description
SKU	{sku from Product master}	e.g. "MB-0001"
Dependent Level Name	{name of dependency level} as stated in DependencyConfiguration PP	e.g. "Global"
Volume #01... #15	Value representing the product volume that will use the given discount	Values here represent at what volumes the discount should start to be applied, e.g.

		• #01 = 5 • #02 = 10 • #03 = 15 Technically, we use these values just for showing them on a price list and applying a discount, so if prices from this package are used by some quoting solution, it has to apply a proper filtering based on the quoted volume. If one of these values is missing, its volume-discount pair will be ignored.
Discount #01... #15	Discount for the given volume	E.g. 15% These discounts apply to ranges of volumes defined in the Volume columns. If one of these values is missing, its volume-discount pair will be ignored.

Example:

Price Parameter Values : VolumeBreakdownExceptions [1]

SKU	Dependency Level Name	Volume 1	Discount 1	Volume 2	Discount 2	Volume 3	Discount 3
MB-0007	Global	5	10.00 %	15	20.00 %	25	30.00 %

WarningConfig PP

For additional details, see also Warning Handling.

Column Name	Values	Description
Key1	{error code}	Code of the warning, e.g. "NO_DIMENSIONS". See their full list.
Key2	• * • {DependencyLevelName}	Defines which dependency this configuration is for. There should be a fallback for each error code by leaving an entry with an asterisk. This field may be ignored if every dependency shares same warning handling.
Message	Any text	Message displayed to the user describing what went wrong, e.g. "Price parameter called 'PriceSettingDimensions' does not exist or is empty."
Solution	Any text	Solution displayed to the user describing how to fix the issue. It is shown only in a ResultMatrix.
Type	Any text	Type of the issue displayed to the user. It is shown only in a ResultMatrix, e.g. "Business issue".

Alert	• Empty • "Message" • "Yellow" • "Red" • "Critical"	Defines how the warning will be classified: on the element level or with an increasing severity. We advise against using the critical alert too often, as it overrides the color for the whole item. It does not quite fit with the corridor configuration from the price checks module.
Matrix	• Yes • No	Defines if the warning will be registered in a ResultMatrix. Default value: Yes
Warning	• Yes • No	Defines if the warning will be registered in the default Warnings column. Default value: Yes

Example:

Price Parameter Values : WarningConfig [58]

Key1	Key2	Message	Solution	Type	Alert	Matrix
CANT_GET_REASON	*	CANT_GET_REASON		Other	Red	Yes
CANT_READ_DATA_FOR...	*	CANT_READ_DATA_FOR...		Error	Red	Yes
CANT_READ_DISCOUNT	*	CANT_READ_DISCOUNT		Runtime	Red	Yes
DEPENDENCY_ADJUST...	*	DEPENDENCY_ADJUST...		Data		Yes
ERROR_GETTING_INDE...	*	ERROR_GETTING_INDE...		Error		Yes
ERROR_LOOKING_UP_S...	*	ERROR_LOOKING_UP_S...		Error	Red	Yes
ERROR_LOOKING_UP_C...	*	ERROR_LOOKING_UP_C...		Error	Red	Yes
ERROR_LOOKING_UP_D...	*	ERROR_LOOKING_UP_D...		Error	Red	Yes
ERROR_LOOKING_UP_...	*	ERROR_LOOKING_UP_...		Error	Red	Yes
ERROR_LOOKING_UP_V...	*	ERROR_LOOKING_UP_V...		Error	Red	Yes
ERROR_LOOKING_UP_R...	*	ERROR_LOOKING_UP_R...		Error	Red	Yes
ERROR_PARSING_VOLU...	*	ERROR_PARSING_VOLU...		Error	Red	Yes
EXCEPTION_IGNORED	*	EXCEPTION_IGNORED		Other	Red	Yes
EXCEPTION_STRATEGY...	*	Exception overwritten	Do you really want to overr...	Business Warning	Yellow	Yes
NO_CORRIDOR_CONFIG	*	NO_CORRIDOR_CONFIG		Error		Yes
NO_COST_ENTRY_IN_C...	*	NO_COST_ENTRY_IN_C...	Check Data	Data		Yes
NO_DEPENDENCY_LEV...	*	Missing Dependency Level...	Define Dependency Level ...	Configuration	Critical	Yes
NO_DEPENDENCY_ADJ...	*	NO_DEPENDENCY_ADJ...		Data	Red	Yes
NO_DISCOUNT_VALUE	*	NO_DISCOUNT_VALUE	Check Discount Data	Data	Red	Yes
NO_EXCHANGE_RATE	*	NO_EXCHANGE_RATE		Data	Red	Yes
NO_FINAL_PRICE	*	NO_FINAL_PRICE		Error	Critical	Yes

PriceSettingLevel PP

See also Adjustments after Changing the PP PriceSettingLevel.

Column Name	Values	Description
Pricelist	{name of dependency level} as stated in DependencyConfiguration PP	
Price level	• Gross • Gross / Net	

		If an additional discount from a gross to net price should be calculated for a specific dependency level.

Example:

Elements Documentation

All significant elements are already documented as groovyDocs: https://gitlab.pricefx.eu/accelerators/price-builder-accelerator

> ⚠ Be aware of the Pricefx field called "Manual Override". It is an out-of-the-box field which is not related to Accelerators and should not be used.

Warning Handling

You can set up this Accelerator in many different ways. Almost every feature is configurable either by specifying where to look for necessary data or by telling the package how to handle this data. Because so many things can be adjusted, there is a lot of things that can go wrong as well. Missing configurations, missing data or conflicting business configurations are only a few examples.

To handle this we introduced a warning/error handling mechanism further described at Error Handling Deep Dive.

In this section:

- Warning Placement
- Warning Configuration
- Warning Codes List

Warning Placement

Warnings can be displayed at the following places:

- Default warning column
- Field where the issue happened (hidden if the element is internal)
- ResultMatrix at the end of the calculation

There are also some issues which prevent WarningManager from being initialized. Calculation will fail and an exception will be thrown in this case.

Warning Configuration

Every warning code signalizes that something could not be executed properly; it is not always a business issue. There are cases like "EXCEPTION_IGNORED" which just inform the user that a table override was ignored due to a manual override. It is possible to configure the visibility of each warning.

For every error, the following behavior can be configured:

- Raising any type of alert (only when the element is visible).
- Adding a default warning.
- Adding to a matrix element at the end of the calculation.

For details on setup see WarningConfig PP.

The recommended approach on initial package deployment is to keep the default error setup until the package was fully configured and tested. Then all warnings related to unused features can be turned off by disabling alerts and hiding them from the default warnings popup. They can be also hidden from the custom Matrix popup, but we don't recommend it since it may make debugging future problems much harder.

Warning Codes List

The table below represents a full list of expected errors and their default configuration that can be found in WarningConfig PP after package deployment:

Error Code	Message	Solution	Type	Alert	Matrix	Warning
CANT_APPLY_STRATEGY_CONDITIONS	The strategy condition(s) cannot be applied. Details are in technical message.	Verify the strategy conditions configuration table and its data.	Configuration	Red	Yes	Yes
CANT_CALCULATE_COST_TYPE	Unable to calculate cost for the specified type. Details are in technical message.	Verify the cost type configuration.	Configuration	Red	Yes	Yes
CANT_CALCULATE_MARGIN_BREAKEVEN_VOLUME	Unable to perform break-even calculation due to division by zero error.	Adjust the final price and cost.	Runtime	Red	Yes	Yes
CANT_CALCULATE_REVENUE_BREAKEVEN_VOLUME	Unable to perform break-even calculation due to division by zero error.	Adjust the final price.	Runtime	Red	Yes	Yes
CANT_GET_ACTUAL_PRICE_FROM_PL	Unable to get the actual price from price lists. No valid list found.	Make sure the price list with the same calculation logic as the current used has already existed and been approved.	Configuration		Yes	Yes

