Accelerate Price Setting Package

Version 2.1.11

July 2023

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.

- Product Info (Price Setting)
- 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)

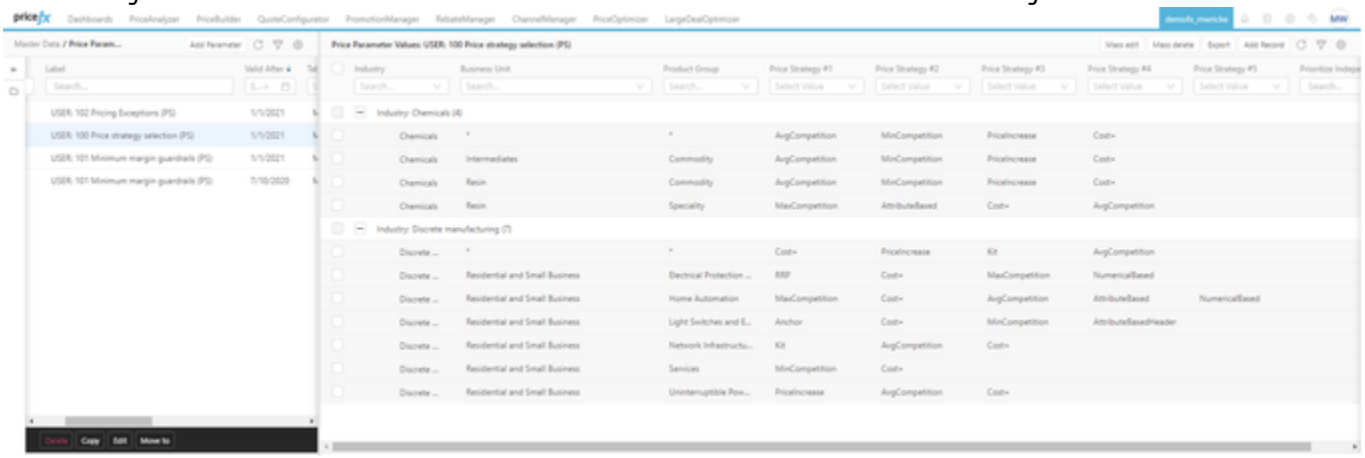
Product Info (Price Setting)

Price Setting Package (in close cooperation with Price Flexibility Package) provides a price setting and management framework powered by eight distinct pricing strategies that pricing managers can use to create list prices, applying different strategies to different parts of their product portfolio as appropriate. Price lists can be easily assigned to customers within their markets.

This package includes:

- Gross & net price lists
- Standalone or independent and dependent price lists (e.g., national & state or global & country, etc.)
- Dynamic pricing (e.g., automated price changes based on an underlying product or competitor changes)
- Eight distinct pricing strategies like attribute-based pricing or competitive based pricing
- Approval workflow through the Approval Workflow Package

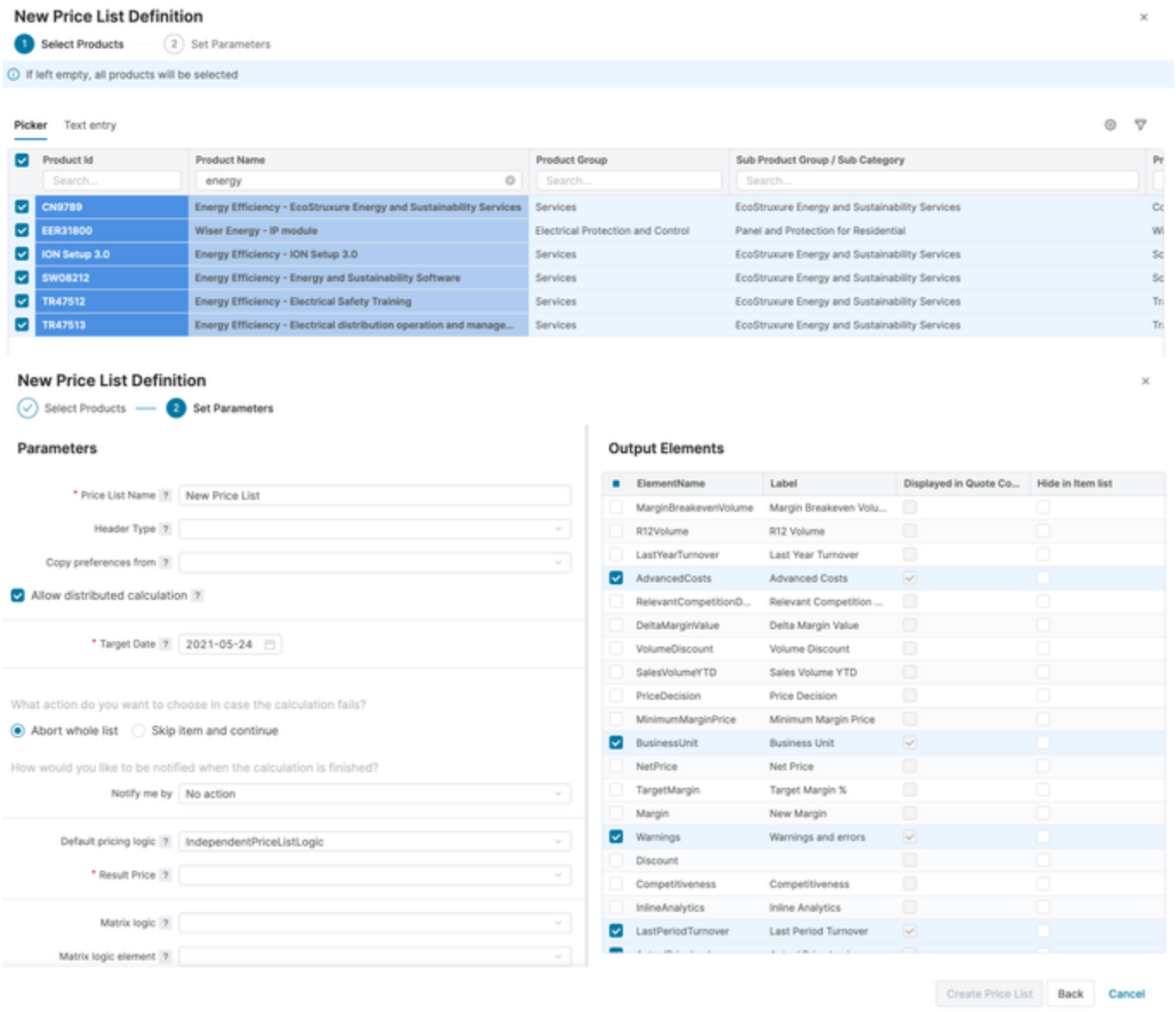
With this package pricing managers get a framework that allows them to create new list prices quickly, which they can then communicate to their sales teams or customers immediately.



Included Price Strategies

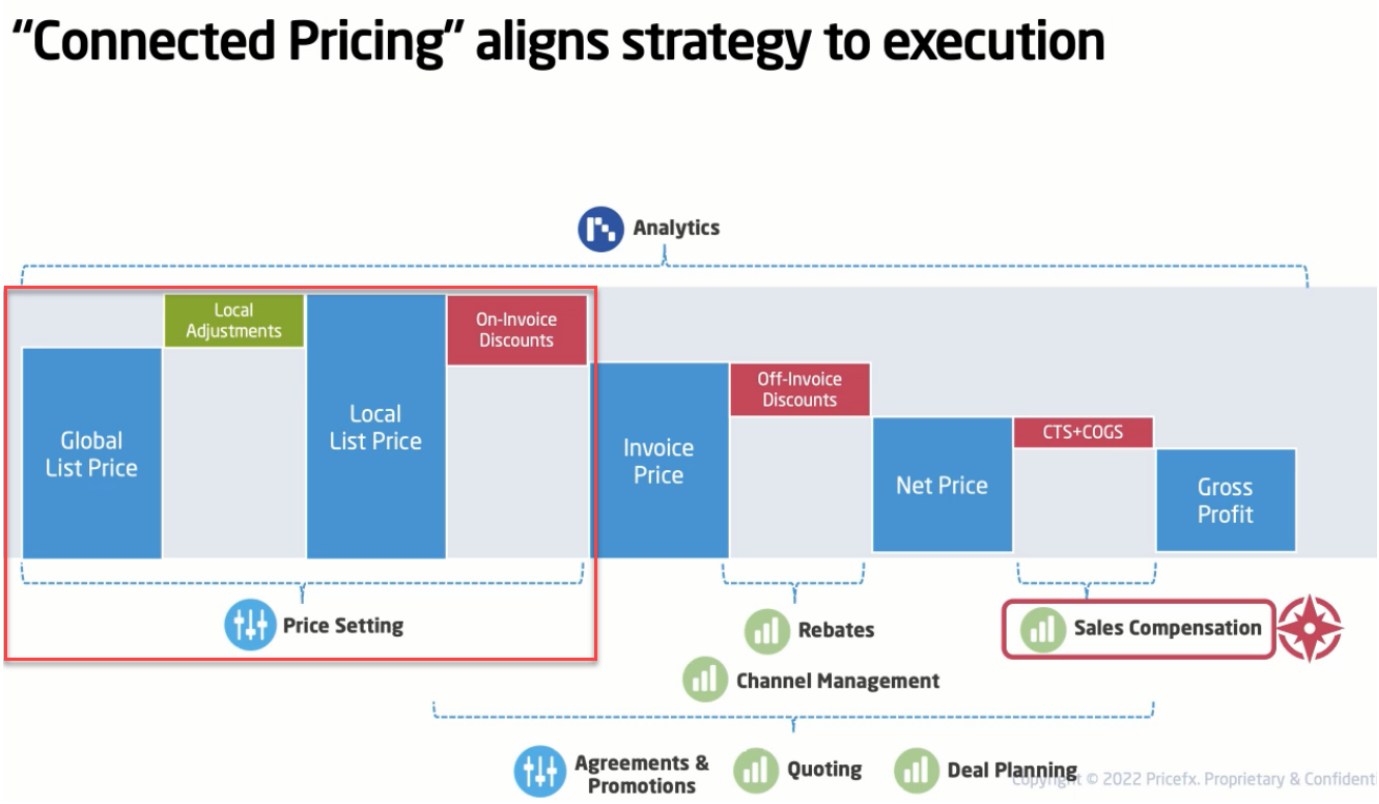
This package brings you up to speed by providing the following price strategies out of the box:

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



Overview (Price Setting)

The Price Setting Accelerator is one of many pre-built solutions from Pricefx that when implemented will provide you with a means to generate global and local list prices and integrate adjustments and on-invoice discounts. This accelerator is designed to provide aggregation at the following points in the waterfall.



Within the design of the Pricefx PLAN/PRICE/PROFIT structure, we can see exactly where the Price Setting Accelerator will be included:

Business Overview (Price Setting)

Premise

You are involved with the generation of static or dynamic pricing for global or local price lists in support of your sales organization as part of a team from either Pricing, Financial, Sales, or Sales Ops within your organization.

Desired Outcome

Leverage comprehensive price settings / management options for price lists as well as dynamic price lists (Live Price Grids) with automated recalculation. Design your price lists with lightning speed and in various hierarchies (standard price list, global vs. country price lists, etc.).

Context and Background

All organizations need a strategy that provides a price setting and price administration framework that will provide list price information on their products to their quoting application for their sales force. This framework would focus on accuracy, consistency, efficiency, dependability, and responsiveness to provide a single source of truth for product pricing.

Problem

The processes for the creation and generation of price lists are too cumbersome and inefficient and do not allow the organization to quickly react to pricing opportunities in the market. Additionally, the pricing strategies utilized are inflexible to market changes and the approval process for price changes is too time-consuming.

Solution Capabilities

Once this Accelerator has been implemented and linked to your application, then these components will be available for immediate use:

- Ability to implement 8 different pricing strategies
- Utilize product segmentation to define pricing decisions
- Perform pricing calculations at dependent or independent levels
- Illustrate summary of prices with Price Insights dashboard

Capabilities Summary (Price Setting)

The main purpose of the Price Setting accelerator is calculation and management of prices. To calculate prices, it uses Price Strategies/Rules.

Price Strategy/Rule

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.

To calculate prices, the accelerator uses Calculation Engines.

Out-of-the-box Price Strategies

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

For more details, see Price Strategies.

Custom Strategies

The accelerator lets you define your own strategy and use it in the calculation process.

Product Segmentation

You can set up a **hierarchy of dimensions** of your product master data that will be used to diversify your pricing strategy.

Using Product Segmentation, you can define all your pricing decisions on every level of granularity you need.

For example, you can define some high-level pricing strategy and specify special product segments on a more granular level where needed

Price Levels

You can calculate scenarios like:

- National & state prices
- Global & country prices
- Regional & country & channels prices

In these situations some prices are "derived" from other prices, this is what we call Price Levels.

You can define several pricing 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 level. You can think about them as "lower" levels of the dependency hierarchy.

Trade Levels

- **List Prices** (usually used in B2C) – Net Prices are discounted List Prices.
- **Gross & Net Prices** (usually in B2B) – List price and some discounted Net Price for your next trade level. Depending on a Discount Group, a corresponding discount is applied for calculation of the Net Prices.

Price Insights Dashboard

Provides summary of prices collected from a range of price lists and price grids.

- Compares prices of an SKU across various price lists, e.g., you can spot which price lists have empty prices.

- Compares price list prices and historical prices, e.g., if the price list prices are lower than the historical ones, you can consider an increase.
- Helps you easily detect errors in calculations or exceptional behavior.

Dynamic Pricing

You can achieve it with the help of another accelerator: Price Flexibility. It is not directly provided by this accelerator.

Approval Workflow

You can achieve it with the help of another accelerator: Approval Workflow. It is not directly provided by this accelerator.

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

Individual features of the accelerator are prepared as modules which can be turned on or off.