CANT_GET_CORRIDOR_CONFIG	Unable to get the corridor configuration.	Verify the corridor configuration tables setting.	Error	Red	Yes	Yes
CANT_GET_COST_TYPES_SELECTION	Unable to get the cost types selection configuration. Unable to calculate related element(s).	Verify cost types selection configuration table and its data.	Configuration	Critical	Yes	Yes
CANT_GET_DEPENDENCY_LEVEL_ADJUSTMENT	Unable to get the dependency adjustment value.	Verify dependency adjustment configuration table and its data.	Error	Red	Yes	Yes
CANT_GET_DISCOUNT	Unable to get the discount value.	Verify discount level configuration table.	Runtime	Red	Yes	Yes
CANT_GET_MIN_MARGIN	Unable to get the minimum margin.	Verify minimum margin configuration table and its data.	Error	Red	Yes	Yes
CANT_GET_PRODUCT_CHANGED_CAUSE	Unable to get the reason for change in the product.	Contact support.	Other	Red	Yes	Yes
CANT_GET_RELEVANT_COMPETITION_DATA	Unable to get the relevant competition data.	Verify relevant competition data configuration table and its data.	Error	Red	Yes	Yes
CANT_GET_STRATEGIES_SELECTION	Unable to get the strategies selection configuration. Unable to calculate related element(s).	Verify strategy selection configuration table and its data.	Error		Yes	Yes
CIRCULAR_DEPENDENCY	DependencyLevelConfiguration has circular dependency.		Runtime	Red	Yes	Yes

DEPENDENCY_LEVEL_ADJUSTMENT_IS_ZERO	There is no markup between Parent Price and this Pricing Level configured.	Check if you want to directly follow the Parent Level Price without any Markup.	Data		Yes	Yes
EMPTY_CONFIG	One of the mandatory fields in the configuration is empty. Details are in technical message.	Verify the configuration table and its data.	Fix config	Critical	Yes	Yes
ERROR_LOOKING_UP_STRATEGY_CONDITIONS_DATA_PP	Unable to get Strategy Condition Price Parameter.	Verify configuration of strategy conditions.	Configuration	Red	Yes	Yes
ERROR_PARSING_COST_TYPE_DEFINITION	Unable to parse the definition of the specified cost type. Details are in technical message.	Verify the cost type configuration.	Configuration	Red	Yes	Yes
EXCEPTION_IGNORED	An exception value has been ignored.	Verify if the exception is valid and its value is valid.	Other	Red	Yes	Yes
EXCEPTION_STRATEGY_OVERRIDDEN	Strategy exception has been overridden by manual selection.		Business Warning	Yellow	Yes	Yes
FIELD_NOT_FOUND	The specified field is not found in the target table. Details are in technical message.	Verify the field name in the configuration.	Runtime	Red	Yes	Yes
INVALID_DIMENSION_LOOKUP_FIELD	The specified lookup keys in the current dimension are not found.	Verify the lookup keys of the current dimension in the pricing	AnyType	Red	Yes	Yes

		dimension configuration table.				
INVALID_FINAL_PRICE	The calculated final price value is invalid.	Resolve critical and serious errors.	Error	Critical	Yes	Yes
INVALID_FORECAST_TYPE	The selected forecast type is not supported.	Verify current quarter Forecast setting.	Error	Red	Yes	Yes
INVALID_PARENT_SOURCE_ID	The source ID of the specified parent price list to be referred is invalid.	Verify the Source ID in dependency configuration table. It should be the referred price list ID.	Data	Red	Yes	Yes
INVALID_MIN_MARGIN_PERCENT	The minimum margin value is invalid.	Verify minimum margin configuration table data.	Data		Yes	No
CANT_GET_STOCK_CONFIG	Unable to get the stock configuration.	Verify Stock setting.	Runtime	Red	Yes	Yes
INVALID_VOLUME_BREAKDOWN_SETTING	Error in parsing volume breakdown due to invalid setting.	Verify volume breakdown configuration table and its data format, syntax.	Configuration	Red	Yes	Yes
MISSING_INPUTS_MARGIN_BREAKEVEN_VOLUME	Missing input (s) for margin breakeven volume. Required: cost, base price, volume, final price.	Verify the required parameter(s).	Runtime	Red	Yes	Yes
MISSING_INPUTS_REVENUE_BREAKEVEN_VOLUME	Missing input (s) for revenue break-even volume. Required: base price, volume, final price.	Verify the required parameter(s).	Runtime	Red	Yes	Yes

NO_ACTUAL_LIST_PRICE_FOUND	No valid actual list price found. Unable to calculate related element(s).	Verify Actual Price setting and its data.	Data	Yellow	Yes	No
NO_BASE_DEPENDENCY	There is no dependency for looking for non virtual dependency level.		Runtime	Red	Yes	Yes
NO_CONFIG	One of the mandatory configurations is not found. Details are in technical message.	Verify the configuration table and its data.	Fix config	Critical	Yes	Yes
NO_CORRIDOR_CONFIG_FOUND	No valid corridor configuration found.	Verify corridor configuration tables data.	Error		Yes	No
NO_COST_FOUND	No valid cost value found. Unable to calculate related element(s).	Verify Cost setting and its data.	Data		Yes	Yes
NO_DEPENDENCY_LEVEL_ADJUSTMENT	Unable to get data for dependency level adjustment.	Check dependency level adjustments data source.	Data	Red	Yes	Yes
NO_DISCOUNT_FOUND	No valid discount value found. Unable to calculate related element(s).	Verify discount level configuration table and its data.	Data	Red	Yes	Yes
NO_EXCHANGE_RATE_FOR_BATCHED_ITEM	One or more of the items in the batch has transaction data with not convertible currency.	Verify the exchange rate configuration and the data.	Runtime	Critical	Yes	Yes
	No valid exchange	Verify exchange rate	Data	Critical	Yes	Yes

NO_EXCHANGE_RATE_FOUND	rate value found.	configuration table and its data.				
NO_PARENT_LEVEL_CALCULATED_PRICES	No valid calculated prices from the parent level found.	Verify Prices of the parent item.	Error		Yes	Yes
NO_PARENT_LEVEL_FINAL_PRICE	No valid final price from the parent level found. Unable to calculate related element(s).	Verify Final Price of the parent item.	Error	Red	Yes	Yes
NO_PARENT_LEVEL_PRICE	No valid price from the parent level found. Unable to calculate related element(s).	Verify Final Price of the parent item.	Error	Red	Yes	Yes
NO_PARENT_LEVEL_PRICE_DECISION	No valid price decision from the parent level found.	Verify Price Decision of the parent item.	Error	Red	Yes	Yes
NO_PARENT_LEVEL_RECORD_FOUND	No price record found for parent level.	Verify the corresponding price record at the parent level source.	Error	Yellow	Yes	Yes
NO_INPUT_FOR_CORRIDOR	Unable to calculate the price corridor due to missing parameter(s).	Verify the required parameter(s).	Error		Yes	No
NO_INPUT_FOR_DISCOUNT	Unable to get product discount data.	Check Discount Data.	Data		Yes	Yes
NO_INPUT_FOR_PARENT_LEVEL_ADJUSTED_PRICE	Unable to get Parent level adjusted price.	Check Parent Level Price and Dependency Level Adjustment.	Error		Yes	No
NO_INPUT_FOR_PARENT_LEVEL_PRICE_PRIORITY	Unable to get Parent Level Price Priority.	Check configuration.	Error		Yes	No

NO_INPUT_FOR_MARGIN	Unable to get margin.	Check value for margin in Price Parameter.	Data		Yes	Yes
NO_INPUT_FOR_MIN_MARGIN_PRICE	Unable to calculate the minimum margin price due to missing parameter(s).	Verify the required parameter(s).	Data		Yes	No
NO_INPUT_FOR_MIN_MARGIN_VALIDATION	Unable to validate the minimum margin due to missing parameter(s).	Verify the required parameter(s).	Data	Red	Yes	Yes
NO_INPUT_FOR_NET_PRICE	Unable to calculate net price.	Check configuration of net price module.	Configuration		Yes	Yes
NO_INPUT_FOR_PRICE_CHANGE_EFFECT	Unable to calculate the effect of price changing due to missing parameter(s).	Verify the required parameter(s).	AnyType	Red	Yes	Yes
NO_INPUT_FOR_STOCK_COVER_DAYS	Unable to calculate stock cover days due to missing parameter(s).	Verify stock and sales volume forecast elements result data.	Runtime	Yellow	Yes	Yes
CANT_GET_VOLUME_DISCOUNT	Unable to get volume discount.	Check data for volume discount.	Data		Yes	Yes
NO_MIN_MARGIN_CONFIG_FOUND	No valid minimum margin configuration found.	Verify minimum margin configuration table data.	Data		Yes	No
NO_MIN_MARGIN_PRICE	Unable to calculate minimum margin price.	Check if everything is available for minimum margin price.	Data		Yes	No
NO_PF_TARGET	Missing Price Flexibility Package target.	Contact support.	Configuration		No	No

NO_PRODUCT_CHANGED_CAUSE	Missing reason in Price Flexibility Package.	Contact support.	Other		Yes	Yes
NO_ROUNDING_RULE_FOUND	There is no suitable rounding rule found.	Verify the rounding rules configuration table and its data.	Configuration	Red	Yes	Yes
NO_SALES_VOLUME_FORECAST	There is no forecast for sales volume.	Verify the transaction data.	Data		Yes	No
NO_SALES_VOLUME_LAST_PERIOD	There is no sales volume in last period.	Verify the transaction data.	Data		Yes	No
NO_SALES_VOLUME_LAST_YEAR	There is no sales volume in last year.	Verify the transaction data.	Data		Yes	No
NO_SALES_VOLUME_YTD	There is no sales volume up-to-date.	Verify the transaction data.	Data		Yes	No
NO_STOCK_DATA	There is no valid record for stock.	Verify Stock setting and its data.	Runtime	Yellow	Yes	Yes
NO_STRATEGY_DEFINITION_FOUND	The strategy definition cannot be found.	Verify strategy definition configuration table and its data.	Configuration	Red	Yes	Yes
NO_SUITABLE_ROUNDING_RESULT	Applying the rounding rule makes the price invalid. The rule has been canceled.	Verify the price and the rounding rules configuration.	Configuration	Yellow	Yes	Yes
NO_TRANSACTION_SOURCE_TABLE_FOUND	The specified transaction table cannot be found.	Verify the Transaction Source setting.	Error	Red	Yes	Yes
NO_TURNOVER_FORECAST	There is no forecast for the turnover.	Verify the transaction data.	Data		Yes	No
NO_TURNOVER_LAST_PERIOD	There is no turnover in last period.	Verify the transaction data.	Data		Yes	No
			Data		Yes	No

NO_TURNOVER_LAST_YEAR	There is no turnover in last year.	Verify the transaction data.				
NO_TURNOVER_YTD	There is no turnover up-to-date.	Verify the transaction data.	Data		Yes	No
PP_VALIDITY_PERIODS_INVALID_FORMAT	Validity periods of the product are in invalid format.	Check Data and Configuration of PP. Validity period fields should have Date value type.	Runtime	Red	Yes	Yes
STRATEGY_OVERRIDE_PROHIBITED	Strategy selections are not available due to overridable setting.	If this is not the expected behavior, verify the Overridable property of member strategies definition.	Other		Yes	Yes
TOO_MANY_ROWS	There are too many records in the current batch.		Runtime	Yellow	Yes	Yes
TOO_MANY_ROWS_ABORTED	Unable to proceed because there are too many records in the current batch.		Runtime	Yellow	Yes	Yes
TOO_SMALL_MARGIN	The current margin is smaller than the specified minimum margin.	Verify the final price and the cost.	Business Warning	Yellow	Yes	Yes
UNABLE_TO_READ_TABLE_DATA	Data lookup could not been performed / results could not be read.		Runtime	Red	Yes	Yes
UNEXPECTED_ERROR	Unhandled error.	Contact support.	Error	Red	Yes	Yes
UNEXPECTED_PRICE_RANGE_FOR_CORRIDOR	Unexpected range for price corridor.	Check the configured corridor for price checks.	Data	Red	Yes	Yes

UNSUPPORTED_ACTUAL_PRICE_SOURCE_TYPE	The specified source type for actual price is unsupported.	Verify the source type of Actual Price setting.	Other	Red	Yes	Yes
UNSUPPORTED_PARENT_SOURCE_TYPE	The source type of the specified parent price list to be referred is unsupported.	Verify the Source Type in dependency configuration table.	Configuration	Red	Yes	Yes
VALIDITY_PERIODS_OVERLAPPED	Validity periods overlap.	Check validity period columns data.	Data	Red	Yes	Yes

Error Handling Deep Dive

Error handling is structured. All errors should be expected and they are all defined in Warning Handling with instructions what to do when such an error happens.

Limitations

Not everything can be covered by warnings due to technical reasons – the warning manager itself needs some data to be initialized. There are two groups of deployment issues which are not configurable – they raise an exception which is not caught:

1. Missing libraries required for Price Setting Package to run:
   a. PriceBuilderCommonElementUtils
   b. PriceListManagement
   c. SharedLib
2. Missing company parameters:
   a. PriceSettingConfig
   b. DependencyConfiguration
   c. DependencyMappingConfig
   d. WarningConfig

Unexpected Errors

These are also two steps for "default" exceptions in case something unexpected happens:

1. "UNEXPECTED_ERROR" entry in the WarningConfig PP. This is manageable by users but we strongly recommend to set it to "Critical, Yes, Yes" and report every occurrence of such an error. When this error is used, it has a modified message to help track what went wrong.
2. Undefined behavior is specified in the code and it has the following code: "NO_ERROR_DEFINED".

Abortable Errors

There is a group of errors that tells us that there is no point in doing any further calculations. We would not be able to get any valid results either way, so we abort the calculation instead of passing on wrong /missing data/configurations. The calculation is aborted on a module-level, so only elements in the module that raised one of these errors are skipped.

The list of abortable error types:

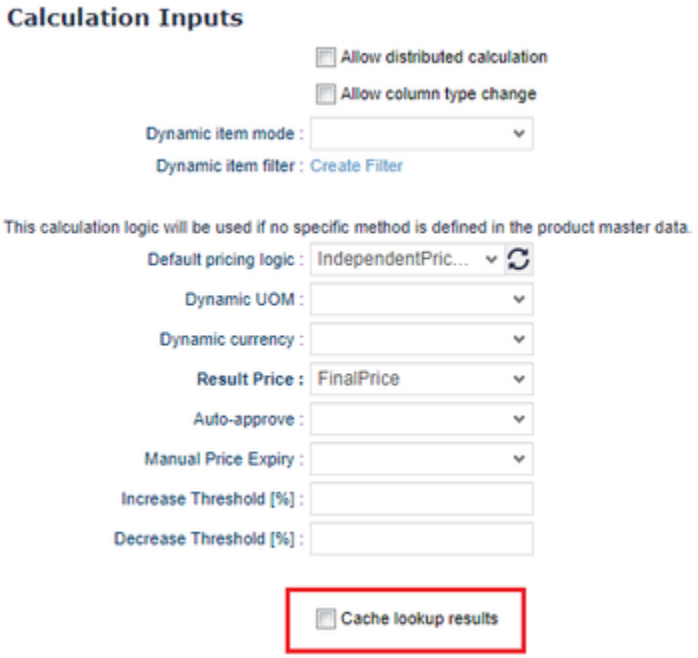
- VALIDATION
- MODULE_UNUSABLE
- NO_CONFIG
- EMPTY_CONFIG

Caching Lookup Results

The calculation logic contains multiple lookups from both raw user data (like Datamarts, Data Sources or Product Extensions tables) and calculation specific configuration in Price Parameters. Some of it is cached by default (e.g. sales and forecast data or configuration from Price Parameters), but some of is not. Especially configuration tables that are split based on "dimensions" which are described in Product Segmentation per Feature.