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.

Modules:

- Conditions – "Finetunes" your decision tree.
- Sales & forecast data – Displays historical data in the price list.
- Exceptions – Sets up custom price behavior and manual overrides.
- Price check – Checks whether the margin is within a suitable range.
- Round – Rounds prices to business friendly values.
- Cost calculations – Chooses among different types of cost calculations (single, average, sum).
- Volume breakdown – Provides different volume discounts, depending on the item quantity.

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)

- Use Cases (Price Setting)

- Business Introduction (Price Setting)

Use Cases (Price Setting)

The Price Setting Package can be used both for Price Lists and Live Price Grids (LPG). The use cases are based on either price levels (one price list or more dependent price lists) or trade levels (list/gross/net prices).

- Price Levels Scenarios
- Trade Levels Scenarios
- Technical Notes

Price Levels Scenarios

- **Standalone Price List** – Covers price setting for just one price list.
- **Independent Price List Dependent Price List** – Covers price setting on the independent/dependent levels. Independent price lists are a base for dependent price lists; this allows to create various pricing scenarios, for example: define an independent "global" price list and then define dependent "country" price lists that will have prices based on the "global" one.
- **Complex pricing level structure** – You can build a complex structure. You can define several **Independent Price Lists** calculating prices independently. You can arrange various **Dependent Price Lists** around them. You can also create a multi-level tree where a Dependent Price List depends on another Dependent Price List. With this you can have for example multiple Independent 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.

Trade Levels Scenarios

- **List Prices** – Prices are calculated as List Prices.
- **Gross List Price / Net Price** – First, prices are calculated as Gross List Prices. Depending on a Discount Group, a corresponding discount is applied for calculation of the Net Prices.

Technical Notes

- Also, the Price Flexibility Package is fully compatible to "monitor" independent LPGs.
- The groups can be defined in the DependencyConfiguration PP and selected during PL/LPG creation using the configurator.

Business Introduction (Price Setting)

The Price Setting Package helps you manage Price Lists and Live Price Grids through a variety of built-in tools. These include:

- Management of **independent and dependent levels** for price lists. You can define an independent "global" price list and then define dependent "country" price lists that will have prices based on the "global" one. The global country approach is only one possible scenario, you can also build any use case where you have different pricing levels with dependency connections between them.
- Support for various **trade levels**. This allows you to define whether prices are calculated as List Prices or as Gross List Prices and then, after discounts are applied, as Net Prices. Usually List Prices are built

in B2C environments where you can typically find Net Prices as discounted List Prices in B2B (B2C) environments where you may want to calculate some List Price and some discounted Net Price for your next trade level.

- The package employs several **price strategies** which you can choose from, such as calculations based on anchor, competition, BoM data or attribute-based pricing.
  - There are extra setup options for these strategies, 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** helps you define all your pricing decisions on every level of granularity. You can set up dimensions of your product master data that are used to diversify your pricing strategy. 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.
- The package can be easily customized through **modularization**. Individual features are prepared as modules which can be turned on or off. Each module can also be customized. The modules cover the following functionality:
  - Through **conditions** you can decide that some prices will be ignored or taken with lower priority. So you can „finetune" your decision tree for the correct proposed price or ensure general rules in your pricing.
  - You can have **sales and forecast data** displayed. They are usually taken from your sales transactions. You can also deliver SKU-aggregated data.
  - You can create custom price behavior by creating **exceptions** and allowing **manual override**
  - You can have an automatic **price check** whether the margin is within a suitable range.
  - You can **round** prices to business friendly values.
  - You can choose among different types of **cost calculations** (single, average, sum).
  - You can run the PL/LPG calculation with a **volume breakdown** which allows you to apply different volume discounts, depending on the item quantity.
- In addition, the package is well prepared for handling **errors** and issuing **warnings** if the calculations fail.

Admin User Reference (Price Setting)

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

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	Initial Prices

		• Pricing Level (Country, Global, Channel, ...)	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 Independent (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)
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)	All KPI regarding “Sales History” • Historical Sales/Turnover • Forecast based on historical data

Installation (Price Setting)

This installation tutorial will guide you though an installation and initial configuration steps of Price Setting Accelerator. Price Setting Package can only be deployed to a partition via PlatformManager. The process consists of a few steps; some require you to provide input data, some are more or less automated.

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

Set up Product Segmentation

The next step will display a form which will let you configure product segmentation individually for each supported feature. More information can be found at Product Segmentation.

Here it is also important to do a proper research before submitting the configuration because any re-configuration will require that you do some manual steps.

Package Bootstrapping

The last step of package deployment is an automatically triggered bootstrapping. It will take all the information that you provided in previous steps and create and set up all the required tables. The background process is described in more details here.

Configuration and Price List/Grid Creation

Now that everything has been created you can start the configuration. You can go through the documentation and configure all required features manually by adjusting PPs, but we highly recommend using our "Price Setting Accelerator Configuration Wizard" to do it.

The last configuration task is to create Price Lists or Price Grids that you will use for your independent and dependent levels.

For an independent/standalone Price List/Grid, the selected logic should be "IndependentPriceListLogic". For others, "DependentPriceListLogic" should be used.

To use the Volume Breakdown functionality, you need to create a Price List/Grid of the MATRIX type, then in the Matrix logic select "VolumeBreakdownMatrixLogic" and in the Matrix logic element select "Volumes".

In the Price Lists/Grids, hide the Product Currency column (it gets the information from Product Master and it is not needed here it; currency is taken from the Currency column). To hide a column, use Preferences. You can also set default preferences for a price list in Pricefx Configuration.

Installation Prerequisites (Price Setting)

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

- Get access to **PlatformManager** and the **partition** (as described in common accelerator installation prerequisites).
- Make sure your partition has the following **data** tables and fields:
  - Product Master table with:
    - Fields metadata set up
    - Data rows available (you need at least 1 row to be able to create a price list after installation)
- Learn about **Product Hierarchy**. You need to know which product attributes represent the levels of the product hierarchy.

- Prepare a **CSV file with Dependency Configuration**. This file will be required in one of the deployment steps.
  Dependency Configuration is essential for this Accelerator. It defines what dependency levels (countries, channels, etc.) you will be using. You can find more information at Dependent Price Lists and Data Fallbacks.

  Examine your customer data and requirements before you fill in this configuration because any re-configuration will require that you go through some manual steps. You can find frequently needed procedures in Price Setting Package Administration Procedures.

  You will need to upload this configuration as a CSV file. We suggest to 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 translated into DependencyConfiguration PP, so you can import all the information that this PP accepts.

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

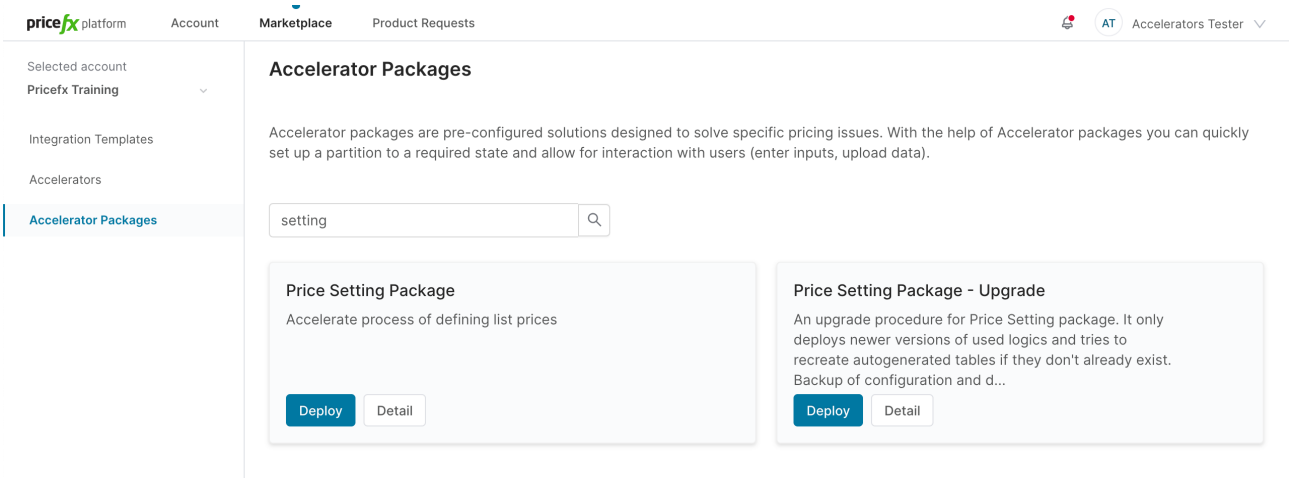
Sample of a CSV file with dependencies:

```
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
```

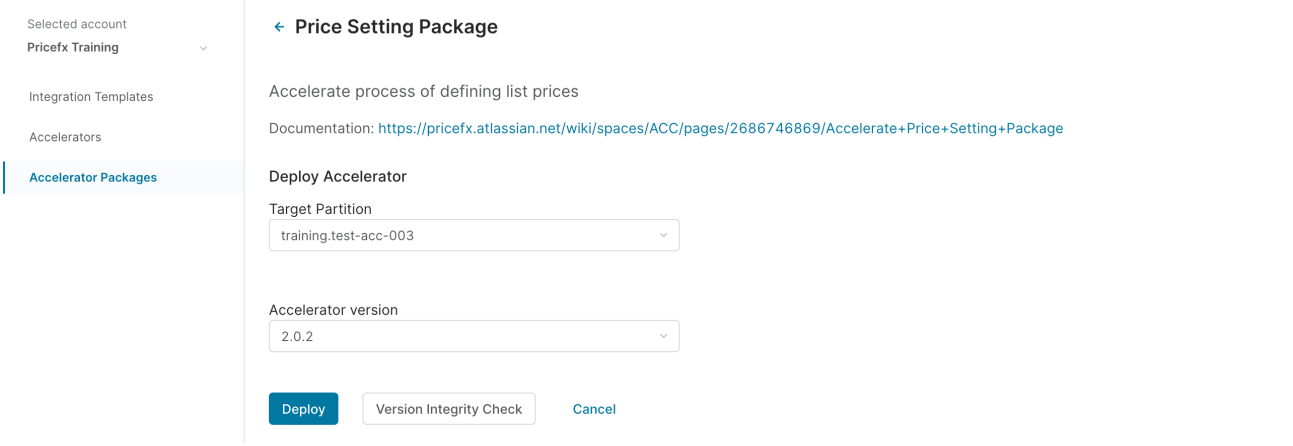
Installation Steps (Price Setting)

To install Price Setting Package:

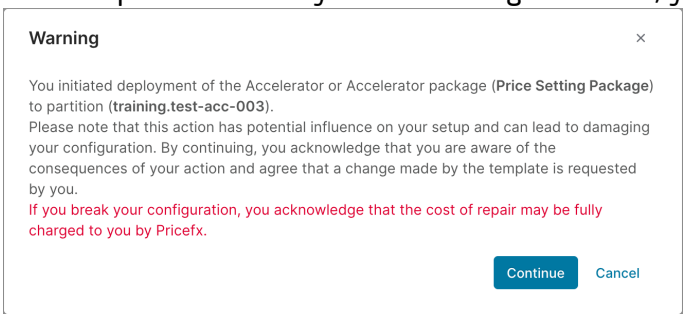
1. Go to PlatformManager at https://platform.pricefx.com/.
2. Navigate to **Marketplace › Accelerators**, find the Accelerator Package *Price Setting Package* and click **Detail**.



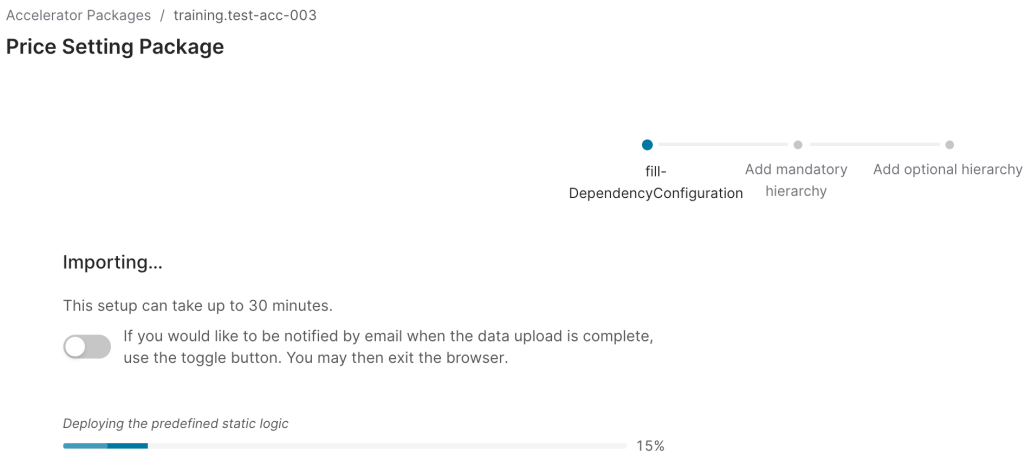
3. Select a partition for installation and version to be installed, and click **Deploy**.



4. There is a warning displayed that installing a new Accelerator may have a harmful impact on the selected partition. Once you acknowledge the risks, you can proceed.