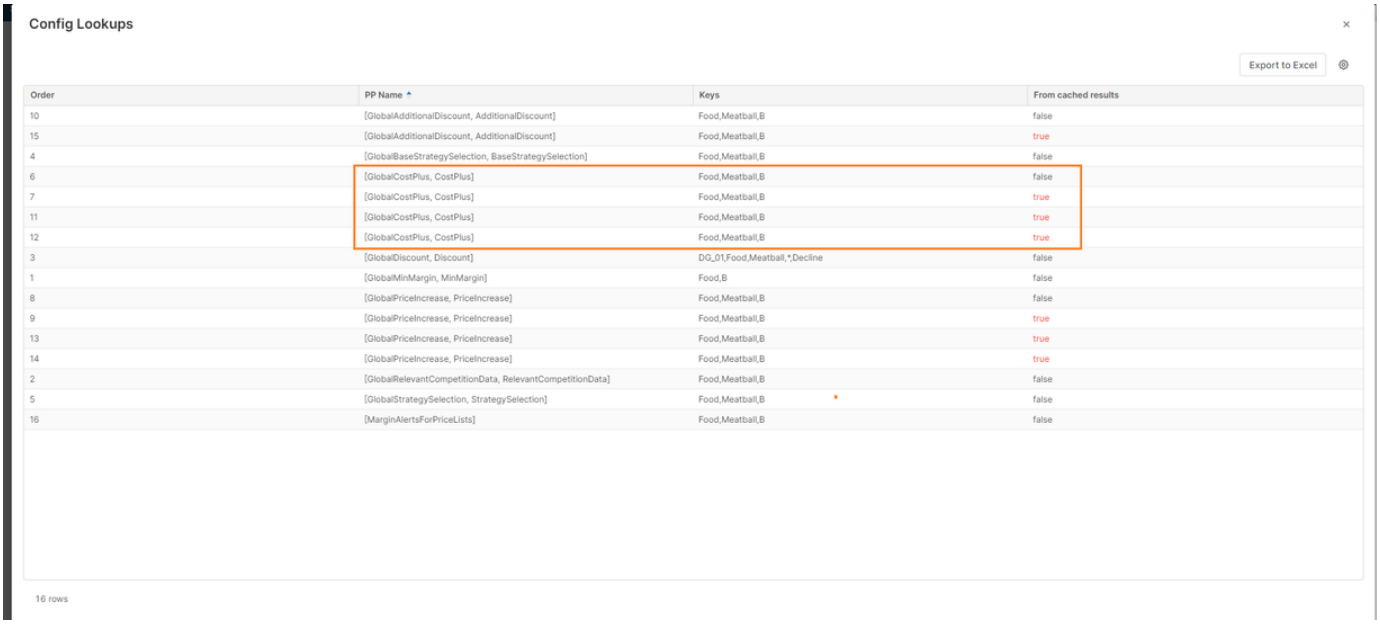
It depends on the granularity of the product segmentation and the number of products whether caching such a configuration helps the logic execution times.

To address this, you can use this option in Calculation Inputs:



By default it is switched off since it may impact the performance negatively (particularly in cases where the segmentation is big). This should be the first thing to check when looking for optimization.

In price list logics, in the "Config Lookups" element you can see multiple lookups of the same information with different features.



Batching

All lookups are batched as described in Data Lookups.

Price Setting Configuration Wizard - Technical Design

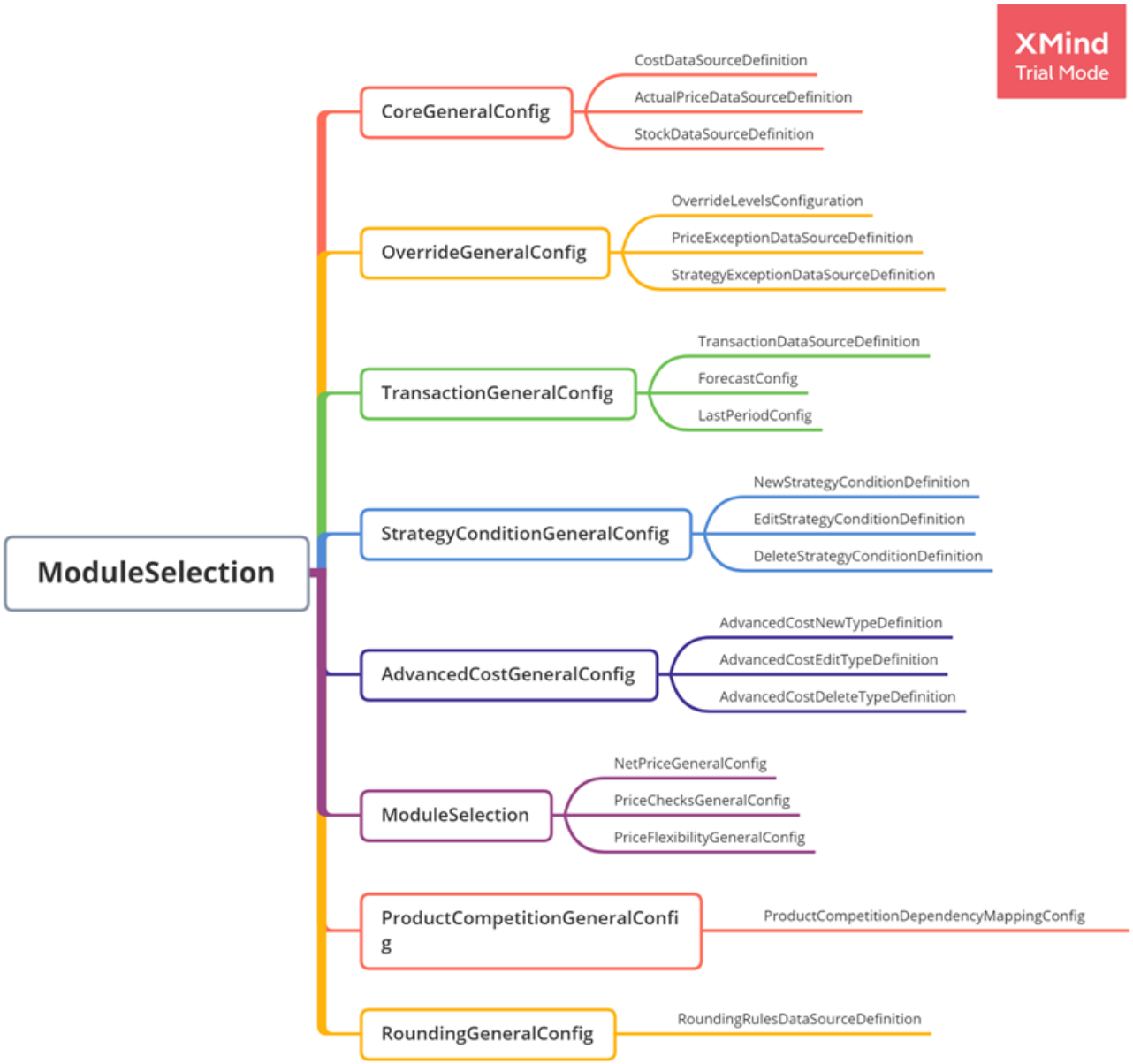
⚠ This document is for maintenance and upgradability. Do not publish it to users.

The business description of the Price Setting Configuration Wizard can be found at Price Setting Package Configuration Wizard.

In this section:

- Screen Map
- Components
- General Calculation Flow
- PSP_ConfigWizardScreenFactory Library Structure
- PSP_ConfigWizardCommonLib Library Structure

Screen Map



Components

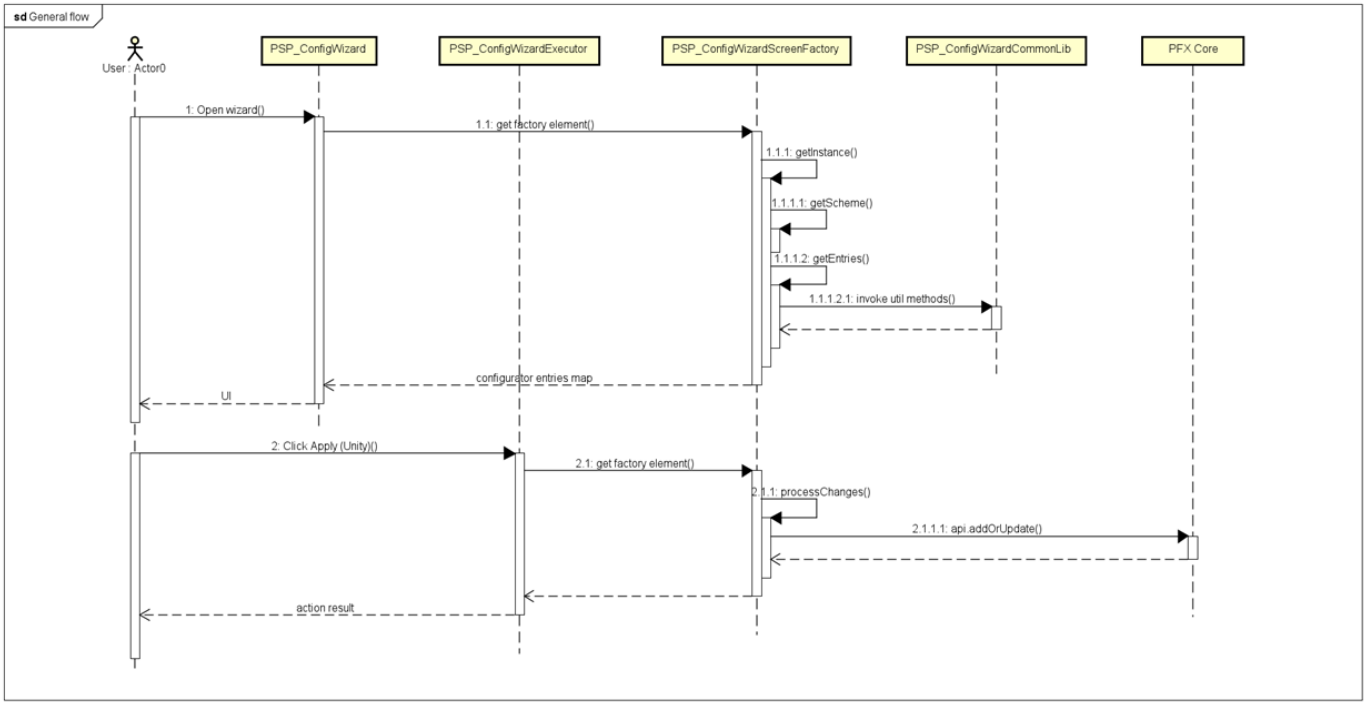
Generic logics

- PSP_ConfigWizard
- PSP_ConfigWizardExecutor

Libraries

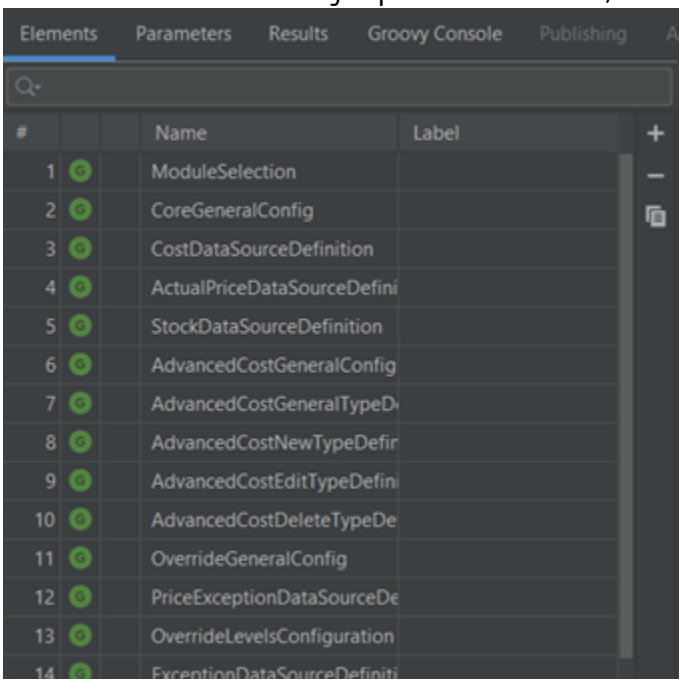
- PSP_ConfigWizardScreenFactory
- PSP_ConfigWizardCommonLib

General Calculation Flow



PSP_ConfigWizardScreenFactory Library Structure

Each element in the library represents a screen, with proper naming.



Syntax to get a factory element: Script screen = libs.PSP_ConfigWizardScreenFactory[currentScreen]

Syntax to get a screen instance: libs.PSP_ConfigWizardScreenFactory[currentScreen].getInstance()

Syntax to get a screen DB update handler: libs.PSP_ConfigWizardScreenFactory[currentScreen].
processChanges()

Element Structure

Must have these public methods:

- Map getInstance() - To produce configurator entries set for the screen from a predefined scheme and handled businesses.
- void processChanges() - To update values to DB (if any).

Scheme Format

Must have _currentInstanceName hidden configurator entry, with fixed value as the current screen name.

Must have current screen instance-id hidden configurator entry. This is to determine whether the screen is a first-time run.

Each input has a properties map. Supported properties:

- **type** - Typically InputType.X, but can also be null. Null means that the result after rendering is a text line, not an input.
- **label** - String, the input label.
- **defaultValue** - Object, the value of the input on the first time run.
- **value** - Object, fixed value.
- **readOnly** - Boolean, disables the input from entering a value.
- **required** - Boolean, the user has to set the input to a valid value to proceed with the wizard.
- **valueOptions** - List, the options of a dropdown list input or a radio button input.
- **valueLabels** - Map, the labels of the options of a dropdown list input or a radio button input.
- **value labels structure** - [option value 1: option label 1, option value 2: option label 2, ...]
- **noRefresh** - Boolean, to prevent logic from rerun when an input value has changed.
- **message** - String, to show message / HTML string instead of a regular input.

Scheme:

```
Map getScheme() {
    String idInputName = libs.PSP_ConfigWizardCommonLib.SchemeUtilities.getInstanceIDInputName(SCREEN_NAME)
    Script descriptions = libs.PSP_ConfigWizardCommonLib.HTMLDescriptions

    return [_currentInstanceName: [type: InputType.HIDDEN, value: SCREEN_NAME],
            (idInputName)       : [type: InputType.HIDDEN],
            description         : [message: descriptions.MODULE_SELECTION],
            currentConfig       : [message: getCurrentSettingsParagraph()],
            moduleSelector      : [type: InputType.OPTION, label: descriptions.SELECT_MODULE_INPUT, noRefresh: true],
            configButton        : [type: InputType.BUTTON, label: "Configure selected module"]]
}
```

Example

The ModuleSelection element structure is as follows:

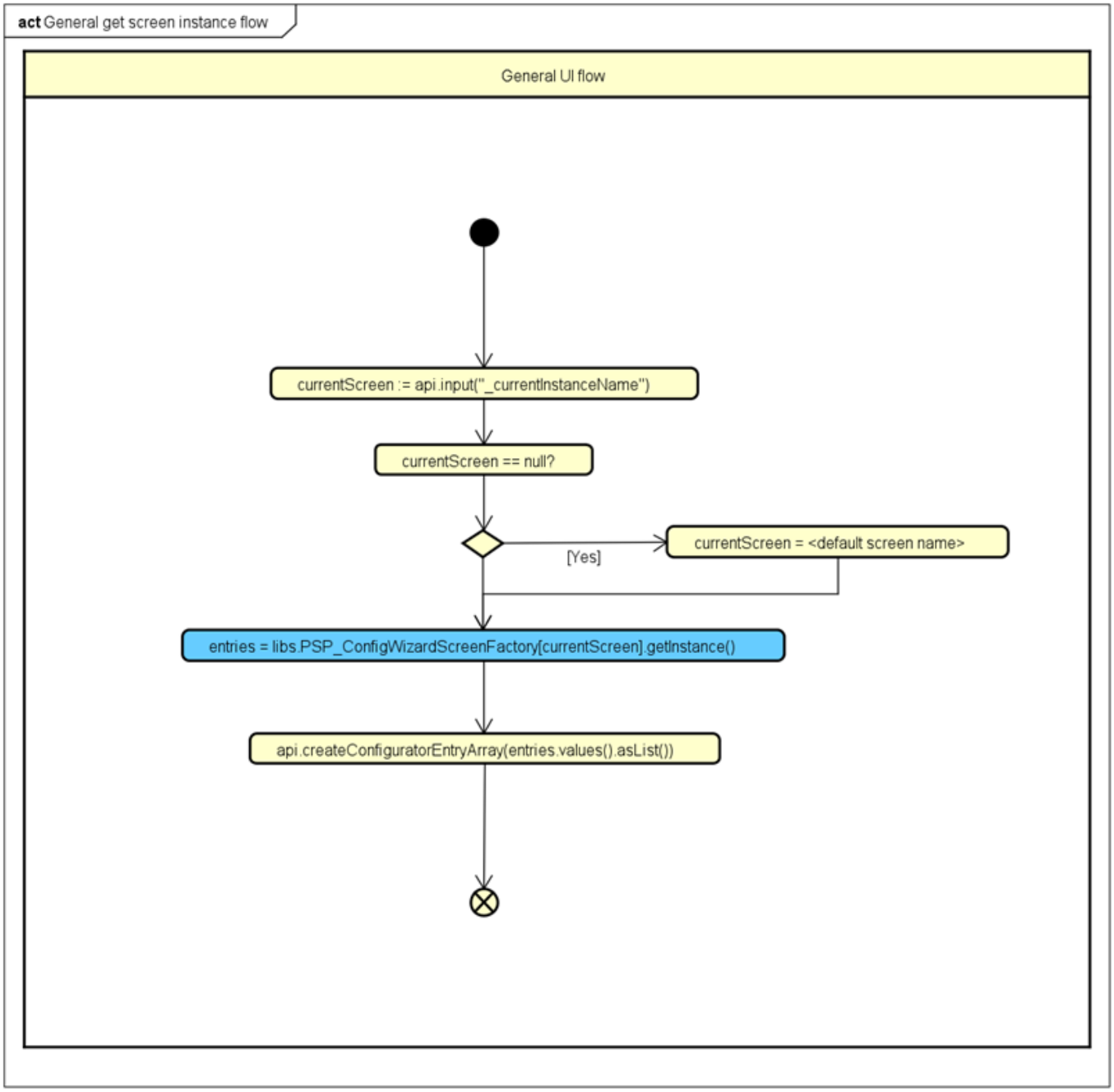
```
@Field String SCREEN_NAME = "ModuleSelection"
@Field String CORE_MODULE_NAME = "coreModule"
@Field String CORE_MODULE_LABEL = "Core Elements"

// produce the configurator entries set
Map getInstance() {...}

void processChanges() {// the function is for the executor logic interface implementation}

// for the modules table html
protected Map getModuleMapping() {...}

// for the modules table html
protected String getCurrentSettingsParagraph() {...}

// for the modules table html
protected String getStatusRowsDefinition() {...}

// for the modules table html
protected String getStatusLabel(def moduleStatus) {...}

// the predefined screen scheme
protected Map getScheme() {...}

// implementation of producing configurator entries set
protected Closure getEntries() {...}

// handle navigation button / get target screen entries
protected Closure handleButtonEvents() {...}
```