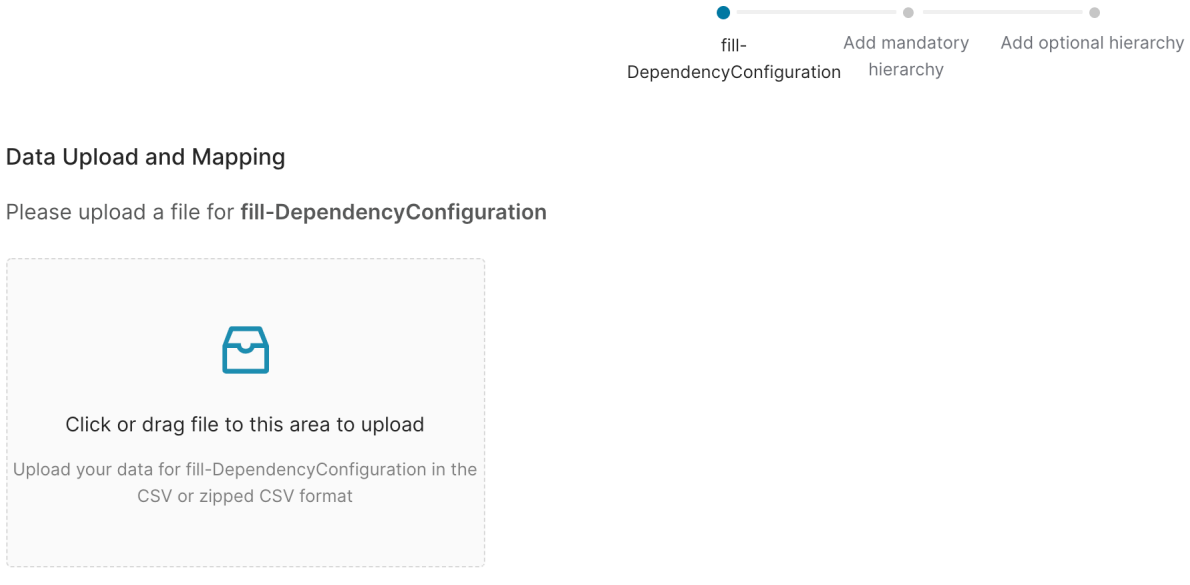


5. The installer will deploy logics to your partition. This can take up to several minutes.

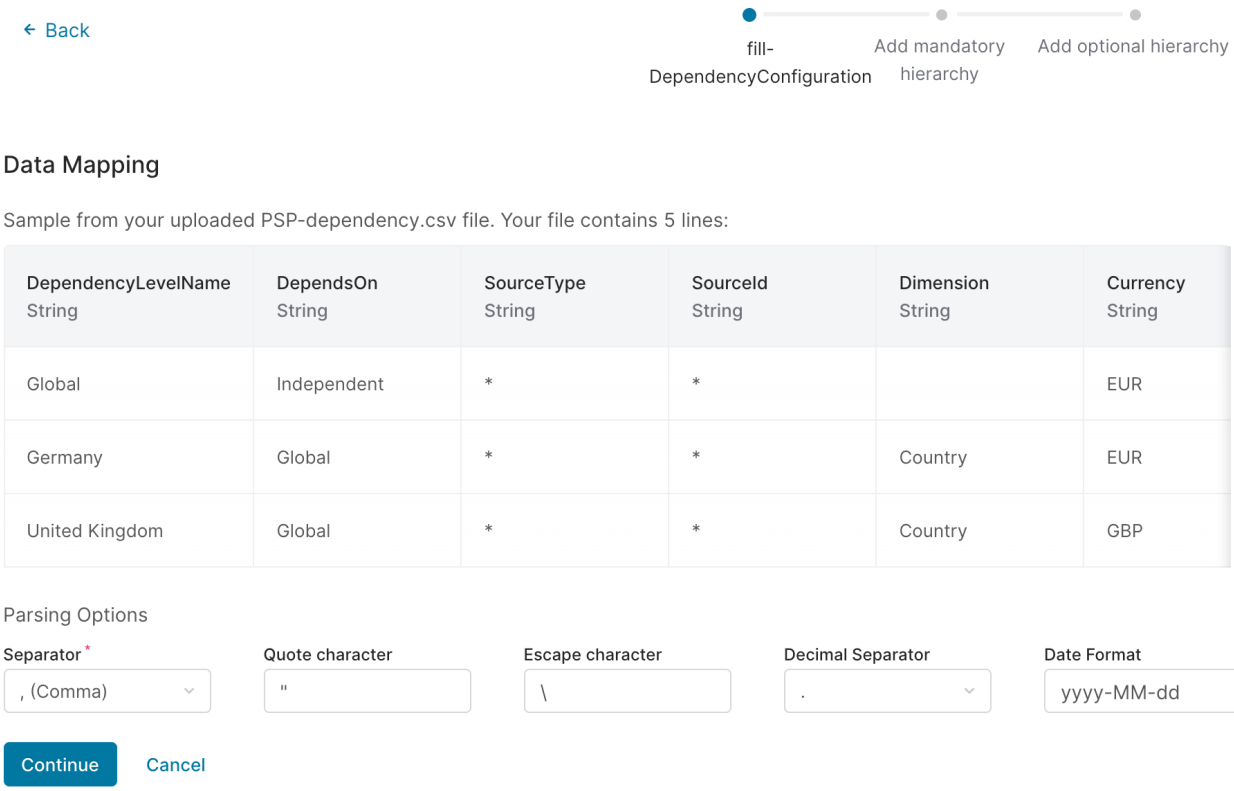


If you need to interrupt the deployment process, you can always come back to it later. You can choose either to continue in the previously started process, or start again.

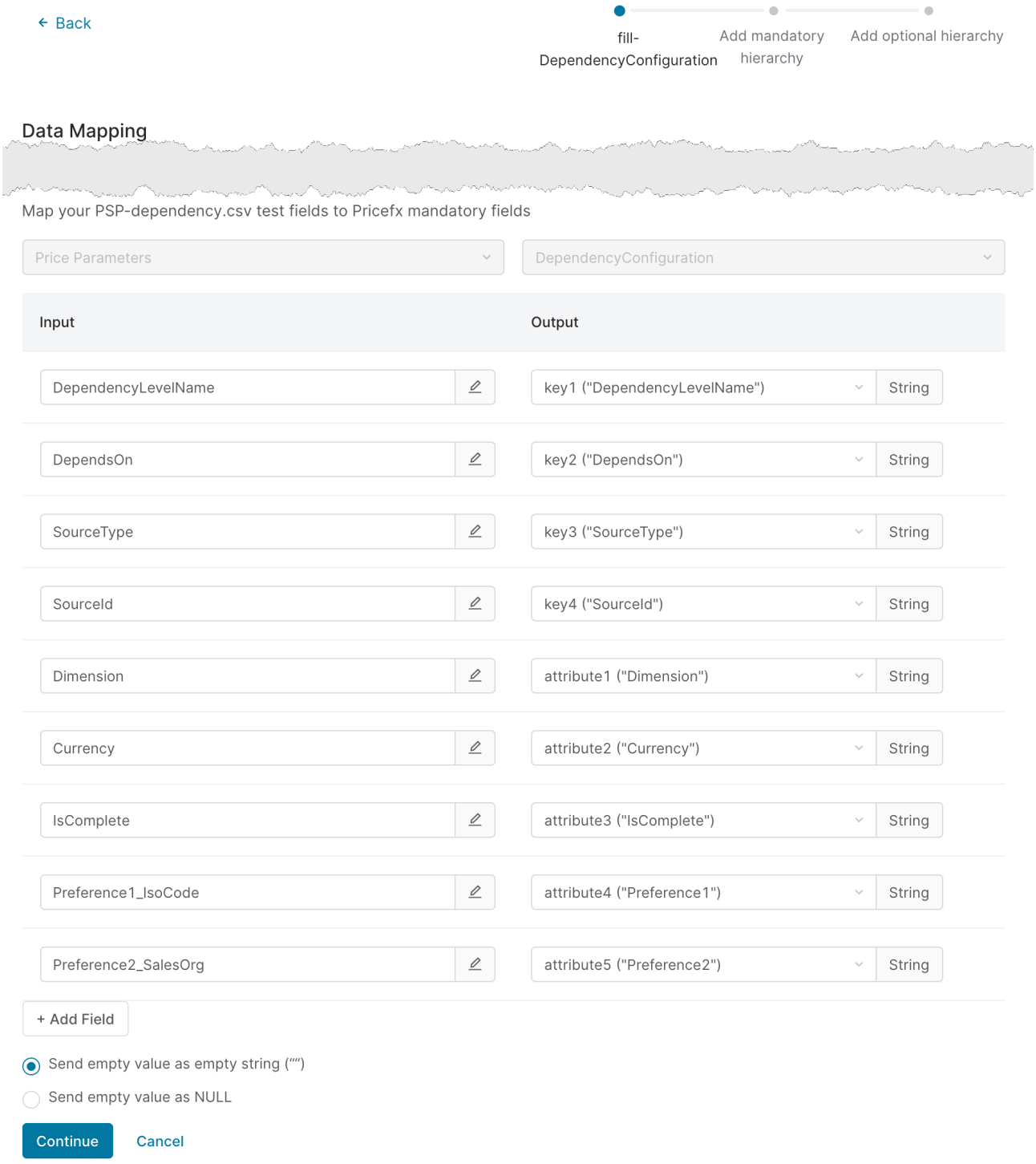
6. Upload the CSV file with Dependency Configuration. For details on this file see Installation Prerequisites (Price Setting).



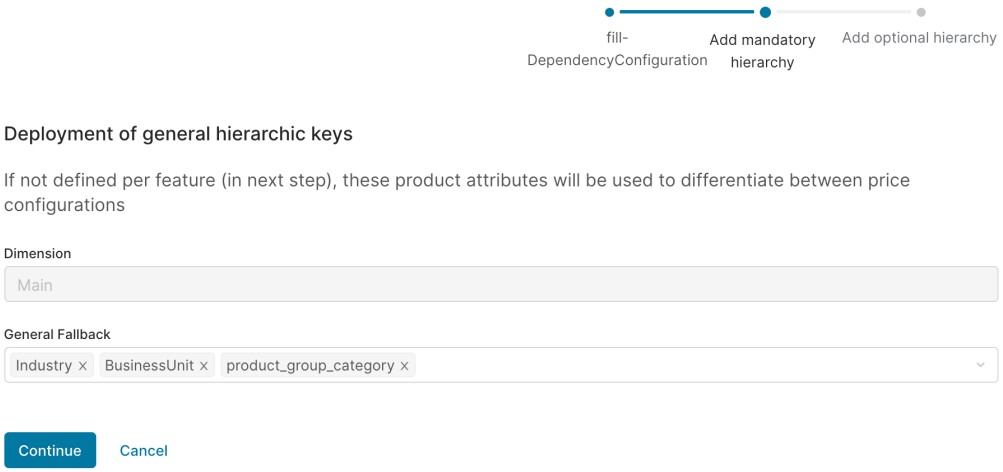
7. Define the parsing setting for the CSV import, and click **Continue**.



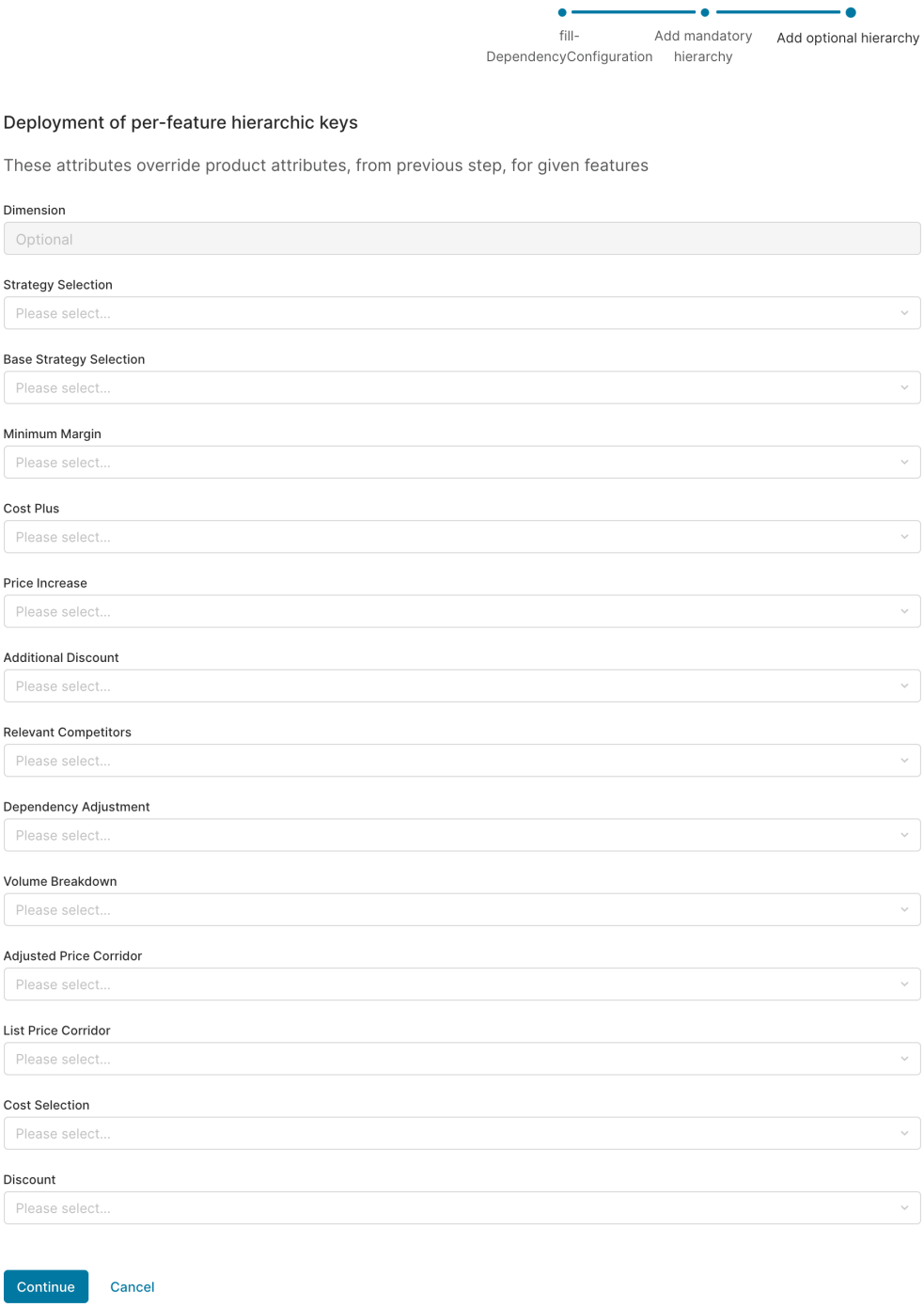
8. Define mapping of your column data from the CSV to the fields of Company Parameter table *DependencyConfiguration*.



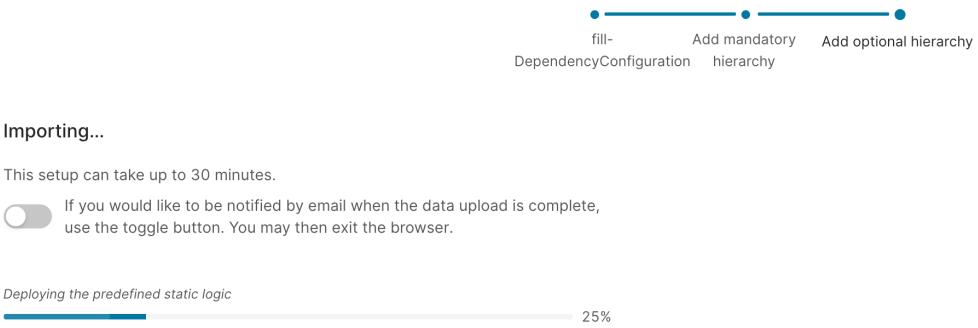
9. Wait for the process to complete.
10. In the next step, define the global product hierarchy for pricing.
    For the *Main* dimension, set up the following dimensions (attributes of product master which store the product hierarchy levels) and then click **Continue**.
    - Industry
    - Business Unit
    - Product Group



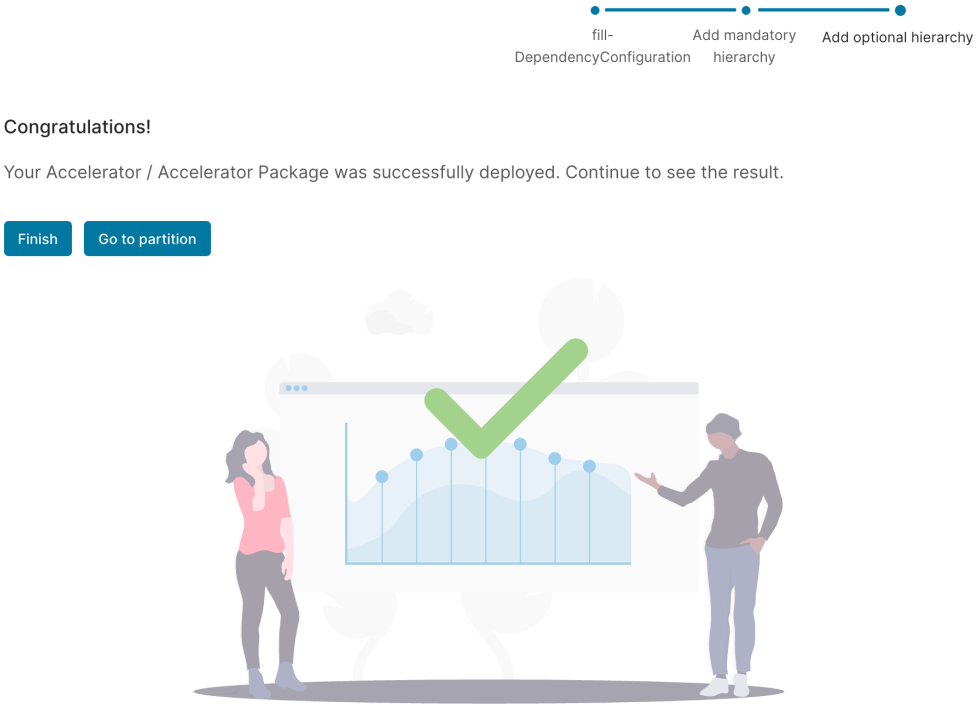
11. Wait for the process to complete.
12. Optionally, define a specific hierarchy per feature. You can set different hierarchies for the different modules/features of the Price Setting Package.
13. Keep the setting empty and click **Continue**.



14. The final importing process takes up to several minutes. It deploys predefined static logic, triggers the logic in Pricefx, etc.



15. Once finished, you will see the congratulations message.



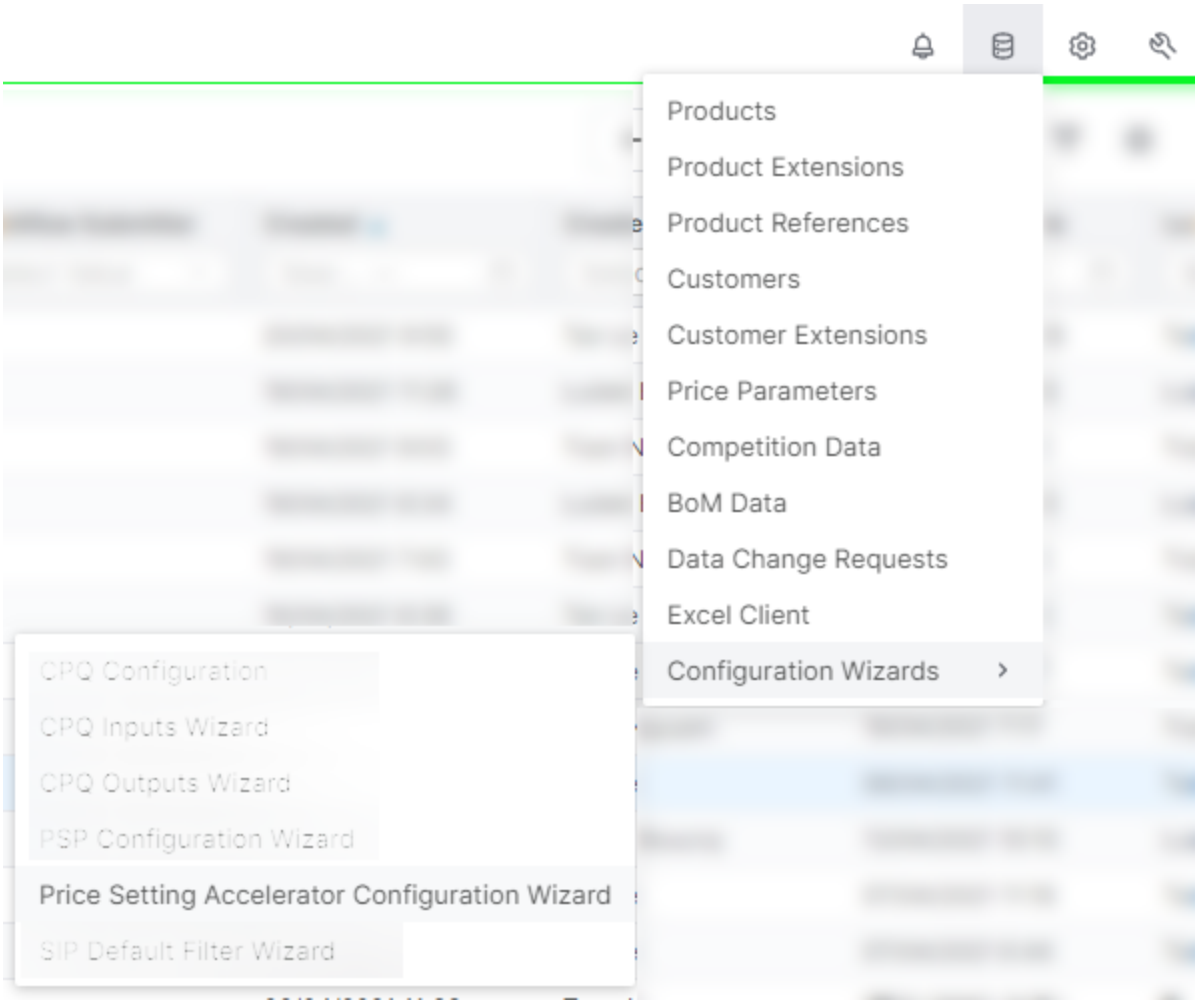
Post-Installation Steps (Price Setting)

- Configuration Wizard (Price Setting)
- Configure Core Elements Module (Price Setting)
- Test Deployed Package (Price Setting)
- Review Installed Components (Price Settings)

Configuration Wizard (Price Setting)

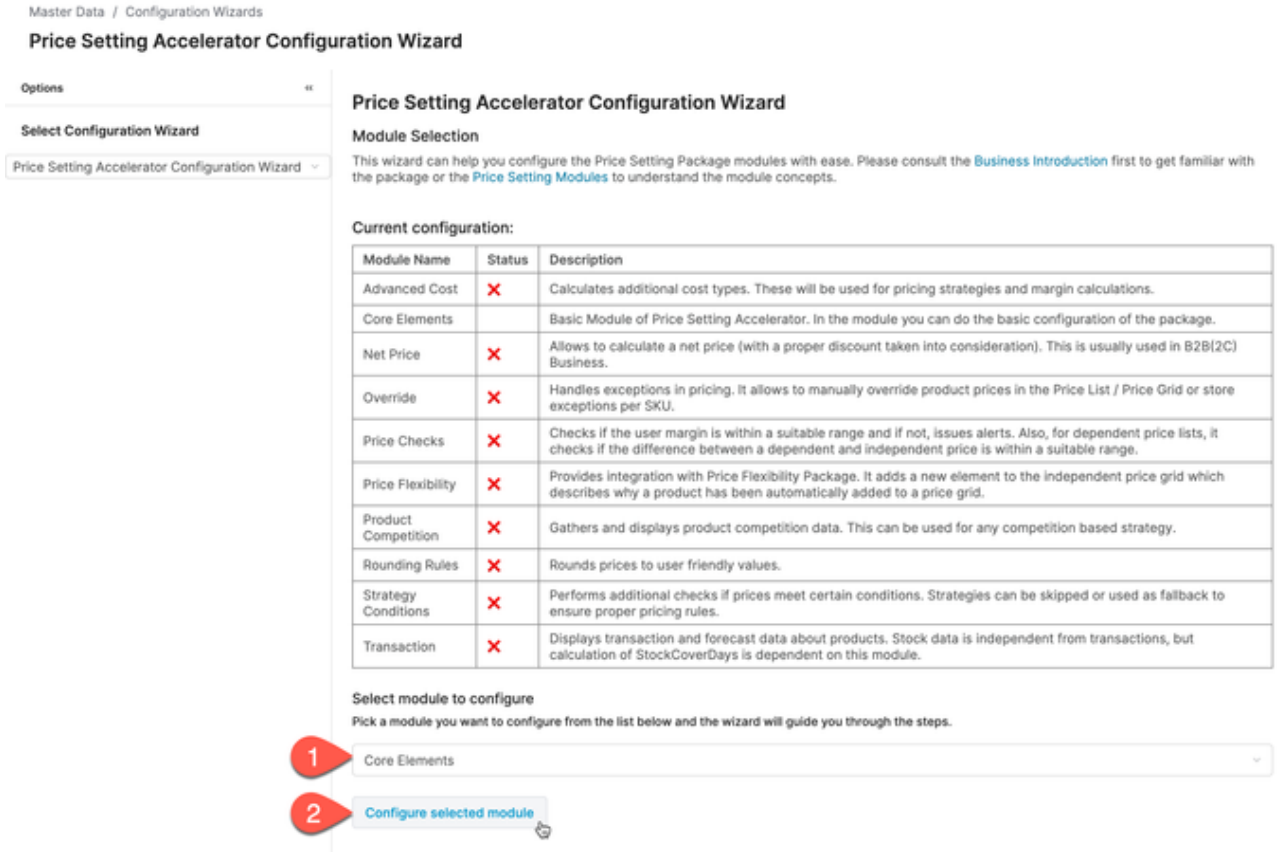
After deploying Price Setting Package you need to configure modules/features of the package. The package comes with a comprehensive wizard to help you with the configuration. With this wizard you can enable/disable and configure the different Price Setting Modules.

The Price Setting Accelerator Configuration Wizard can be found in the default wizard section of the Pricefx application:

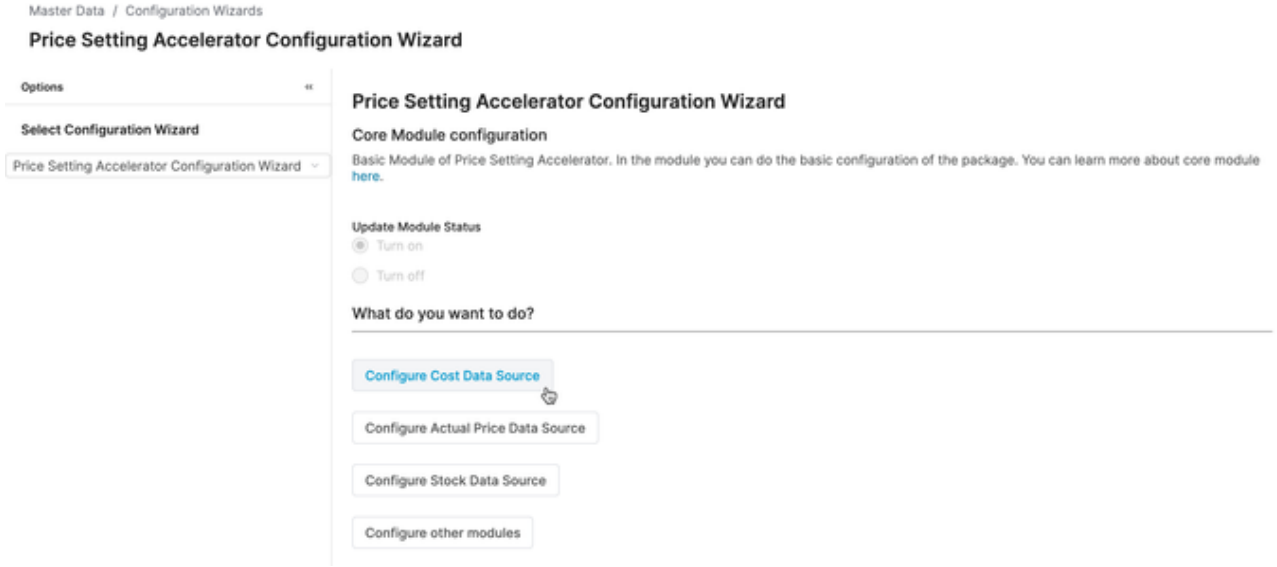


Configure Core Elements Module (Price Setting)

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**.