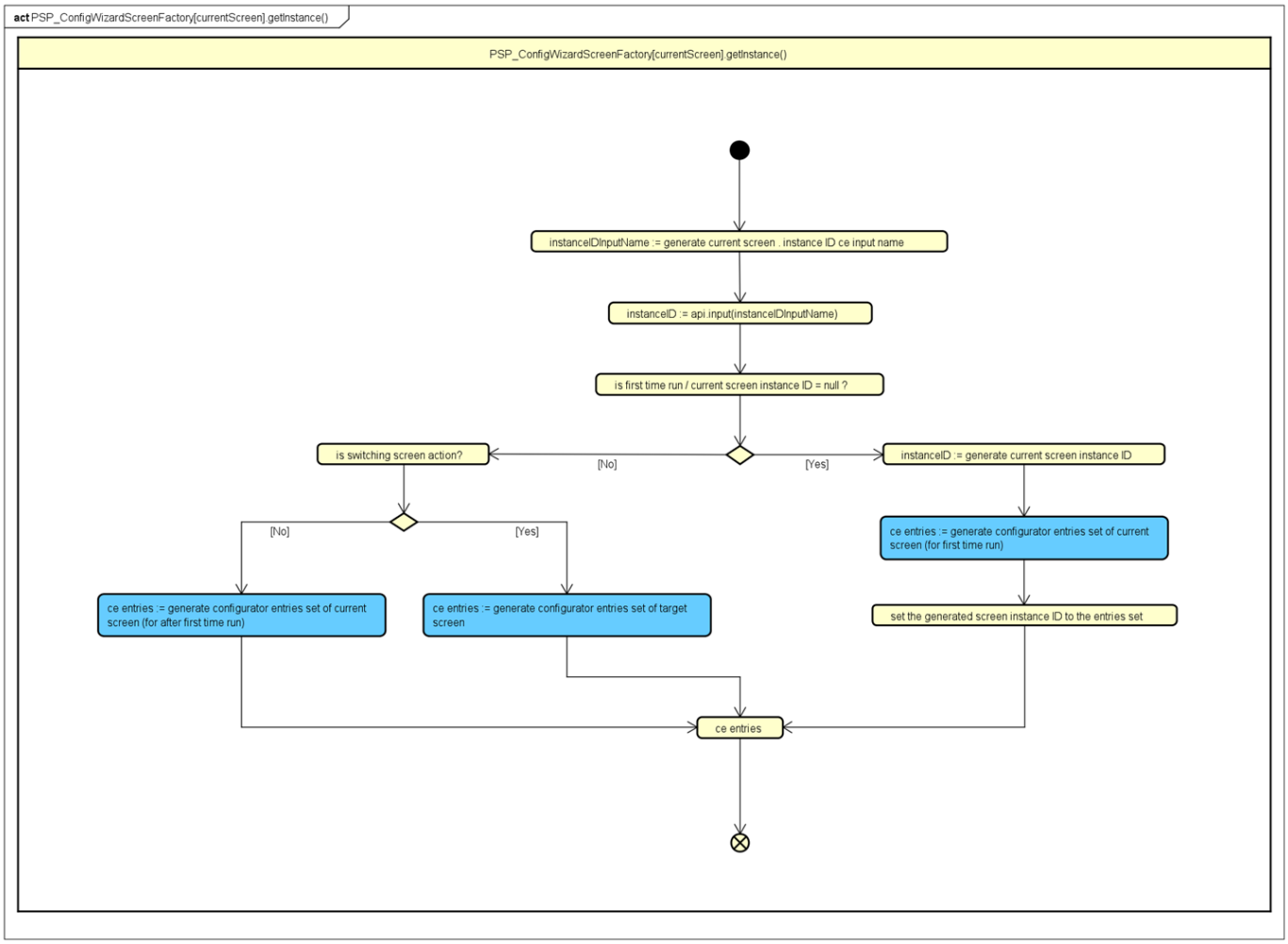
PSP_ConfigWizardCommonLib Library Structure

A set of predefined utilities which consists of:

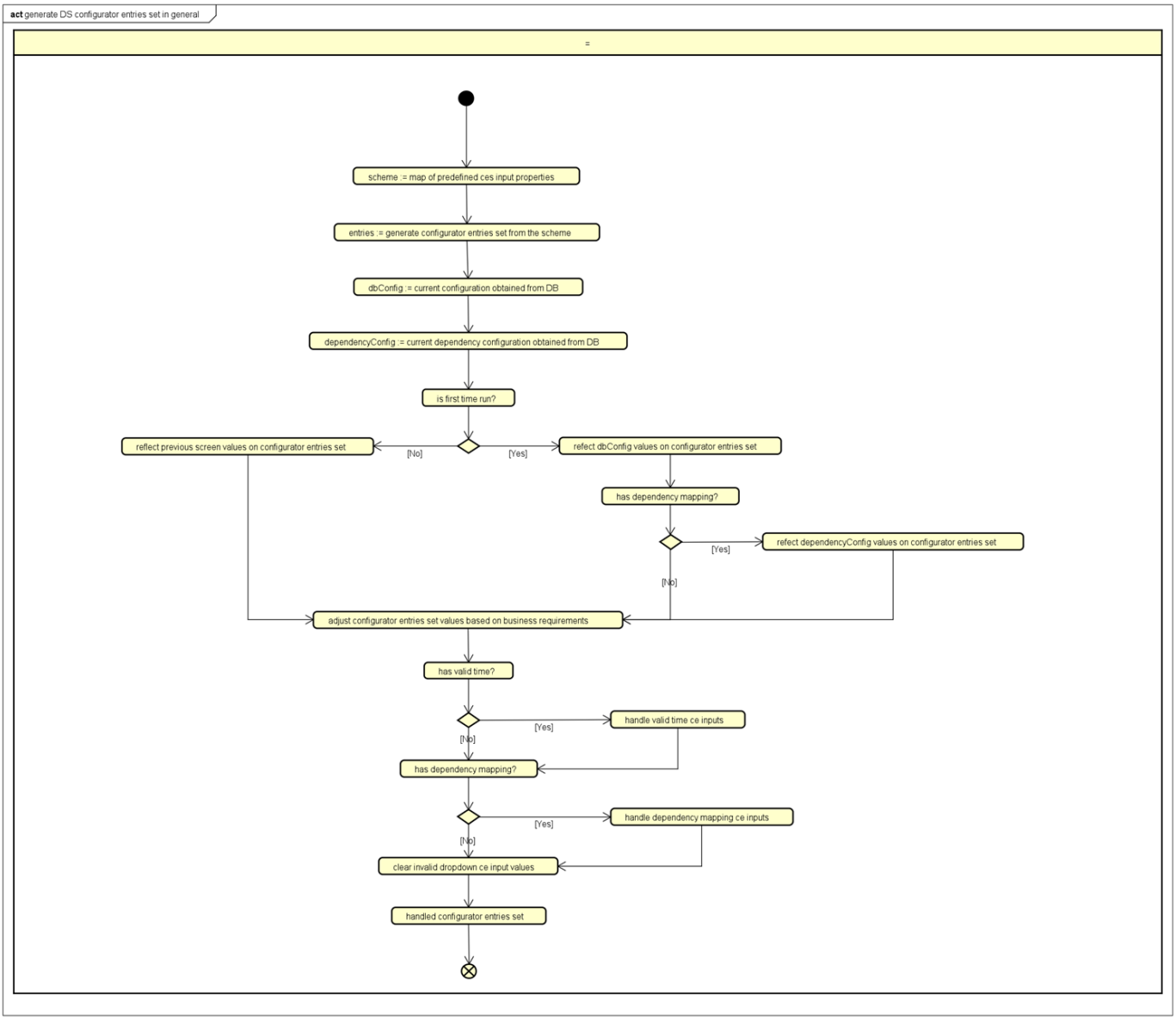
- **Constants** - Contains constants used throughout the wizard such as common input labels,...
- **HTMLDescriptions** - Contains HTML constants used throughout the wizard.
- **HTMLDescriptionUtilities** - Contains HTML handler methods.
- **Errors** - Contains error handler methods (temporarily unused).
- **ScreenStateUtilities** - Contains configurator entry state modification methods, such as set a CE input value, properties,...
- **InputBuilder** - Contains configurator entry builder methods, to build a CE from a predefined scheme.
- **DataUtilities** - Contains database-related methods, such as fetching configuration data rows,...
- **SchemeUtilities** - Contains methods for common scheme manipulation/parsing, with business handling.
- **ModuleUtilities** - Contains methods for common module-related operations, with business handling.
- **NavigationUtilities** - Contains methods for common stuff related to handling screen navigation.
- **InputUtilities** - Contains common methods that provide data for a dropdown input, validate an input value, handle common business operations,... Typically they are for inputs that have dynamic data /properties based on the user interaction (eg. checkbox selected by user unlocks some input fields).