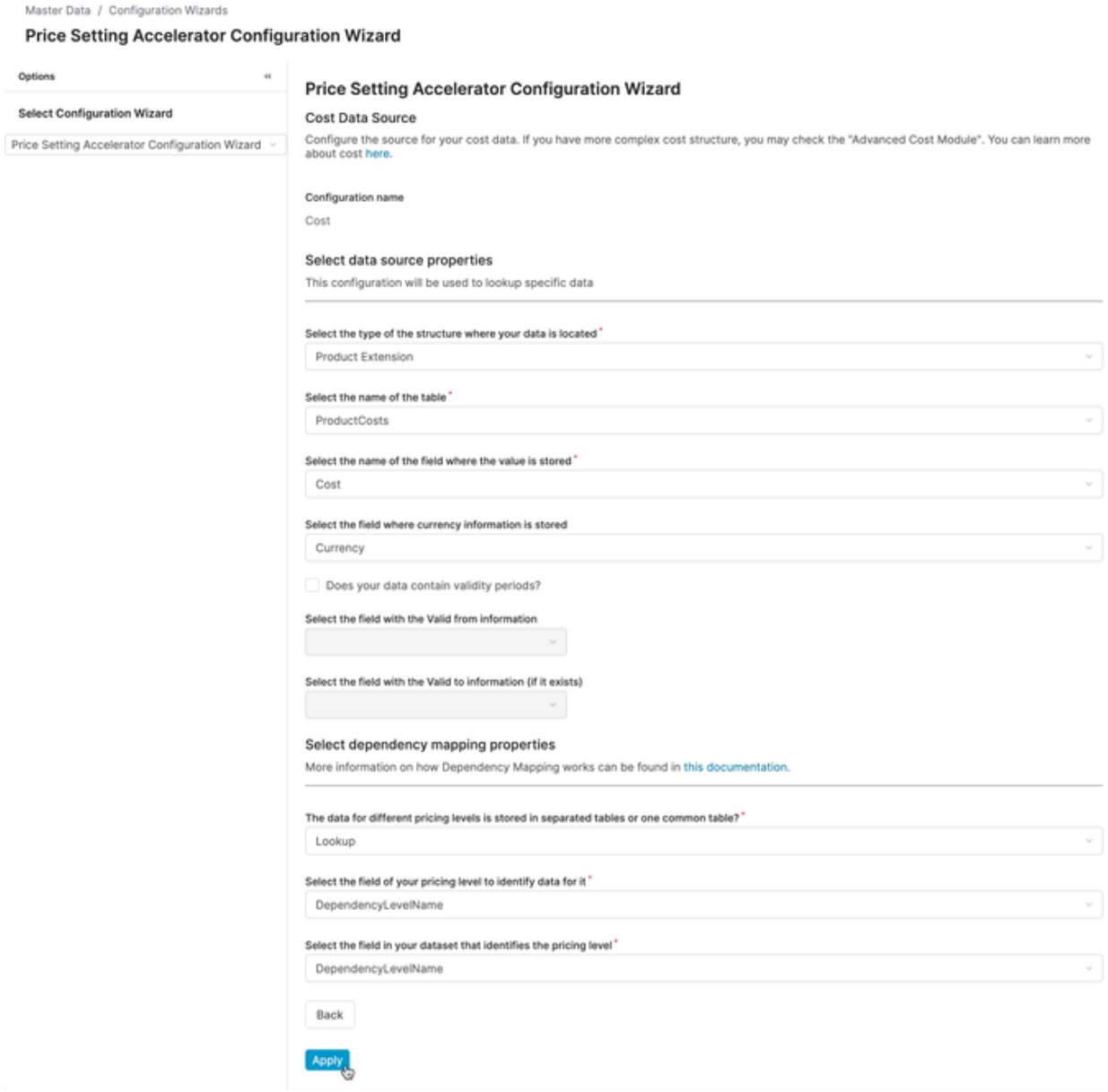


4. Click **Configure Cost Data Source**.

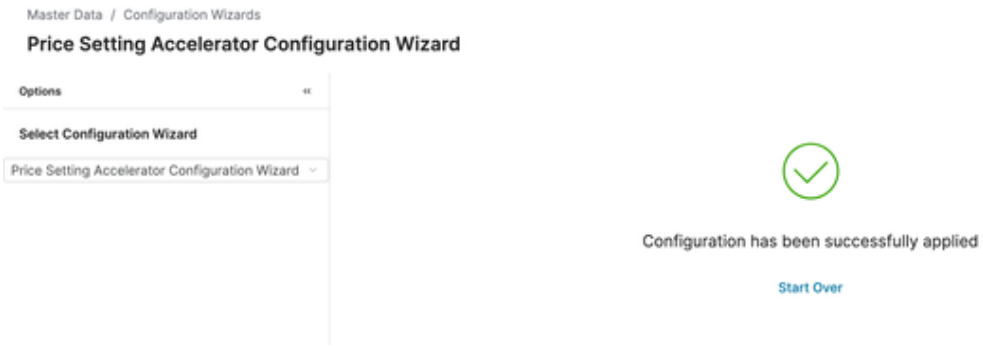


5. Set up Cost Data Source. In our case, we kept the pre-populated default values.
   a. Select Data Source table and fields to be used:
      - **Select the type of structure where your data is located** = Product Extension
      - **Select the name of the table** = ProductCosts
      - **Select the 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.



Test Deployed Package (Price Setting)

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

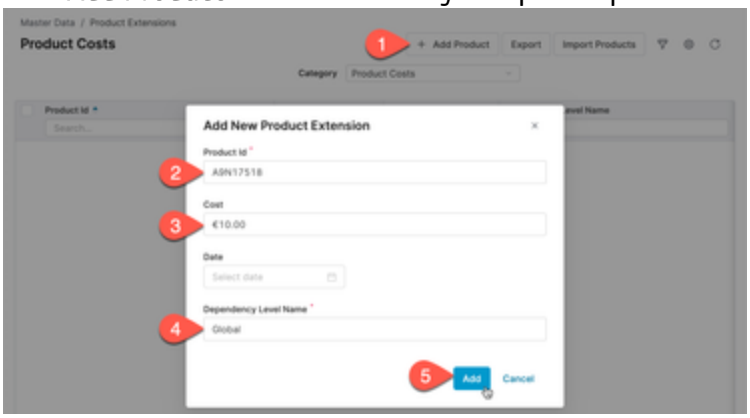
- Prepare Sample Data

- Create and Calculate Price List

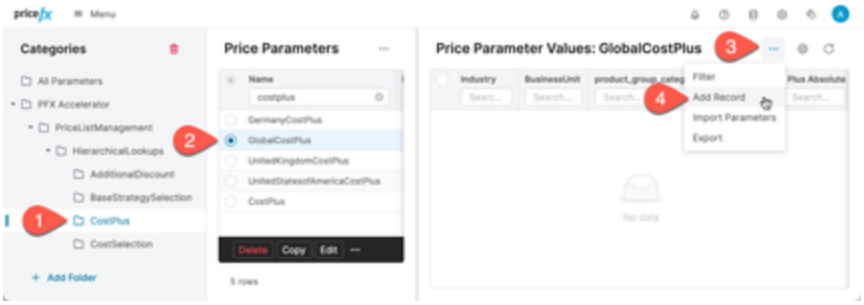
Prepare Sample Data

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.

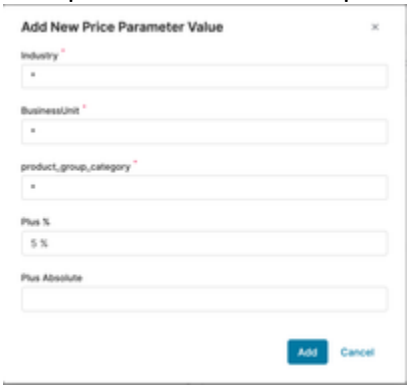
1. Manually add cost of some of your products.
   a. Navigate to **Master Data › Product Extensions**.
   b. Click **Add Product** and set cost for your specific product.



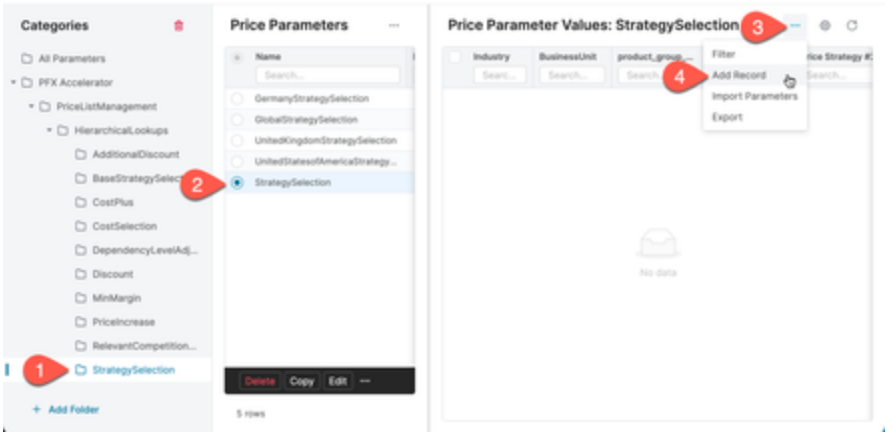
2. Manually add the percentage value added to the cost to get the price.
   a. Navigate to **Master Data › Company Parameters**.
   b. Find and open the detail of the Company Parameter *GlobalCostPlus* and then click **Add Record**.



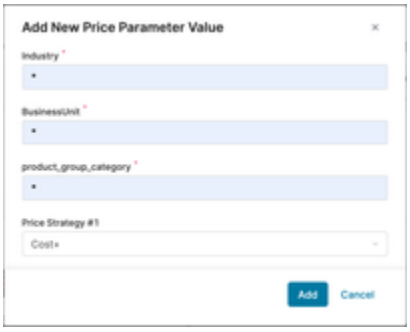
   c. Set up a default 5% for all products (regardless of the product hierarchy).



3. Set up a default price calculation strategy.

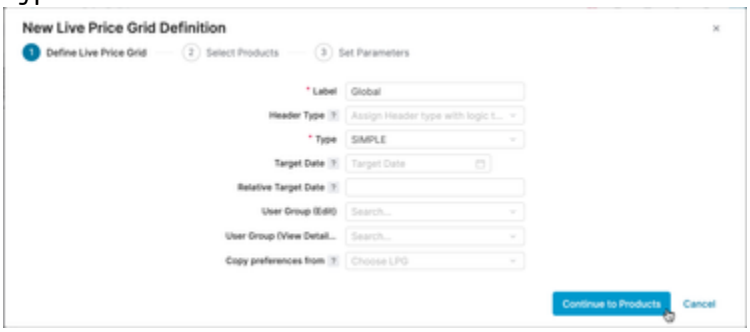


a. Set the record data.

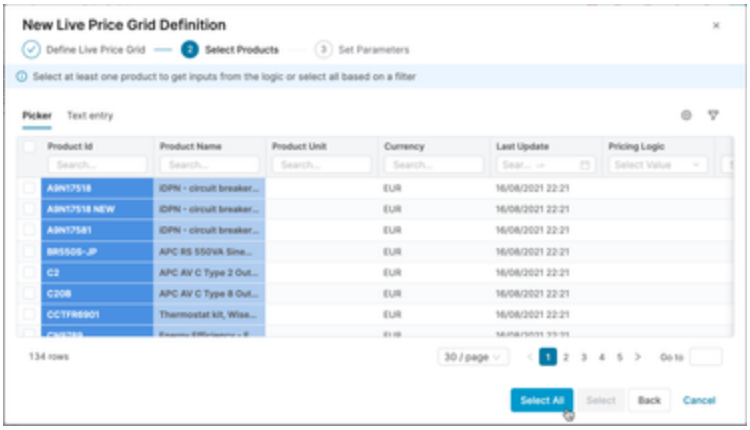


Create and Calculate Price List

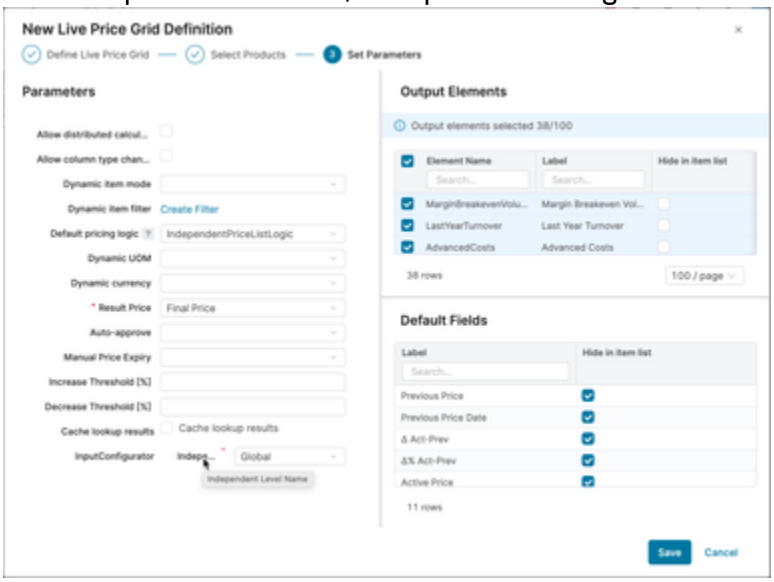
1. Navigate to **Price Setting › Live Price Grids**.
2. Click **New Price Grid**. A dialog with 3 steps will pop up.
   a. In the step *Define Live Price Grid*, set up the label and type and then click **Continue to Products**.
      - Label: Global
      - Type: SIMPLE



   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.