Screen Instance Flow


act General get screen instance flow
General UI flow
currentScreen := api.input("_currentInstanceName")
currentScreen == null?
[Yes]
currentScreen = <default screen name>
entries = libs.PSP_ConfigWizardScreenFactory[currentScreen].getInstance()
api.createConfiguratorEntryArray(entries.values().asList())



The following chart is for a typical data source configuration screen. Depending on businesses, it may get different between screens.



Logic Element Names for Approval Workflow (Price Setting)

Accelerate Approval Workflow Package (AWP) can be used with Price Setting package.

To learn what elements can be used in workflow step conditions see Element Names.

Module Dependencies

Configuration

A module might require other modules so that it can work properly. The dependencies can be set during module development in the `ModuleManager.MODULES_DEFINITION` by the below configurations:

```
stock             : [ELEMENTS                                  : ["Stock", "StockCoverDays"],
                     REQUIRED_MODULES                          : ["transaction"],
                     REQUIRED_MODULES_NOT_INSTALLED_ERROR_CODE: "STOCK_REQUIRED_MODULES_NOT_INSTALLED",
                     REQUIRED_MODULES_ABORTED_ERROR_CODE       : "STOCK_REQUIRED_MODULES_ABORTED",
                     ABORTED_STRING                            : "Stock module has failed",
                     ABORTED_ERROR_CODE                        : "STOCK_MODULE_ERROR"]
```

- `REQUIRED_MODULES` – Defines a list of modules that this module requires to work properly.
- `REQUIRED_MODULES_NOT_INSTALLED_ERROR_CODE` – If one of the required modules is not installed, an error will be thrown.
- `REQUIRED_MODULES_ABORTED_ERROR_CODE` – If one of the required modules is aborted, an error will be thrown.

Out-of-the-box Module Dependencies

Module Name	Required Modules
stock	transaction

Glossary (Price Setting)

This glossary summarizes various terms used in Price Setting Accelerator.

Core Terms

Term	Description
Pricing Levels Price Setting Levels Pricing Structure Dependency Structure Dependency Configuration	You can calculate product prices on more levels, e.g. 1) Global 2) Regions 3) Countries 4) Shop. The prices on lower levels can depend on prices from higher levels, i.e. the lower level price can be derived from the higher level price. Prices for various Pricing Levels are then calculated in separate price grids/lists. The various pricing levels and their dependencies are also referred to as Price Setting Levels, Pricing Structure, Dependency Structure, or Dependency Configuration. The word “Dependency” is sometimes used in the meaning of “Level”, e.g. in the term “Dependency Mapping”.
Product Segmentation Segmented Product Portfolio Product Attribute Hierarchy	The various pricing factors (e.g. margin uplift in the Cost+ method) can depend not only on the Pricing Levels, but also on a combination of several product attributes, e.g. Industry, Business Unit, Product Group. Those product attributes are not necessarily hierarchical, but for the definition of the pricing factors it is very useful (and needed) to have them in certain order.

	A certain combination of values of those product attributes is known as a product segment. The product segment can be either very narrow (e.g. products with industry A, business unit B, product group C), but also more broad (e.g. products with industry A, business unit “any”, product group “any”).
Dependency Mapping	Technique how to find data relevant for certain Pricing Level (Dependency). Data for different Levels can be stored either in separate tables, for example: a separate table for each country (this is called “Table”), or together in a single table where one field is dedicated to distinguishing which row belongs to each Level (this is called “Lookup”), or together in a single table without any differentiation to which level each row belong (this is called “None”).

Additional Terms

Terms	Description
Actual Price Lookup	Lookup for the current product price which is fetched based on the configuration.
Adjusted Price Corridor	Deviation between the global adjusted price and the final list price
Allow Manual Price Override	Defines whether the ManualPrice and ManualPriceReason elements should be visible.
Allow Manual Strategy Override	Defines whether the PriceSelector element should be visible.
Base Currency	Currency of the price list/grid.
Base Strategies	Ordered list of base strategies that precede normal price strategies. Base Strategies are usually defined on a more generic level to have some basic pricing rules across the complete product portfolio. They are used, for example, to make promotions override the standard strategy configuration.
Calculated Prices	Calculated prices for every defined strategy.
Changes From Monitor	If isPLCMTarget is true, it contains ResultMatrix describing to the user why this element was added to the price grid. Currently supported only for Global LPGs.
Competition Data	Data about competitors’ pricing, read from the PCOMP table.
Cost	Cost of the product, read from the PX configured in the “PriceSettingConfig” PP. If not found, null is returned.
Country	Country for which the current calculation runs. If Global mode is used, the country will be “Global”. If Headquarter mode is used, a value configured

	in the "PriceSettingConfig" PP will be used. Otherwise, the country has to be selected from a drop-down list in a given PL/LPG configuration. A list of available countries will be created dynamically based on the "CountryInformation" PP.
Country Adjustment	Percentage by which global prices will be multiplied to obtain local prices.
Country Lookup	Row from the "CountryInformation" PP.
Discount	Percentage discount applied when calculating the transition from a gross to net price.
Discount Group	Discount group for a specific product in a specific country.
Exceptions	Message with exceptions (if any). The value will be: Price Exception: <value> Strategy Exception: <value> Price Exception: <value> (+1) – if both exceptions are present
Exceptions Manager	Interface used for managing exceptions / manual overrides and making them allowed. Exceptions/Allowance logic is implemented here.
Exchange Rate	Exchange rate between global and current local currencies.
Final List Price	Value copied from used list price. This is the price before applying a discount.
Final Price	Used list price in case of a gross calculation; net price in case of a gross /net calculation.
Forecast Data	Cached map with forecast data for every product. It is calculated based on the PP configuration.
Global Adjusted Price	Final price from the global price list, adjusted with the country adjustment.
Global Calculated Item	Calculation result for the product on the global level.
Global Currency	Currency of the headquarter country or global price list. The data is loaded from the "CountryInformation" PP.
Global Decision	Decision extracted from a Global Calculated Item. It can be a strategy name, an exception table type message or a manual reason.
Global Level Adjusted Prices	Prices from the global price list, adjusted with the country adjustment.
Global Min Price	Minimum price read from a PX checked against final price. (coming soon)
Global Price	Price from the global calculation result extracted from a Global Calculated Item, after applying exchange rates.
Last Year Sales Volume	Volume from last year.