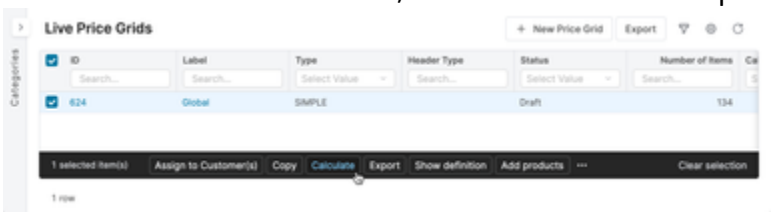


d. In the section *Parameters*, set up:
   - Default pricing logic: IndependentPriceListLogic
   - Result Price: Final Price
   - InputConfigurator
     - Independent Level Name: Global

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

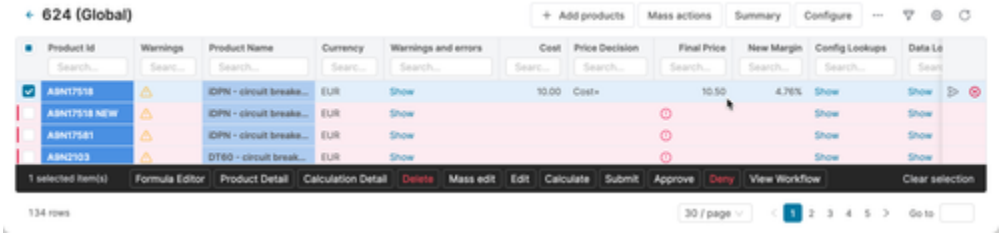
f. In the section *Default Fields*, set *Hide in item list* for all fields, so that the price list does not have too many columns for now.

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

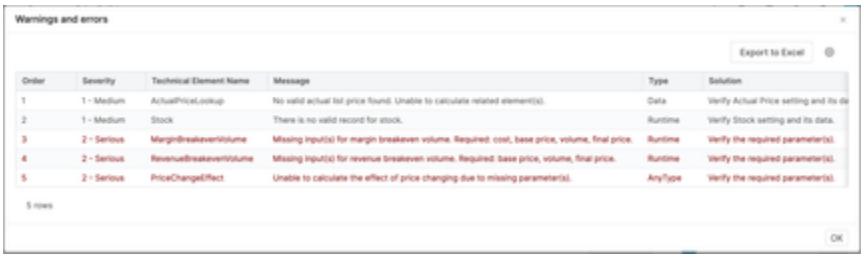


4. Wait for the *Ready* state.
   Note: The calculation will take time based on the number of products calculated. If you have thousands of products you may want to leave the calculation running in the background and come back later.
   Wait a couple of seconds and then refresh the list of LPGs to see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



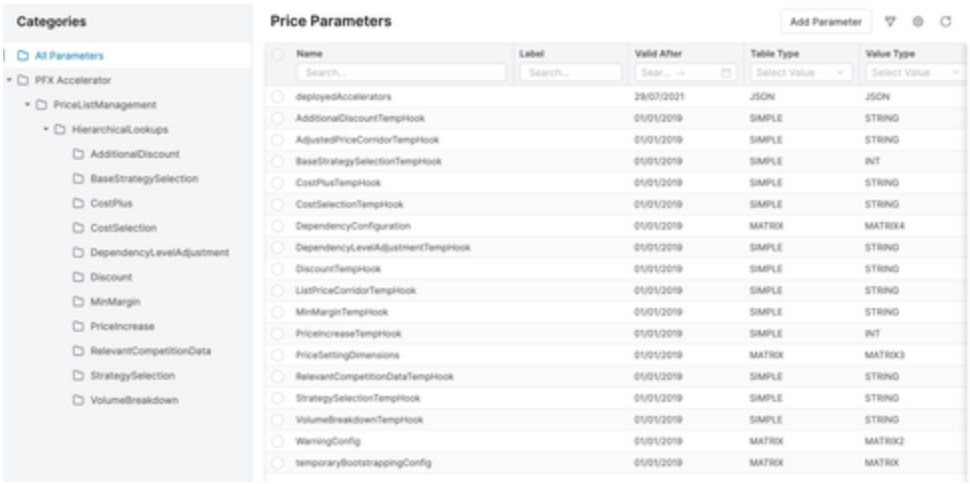
   a. You should be able to see on the line:
      - Cost you defined for your product
      - Defined strategy
      - Final price
      - Margin gathered on the product when sold for the final price
   b. Click the *Show* link located in the **Warnings and errors** column to see why the prices were not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



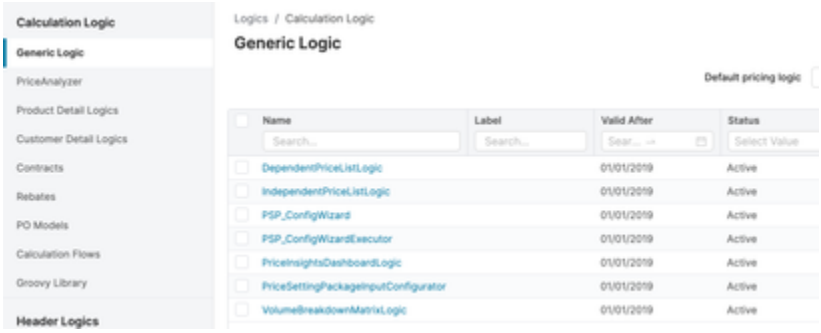
Review Installed Components (Price Settings)

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

- Product Extensions
  - Product Costs
  - List Prices
  - Promotion Prices
  - Recommended Retail Prices
- Company Parameters
  - Many new Company Parameters in the following folder structure:

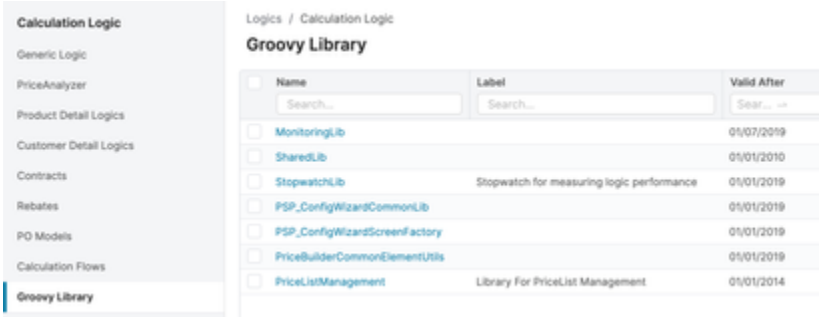


- Dashboards
  - PriceInsightDashboard
- Configuration Wizards
  - Price Setting Accelerator Configuration Wizard
- Logics
  - Generic Logics



  - Calculation Flows

  - Groovy Library



Architecture Components (Price Setting)

Table of Contents

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
- IndependentPriceListLogic
- PSP_ConfigWizard
- PSP_ConfigWizardExecutor
- PriceInsightsDashboardLogic
- PriceSettingPackageInputConfigurator

Matrix Logics

- VolumeBreakdownMatrixLogic

Dashboards

- PriceInsightsDashboard

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

- 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

Upgrade (Price Setting)

Upgrade of Customized Accelerator (Price Setting)

> ⚠ We do not provide support for custom modifications, so before raising any issue in Accelerator, please make sure it is not caused by your custom code.

As Accelerators utilize the usual common Pricefx objects and wrapped pfxpackage tool for deployment, they are bound with the same restrictions as any other customer project. The biggest one is that **upgrading the package overwrites objects existing on the partition**. For this particular reason, any changes made to the package logics will be overwritten during an Accelerator upgrade and will have to be re-applied manually.

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

- **Tier One** - User facing logics. These define what data is shown to the user in price lists / grids together with some hidden technical elements. We try to implement these elements as simple "dispatchers", so usually the only thing that they do is to gather required inputs from previous elements and dispatch the calculation to Tier 2 logic. Because it is the really easy to swap the dispatching method to use some custom library, **this is where we recommend to make all of your modifications**. Upgrades should not cause any complex conflicts even if the underlying Accelerator business logic changes and re-applying modifications should be a breeze. Logics:
  - IndependentPriceListLogic
  - DependentPriceListLogic
  - VolumeBreakdownMatrixLogic
- **Tier Two** - Common logic libraries. This is where most Accelerators business requirements are implemented. It is full of detailed utilities that calculate and prepare data for Tier 1 logics. This is where most package's complexity is handled. **Changes at this tier are still possible but not particularly encouraged.** Just be aware that resolving conflicts after upgrading this logic may be quite demanding. Logics:
  - PriceBuilderCommonElementsUtil
- **Tier Three** - Library logics. These are calculations and utilities that are designed to be working independently from the rest of the package. This tier stores for example all calculation engines' implementations. Over time this logic became smaller and smaller because we moved some of the

existing utilities to the Pricefx Groovy Library (SharedLib). **Changes at this tier are not recommended.** L ogics:
  - CalculationEnginesLib
  - SharedLib from Shared Groovy Library
- **Tier X** - All remaining logics that do not directly affect business requirements. Changing them will usually not be necessary, but if for some reason it is - you can treat them as Tier 2 when it comes to upgradeability. Logics:
  - PriceSettingPackageInputConfigurator
  - PSP_ConfigWizard
  - PSP_ConfigWizardCommonLib
  - PSP_ConfigWizardExecutor
  - PSP_ConfigWizardScreenFactory

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** > **Accelerator Packages** > **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.

Technical User Reference (Price Setting)

- Configuration Tutorial (Price Setting)
- Configuration (Price Setting)
- Common Configuration Procedures (Price Setting)
- Technical Information (Price Setting)
- How to Customize (Price Setting)

Configuration Tutorial (Price Setting)

- Tutorial Case Study
- Create New Partition
- Install Demo Data
- Install the Price Setting Accelerator
- Configure Price Calculation and Calculate Prices
- Initial Setup of Price Setting Modules

- Cost Plus Pricing
- Competition Based Pricing
- Attribute Pricing
- Dependent and Independent Pricing
- Custom Price Strategy Configuration
- Custom Fields in Price List/Grid

Tutorial Case Study

This case study provides you a high level business requirements, which you will implement in the next tutorials.

Introduction

A potential customer is a large Discrete Manufacturing Group, with its global headquarters in Zurich, Switzerland.
They are currently seeking a new pricing solution that is flexible enough to handle a global business and several complex use cases that they are currently managing manually.
A real dataset has been provided by the client, and they have requested a working demonstration of the Pricefx Price Setting capabilities within 48 hours.
You have been given 24 hours to produce it, as your management team needs time to review and sign it off ahead of the demonstration.
You have the help of an SME and the knowledge base documentation to guide you.

Business Information

The customer manufacturers electrical components and sells them to other businesses who use them in the manufacture of their own end products. This is a Business to Business (B2B) relationship. The customer does not sell to consumers directly.
The customer has locations in Switzerland, Poland and Germany.
The business is separated hierarchically by the Business Unit, Product Group, and Product Line.
Pricing is set at headquarters and then tailored to various variables, market conditions, and locations to get the best result.

The Dataset

The dataset comprises product and competition-based data.
From the product data, we can determine the product cost, currency and we see they are grouped into discrete product groups that separate them into a logical hierarchy.
The competition data provides us with key list prices for some products, based on competitors' pricing.

The Requirement

The customer has requested a demonstration of the following capabilities:

1. A global default price list that contains product cost information and a standard% or fixed cost added to determine a global list price.
2. A pricing strategy that considers competitor pricing, along with the ability to prioritize certain competitors over others. The hierarchy for the competitors is 1)ABB; 2)Legrand; 3) Eaton
3. An attribute based pricing strategy that enables the customer to differentiate prices based on the following 3 attributes:
   a. Competitiveness
   b. Size
   c. Color
4. A Dependency based pricing strategy, where the various factories can set prices based on local market conditions, but with a requirement (dependency) to use the global default price list as a starting point.

Create New Partition

Outcome

At the end of this tutorial, you will have a new empty partition, which you will be able to use for the practicing of Price Setting accelerator configuration.

Pre-requisites

- access to Platform Manager
  - rights to create new partition
- approver, who will approve your partition request

Create New Partition

> Here we're describing only steps specific for the need of this tutorial. For details about the partiton creation process, review official How to Create Partition in the Platform Manager documentation.

Login to Platform Manager.
Navigate to menu **Account**, and select account *Pricefx Training*.
On page **Account › Partitions** use the button **Create Partition** and fill in the following settings:

- **Partition Serial Name**: training-ps-acc-<your name>
- **Partition Label**: Price Setting Accelerator Training of <your name>
- **Purpose**: Training
- **Cluster**: Shared Training Cluster
- **Currency**: EUR
- **Unit of Measure**: EA
- **Activate Modules**: Analytics (for 5 users), Price Setting (for 5 users)
- **Number of Admin users**: 5
- **Usage end date**: <select a date right after your training>

Use **Submit** to send the request for partition.
Wait for the approver to approve it - then you will receive credentials for the *admin* user.

Set up the Admin User on Partition

Using the received credentials, log in to the partition.
Navigate to **Administration › Access Admin › User Admin**
Select the row with user *admin*.
In the *User Roles* tab/section (on the right side), find and cross the following roles.

- Master Data
  - Administer Company Parameters
  - Administer Customers
  - Administer Products
- Price Setting
  - Administer LPG
  - Administer Price Lists
- Administration
  - General Admin
- Analytics
  - Administer Schemas
- Dashboards
  - View Dashboards

Save the user roles setting.
Reload the page in browser.
Ensure you can see modules *Analytics* and *Price Setting* in the menu in the left top corner.

Install Demo Data

Outcome

At the end of this tutorial, you will have the partition populated with demo data needed for the training.

Pre-requisites

- access to an empty partition on the training server
- access to Platform Manager
  - rights to install *Non approved* accelerators in *Market Place*.

Install the Demo Data

> Here we're describing only steps specific for the need of this tutorial. For various general information about accelerators installation, review the official How to Deploy Accelerator Package in the Platform Manager documentation.

Log in to Platform Manager.
Navigate to menu **Marketplace**.
Select account *Pricefx Training*.
On page **Marketplace › Non Approved**, find the accelerator named *Price Setting Sample Prerequisite Partition*.
Use button **Deploy**, select the partition created for this training and **Deploy** the accelerator to the partition.
Once finished, review the tables created on your partition:

- *Products*
- Product Extension *Product Cost*
- *Competition Data*

Install the Price Setting Accelerator

Outcomes

At the end of this tutorial, you should be able to:

- collect the pre-requisites based on the business requirement
- install Price Settings accelerator

Prerequisites

- partition with Products data

Business Decisions and Preparations

Price Dependencies, Pricelist Hierarchy

In the Case Study requirements, you can read, that:

- *The customer has locations in Germany, United Kingdom and United States.*
- *Pricing is set at headquarters and then tailored to various variables, market conditions, and locations to get the best result.*
- *A Dependency based pricing strategy, where the various factories can set prices based on local market conditions, but with a requirement (dependency) to use the global default price list as a starting point.*

It comes out, that in some cases the prices will be done calculated in 2 steps:

1. first the global price
2. then the local prices, which will depend on the global prices

From Price Setting Accelerator perspective it means, that there's a dependency between the *global* and *local* pricelists.
The *global* pricelist can be calculated without having any other one, so it will be *independent*. The *local* pricelist needs to have the *global* pricelist to be calculated first and it will use the *global* prices as a starting point. So the *local* pricelist will be *dependent*.
To provide this information during the installation process, create a CSV file with content

```
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
```

When visualized in table, it would look like this:

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

Product Hierarchy

In the Case Study requirements, you can read, that:

- *The business is separated hierarchically by the Industry, Business Unit, Product Group.*

Based on the discussion with customer it became obvious, that various calculation factors depends on the combination of those product hierarchy attributes.
Log in to your partition, review the *Products* table and find the fields with the attributes forming the product hierarchy.

Install the Price Setting Accelerator

Here we're describing only steps specific for the need of this tutorial. For more detailed descriptions (including screenshots) of the Price Setting accelerator installation, visit page Install Price Setting Accelerator v2.0.2. For various generic information about accelerators installation, review the How to Deploy Accelerator Package in the Platform Manager documentation.

the process of installation can have slightly different naming of inputs, but generally the process is the same for 2.0.2 and 2.1.x version of the Price Setting accelerator.

Log in to Platform Manager.
Navigate to **Marketplace**.
Switch the selected account to *Pricefx Training*.
On the page **Marketplace › Accelerator Packages**, find the *Price Setting Package* accelerator.
Click on **Detail**.
For the *Target Partition* select your partition used for this training.
Select the latest Accelerator version.
Click **Deploy** to start the process.
Upload dependency configuration from the CSV file. After upload, you will be prompted to set the data mapping - ensure, that the mechanism parsed your file ok - if yes you will see your data in the form of meaningful table and you can **Continue**.
Most fields are mapped ok, just the last 2 must be done manually to *Preference1* and *Preference2* fields.
Once set, do **Continue**.
When asked to provide the hierarchy keys, select those product attributes, which were identified and decided in the business decision above.
For this tutorial, do NOT use any per-feature hierarchic keys, all the features should be based on the same hierarchy of product attributes.

So for the common general fallback, select fields (all of them are added, one by one, into the same input field):

1. Industry
2. Business Unit
3. Product Group (product_group_category)

Continue and after a while, the installation will finish.

Review the Product Extensions tables

To avoid confusion in the next tutorials, shortly visit the Product Extensions and review the "cost" tables. You will find, that there are 2 tables with similar names:

- Product Cost - this comes from our demo data package
- Product Costs - this is a templates provided by the Price Setting accelerator

Configure Price Calculation and Calculate Prices

Outcome

At the end of this tutorial, you should be able to:

- configure a Live Price Grid, which will utilize the Price Settings accelerator for calculations of the prices.
- calculate the prices
- review the calculation errors from the accelerator.

Prerequisites

For this tutorial you need partition with:

- Price Setting accelerator
- demo product data

Business Decisions

Based on the use case requirements, we should calculate both global (*independent*) and local (*dependent*) prices.
We will start with calculation of *independent* prices, which do not need any other prices calculated yet.

Configure a Live Price Grid

> Here we're describing only steps specific for the need of this tutorial. For more detailed descriptions (including screenshots) of the Live Price Grid configuration, visit page How to Create and Configure a New LPG.

Log in to your training partition.
Navigate to **App › Price Setting › Live Price Grid**
Use action **+ New Price Grid** and select *(default)* type.

1. Define Live Price Grid
   - Label: <give it some name>
   - Type: SIMPLE
2. Select Products
   - use button **Select All** to select all products
3. Set Parameters
   - Allow column type change
   - Default pricing logic: *IndependentPriceListLogic* (this is the logic provided by Price Setting accelerator)
   - Result Price: *Final List Price*
   - Independent Level Name: *Global*
   - Output Elements: <select all>
   - Default fields: <hide all in item list>

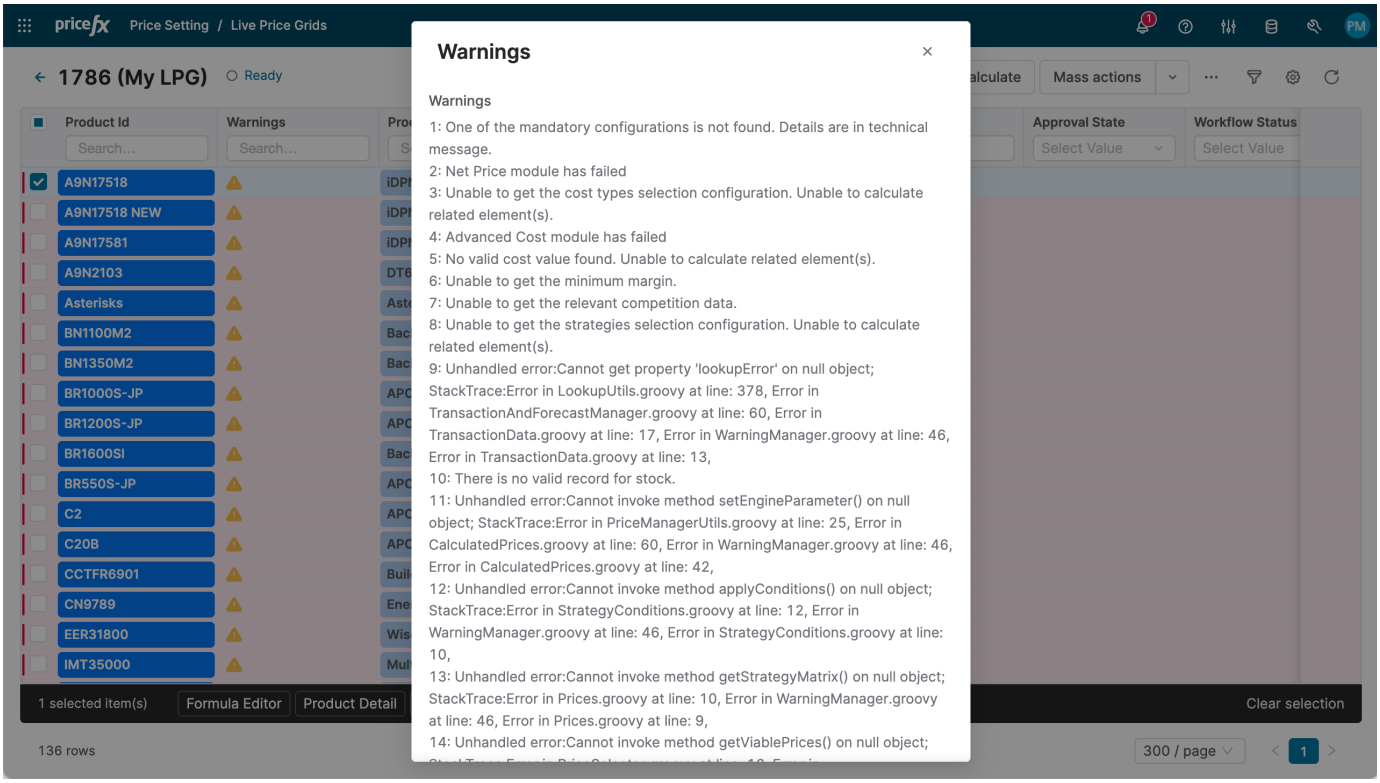
Hit **Save**
In the list of Live Price Grids, select the one you just created, and use action **Calculate**.
As we have only about 100 products, it should take only couple seconds.

Warning Manager and Debugging

It shouldn't be a big surprise that there is nothing calculated so far because we just started with a blank partition.
However, what we can see is the amount of warnings we have received. Our warning manager can provide valuable hints when configuring the package for the first time or when debugging issues.
You can view all the warning that are configured and can find more information in our warnings and error pop-ups, where you can also see the severity of the warning, along with useful technical debugging information.
As we progress with adding additional configuration parameters to the price setting strategy using the configuration wizard, we'll see the errors dropping away.

Open your live price grid to see, what's calculated for the products.
Review the whole row for some product to see all the alerts and issues being found. You can also see the detail with messages, when clicking on the icon in column *Calculation Results* (the last column).
To see all the warnings

- either click on the exclamation mark icon in the *Warnings* field,
- or click on link *Show* in the *Warnings and errors* column. Here you will get good idea about severity of each issue too.



Initial Setup of Price Setting Modules

Outcome

At the end of this tutorial, you should be able to:

- make initial selection of the modules used by Price Setting accelerator, when calculating prices.

Prerequisites

For this tutorial you need partition with:

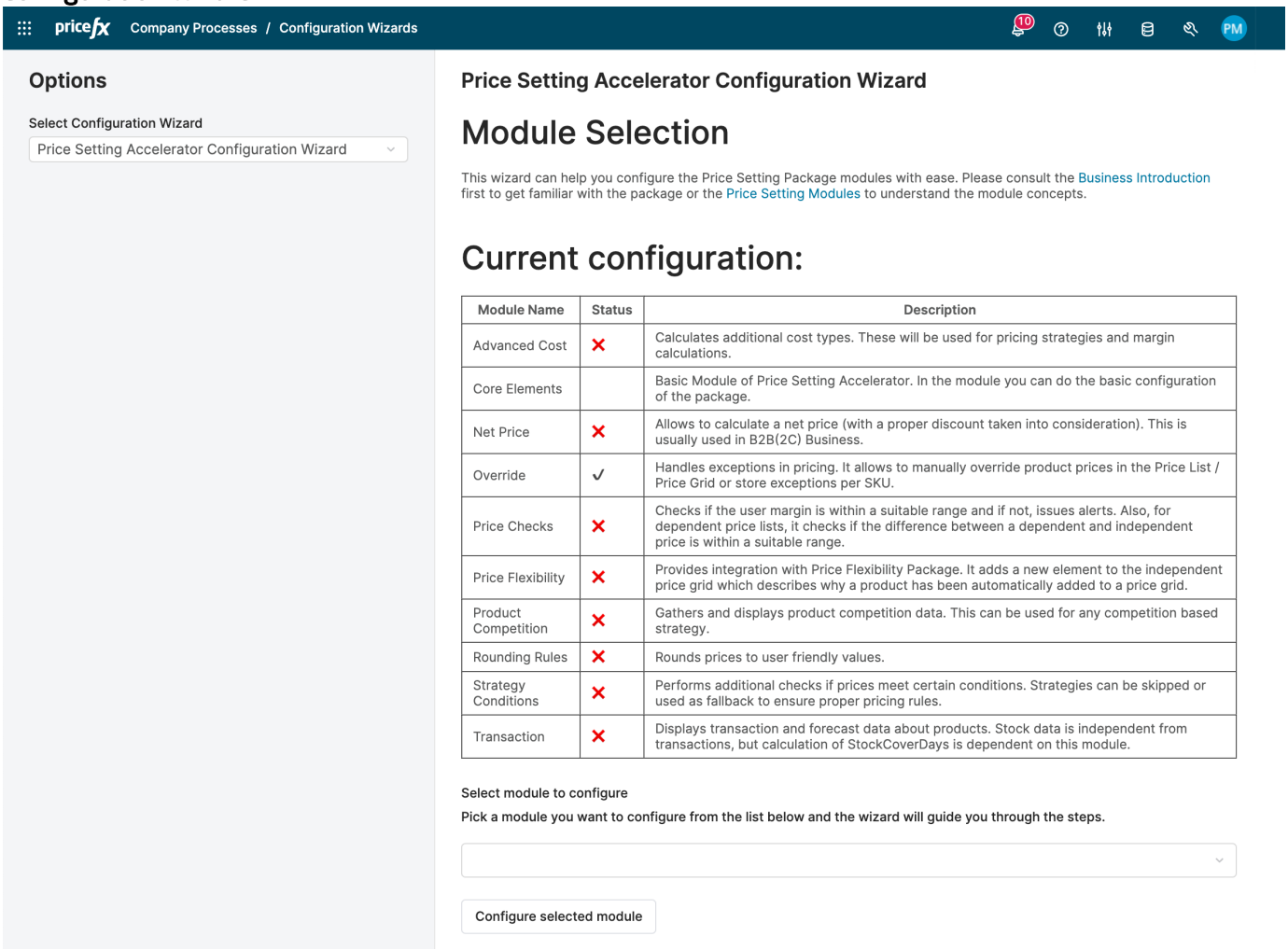
- Price Setting accelerator
- demo product data

Business Decisions

Based on the use case requirements, we should calculate the prices only based on certain strategies. So at minimum, we should limit the amount of modules (capabilities) which the accelerator is capable of.
But since we're at the beginning, we'd like to limit the capabilities to minimum and only add features as needed.