Last Year Turnover	Turnover from last year.
List Price Corridor	Deviation between the global price and final list price
Lookup Keys	Keys in the format Map<String, List<String>> where the key is a dimension and a List is a sorted list of keys (dimensions) for hierarchical lookups. Currently, only the Product dimension is supported.
Manual Price	Empty field acting as an input field for the user. Fill it with an override price if necessary. Its visibility depends on the configuration in Exceptions Manager. This value pops up automatically if there is an ExceptionsTable price exception.
Manual Price Reason	Empty field acting as an input field for the user. Fill it with a reason why a price has been overridden. Its visibility depends on the configuration in ExceptionsManager. If there is an ExceptionsTable price exception, the value is "Price Exception".
Margin	Margin calculated based on the final price and product cost.
MinimumMargin	Minimum margin for a specific product checked against "Margin".
Net Price	Used list price with a discount applied.
Net Price Level	Indicates if a net price will be calculated.
OverrideRemover	Used for clearing manual/table overrides/exceptions when the default strategy does not allow it.
Pop Up Data	Calculates margin values for a popup data input.
Price Decision	Reason for a given price. ManualPriceReason if the price was overridden; a pricing strategy from Price Selector otherwise. If there was an ExceptionsTable exception, the value is "ExceptionTable - <exception type>".
Prices	PopUpData formatted into a user friendly MatrixResult.
Price Selector	Combo box with calculated prices and their strategies. Users can override a default strategy here.
Prices Summary	Generates data for a popup and price selector. It also collects prices from the global price list and applies a country adjustment to it.
Price Strategies	Ordered list of possible pricing strategies for the current product, read from the "StrategySelection" PP.
Sales Volume Forecast	Volume forecast for the upcoming period with values taken from the specific strategy set on the PriceSettingConfig PP.
Sales Volume YTD	Volume from this year.
Strategy Selection Entry	Lookup result from the "StrategySelection" PP. It is only in the country price list because only there more than one element can utilize this result.

Transaction Data	Cached map with transaction data for every product. It is split into the last year and year to date results.
Turnover Forecast	Turnover forecast for the upcoming period with values taken from the specific strategy set on the PriceSettingConfig PP.
Turnover YTD	Turnover from this year.
Used List Price	Value selected in PriceSelector or inserted in ManualPrice.
Warnings	User friendly table showing all warnings which occurred during the calculation.

Price Setting Package 2.3.0

This document summarizes fixes introduced in the Accelerate Price Setting Package release version.

Version	2.3.0
Release Date	Jul 9, 2024

In this document:

Upgrade Notes

- Default dependency mapping has been disabled; i.e. in the DependencyMappingConfig PP the **Type** is now set to *None* for new installations. However, dependency columns in the data are kept as unused and old configurations are still valid. (PFPCS-5322)
- Custom Engines: Removed the distinction between static and dynamic parameters in `AdditionalCalculatorParameters`; all parameters must now be closures for dynamic execution. (PFPCS-7297)
- The internal components of the wizard have been reworked, so if you have a customized wizard, it will not work after you upgrade. (PFPCS-7854)
- Description has been added to most of the tables. This is just QoL change, however it is too big to be stated as Manual Change.

Manual Changes

After finishing the upgrade:

- Go to Price Parameters **PriceSettingConfig** and update keys from "**Independent** Manual Override Allowance" to "**Parent** Manual Override Allowance".
- Go to Price Parameters **PriceSettingConfig** and update keys:

Old key	New key
Independent Manual Override Allowance	Parent Manual Override Allowance

- Price Parameter for **WarningConfig** contains many changes in this release, mostly due to Configurable Lookups. If you want to keep user friendly errors and avoid "UNEXPECTED_ERROR" code, this is the new configuration: 

- Update the attribute of the **StrategyDefinition** Price Parameter based on the structure described at StrategyDefinition PP.

Attribute	Name	Label
attribute2	ParentLevelOnly	Parent Level Only
attribute3	ParentLevelPriority	Parent Level Priority

- Update attribute's name of the **StrategySelection** Price Parameter:

Attribute	Name	Label
attribute29	Prioritize Parent Level Price	Prioritize Parent Level Price

- If you have added a custom parameter in the price list logic, check it and update based on Add Custom Parameter to Strategy Call.
- There is a new feature "Configurable Lookups". Go to Price Parameters and create **LookupConfiguration** with JSON type.
  - It needs to have these columns:
    - key1Field
    - checksConfig
    - customFilter
    - selectedFields
    - userSourceName
    - userSourceType
    - isUsingDependencyLevelHierarchy
    - sortByField
    - key2Field
    - key1Transformation
    - key2Transformation
  - Entries of this PP can be filled by going through each PSP wizard. For more complex implementations, follow Configurable Lookups Guide.

- Standard config of previous versions of PSP is reflected by this file

- Some of **PriceSettingConfig** configuration has been moved to LookupConfiguration – now it can be removed from the original table:
  - `Cost,*` can now be removed
  - `Price Exception,*` can now be removed
  - `Strategy Exception,*` can now be removed
  - `Rounding Rules,*` can now be removed
  - `Stock,*` can now be removed
  - `Actual Price,*` has changes
    - If value of attribute2 was "PX", now it should be renamed to ConfigurableLookup.
    - Every value after attribute2 can be cleared.
- **CostTypeDefinition PP** – Part of the Advanced Cost module has been changed. Now it points to the LookupConfiguration table where lookup is defined. New columns are:
  - attribute1: `CalculationEngineSuffix`
  - attribute2: `Type`
  - attribute3: `CalculationMethod`
  - attribute4: `LookupName`
  - attribute5: `DependencyField`
  - attribute6: `DependencyType`
  - attribute7: `MappingSourceField`

New Features and Improvements

New Feature Description	ID
Default dependency mapping has been disabled; i.e. in the DependencyMappingConfig PP the **Type** is now set to *None* for new installations.	PFPCS-5322
Configurable Lookups: There is now a lookup framework that allows you to define PX/PP/PCOMP lookups in a Price Parameter table, enabling you to modify existing lookups without any Groovy coding and add new ones with minimal Groovy required.	PFPCS-5968

Strategy Designer configuration has been moved to a separate repository and now functions as a dependent package to PSP.	PFPCS-6941
Custom Engines: Removed the distinction between static and dynamic parameters in AdditionalCalculatorParameters; all parameters must now be closures for dynamic execution.	PFPCS-7297
To make PSP work well with EngineCalulator, the calculatePrice method of engines returns now a Map (instead of BigDecimal).	PFPCS-7462
STRATEGY_RESULTS key has been added as an optional parameter for the Engine calculator. It is a list of maps where each map represents a value returned from the calculate() function. This key allows for composite pricing strategies and optional final price selection based on previous strategy results. Additionally, it introduces the option to reverse the order of strategy calculations, ensuring the most relevant strategy is considered.	PFPCS-7499 PFPCS-7808
The internal components of the wizard have been reworked, so if you have a customized wizard, it will not work after you upgrade. To be able to proceed with the upgrade, you need to agree that the wizard will override your custom changes.	PFPCS-7854
Tooltips have been added to configuration PPs to provide guidance to users.	PFPCS-7922
In the PriceSettingDimensions Price Parameter table, the following columns have been renamed to reflect their contents better: Field Name column => Dimension Dimension column => Dimension Type	PFPCS-7924
Implemented a new runtime fallback for StrategySelection to handle missing */*/* keys and products without defined strategies. Improved error handling and notifications for scenarios with no strategies or final prices, avoiding issues during bootstrapping.	PFPCS-8029
In PSP documentation, a new article on setting up a Custom Dimension Lookup has been added.	PFPCS-8032
The term "Independent" has changed to "Parent" in: DependencyPriceListLogic Price Parameters tables Configuration wizard	PFPCS-8105 PFPCS-8106 PFPCS-8107

Fixed Issues

Bug Description	ID
PSP deployment fails when CPQ is already deployed and has global preferences on the Product Cost table.	PFPCS-7923
	PFPCS-8088

There is a “zombie column” when the number of secondary keys decreases during bootstrapping.	
Fallbacks in hierarchic lookups do not work.	PFPCS-8292
When you open the wizard, select Core Module and click configuration, you get an error.	PFPCS-8346