Using the Configuration Wizard

The Configuration Wizard is the primary tool you will use to configure the Price Setting Package.
You can find The Configuration Wizard in the menu **Company Processes › Price Setting Accelerator Configuration Wizard**



Module Selection

The first thing in Configuration Wizard is the Module Selection Panel. On the top you can notice a link to the documentation, where you can find further information on all configuration setting.
By default, after installation, the system has enabled all modules. The first thing we're going to do is disable most of them and then later we can enable them step by step, as needed.
Turning these modules off will disable their functionality. However, it will also reduce the number of warnings and errors related to it and reduce the amount of needed configuration to get started.

You can read about the main functionality of each module in the description box to determine when you may need to turn it on as you configure different sections within the system.

Let's begin with the *Advanced Cost Module*. Select it from the dropdown menu, below the table and select the **Configure Selected Module** button.
In every module configuration, there is a turn-off, turn-on section. We update the module status and we can see it's disabled. We do the same for each of the other modules, leaving only the *Override* module enabled.
You'll also notice the *Core Elements* module, which remains permanently enabled.

Impact on LPG Warnings

Recalculate your live price grid, and review the list of warnings and errors again. You should see, that the list is significantly shorter, than before, because the calculation is not trying to use all the features.

Cost Plus Pricing

Table of Contents

Outcome

At the end of this tutorial, you should be able to:

- configure Price Setting accelerator for price calculations using Cost+ strategy.

Prerequisites

Training partition with:

- installed Price Setting accelerator
- demo product data

Business Decisions

In the business requirement we can read:

> *The customer has requested a demonstration of the following capabilities:*
>
> - *A global default price list that contains product cost information and a standard % or fixed margin added to determine a global list price.*

So we will need:

- to set up a Cost Plus strategy
- an independent price list for the global prices. Since we already have independent global LPG configured, we will use it.

Configuring a Cost Plus Strategy

In order to create a cost plus strategies we will need to tell the system:

- where we can find the costs,
- that it should calculate using the CostPlus strategy
- what is the plus method or the plus we want to use.

Cost Data

Review the Cost Data

The cost information is stored in a product extension, on the master data menu (top right).
Navigate to the product extension *Product Cost* provided in the demo data, and we can see it already contains the cost information. This is the cost source for all of our products and we have a dependency level name which is blank as we're using a global price list. If we have a quick look at our Live Price Grid, we can see that we have a field for the cost, which is empty, because it's not been configured yet.

Configure the Cost Data Source

Navigate to **Company Processes › Price Setting Accelerator Configuration Wizard**.
The *Core Elements* module is the basic module and contains three mandatory elements required:

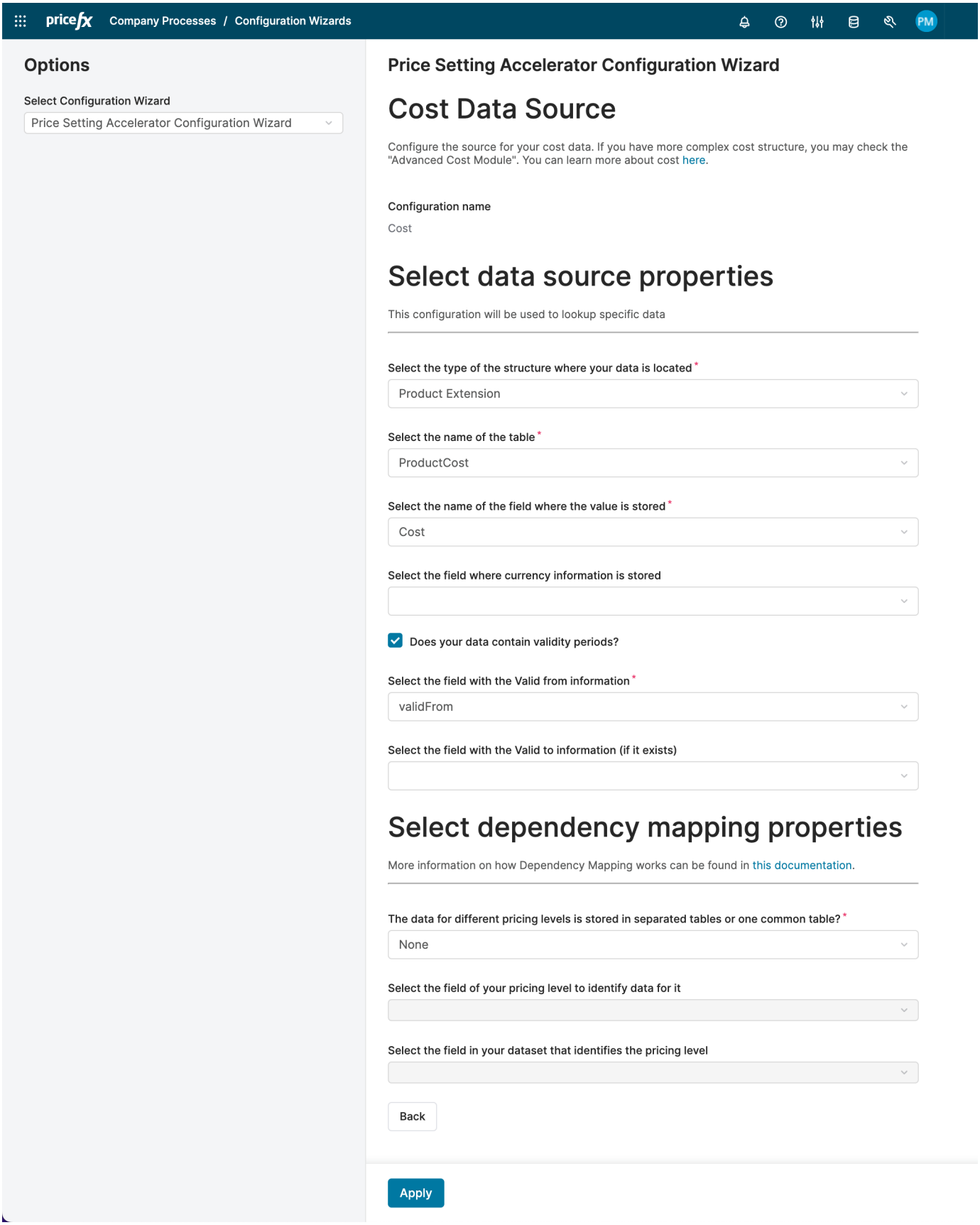
1. Configure Cost Data Source: From the Product Extension data
2. Configure Actual Price Data Source: We can set the source from several places:
    a. The last approved price or last price in the Live Price Grid
    b. or a specified price list
    c. or in some product extension for initial uploads
    d. or sync with ERP systems
3. Configure Stock Data Source

> Configuration of the various look-ups in the pricing package is similar. For the Cost Data Source, we only support Product Extensions. However, for other elements, we support product master; customer master; price parameters; data source; and data marts, depending on what kind of information it is.

Click on **Configure Data Source**

- We select Product Extension from the look-up
- Then the name of the table. This will list all tables in the product extensions and we want to pick the product cost table.
- Then the name of the field where the value is stored, in this case the cost.
- Then currency information. We don't have currency information here, so we can skip this.
- Then we have validity periods. We have a valid from, which is mandatory, then valid to is optional. It will take always the latest valid record.
- Then we have dependency mapping. Dependency mapping indicates how the data source is linked to the dependency, so the pricing level. In our case the region and countries.

When that's done click to **Apply**.



Once the configuration has been successfully applied, the product cost should be in the Live Price Grid. Return to Price Grid select the first three products again. Hit the **Calculate** button, and it brings us the cost. Let's look at the first product.

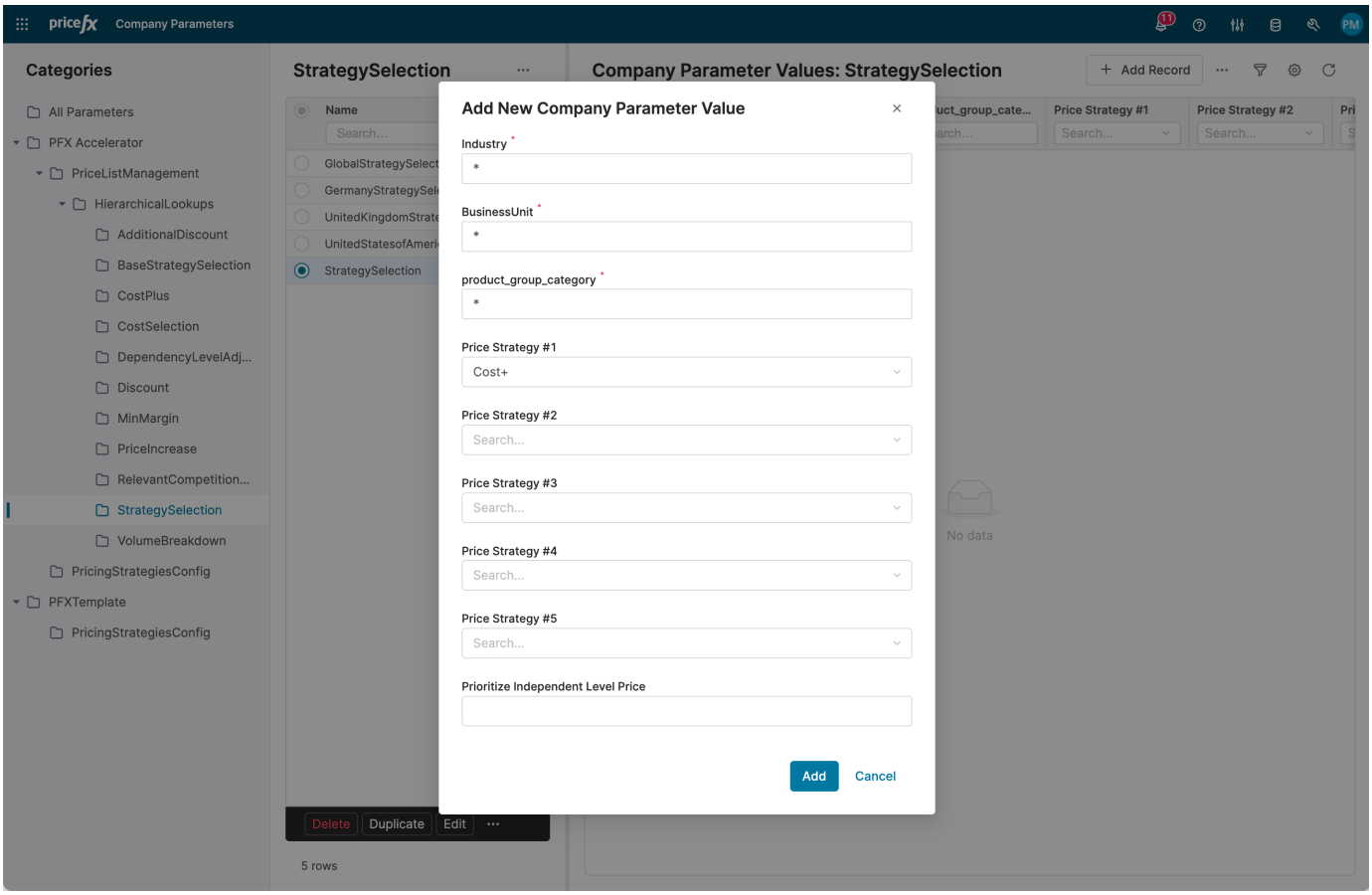
We can see the cost and if we want to double check, we can review the price and compare it to the product price in the product extension table, to ensure it's come across correctly.

Strategy Selection

Navigate to **Company Parameters > PFX Accelerator > PriceListManagement** .
There are different parameters in the sub-folders. The sub-folders relate to the different strategies of the price setting package. If you don't need some of them you can hide them or delete them.
To set parameters for your Pricing Strategy, continue to navigate within the *PriceListManagement* folder to **HierarchicalLookups > StrategySelection** to open the Strategy Selection Parameters.
Now we can see a list of several *StrategySelection* tables, named based on the dependency levels.
We'll start with the one at called *StrategySelection*, because we want to set up a strategy common for all levels. By selecting it, a second window opens to the right, where we can define up to five pricing strategies per combination of our product attributes identified by the business - in our case Industry, Business Unit and Product Group (product_group_category).
The remaining strategies are one for each pricing level or dependency level we have. There is one for the *Global* region and one for each of the countries.
The package starts with the full set and you can decide and choose what to use or not.
The *StrategySelection* parameter will be always the fall back when there is no data for a country or region specified.
However, you can configure your pricing strategy on the country or the region level or the global fall back level.

> For testing, we suggest starting with the generic StrategySelection parameters, experiment with it in combination with region and country specific ones. In customer projects, check the requirements to determine if all of them are required. If not, you can remove them.

Now let's add a record to *StrategySelection*, by clicking on the **+Add Record** button on the right of the screen. We will say for every product group; every product line; every product type and we start with a cost plus price strategy, for this example.
Click **Add** to update the default StrategySelection Parameters.

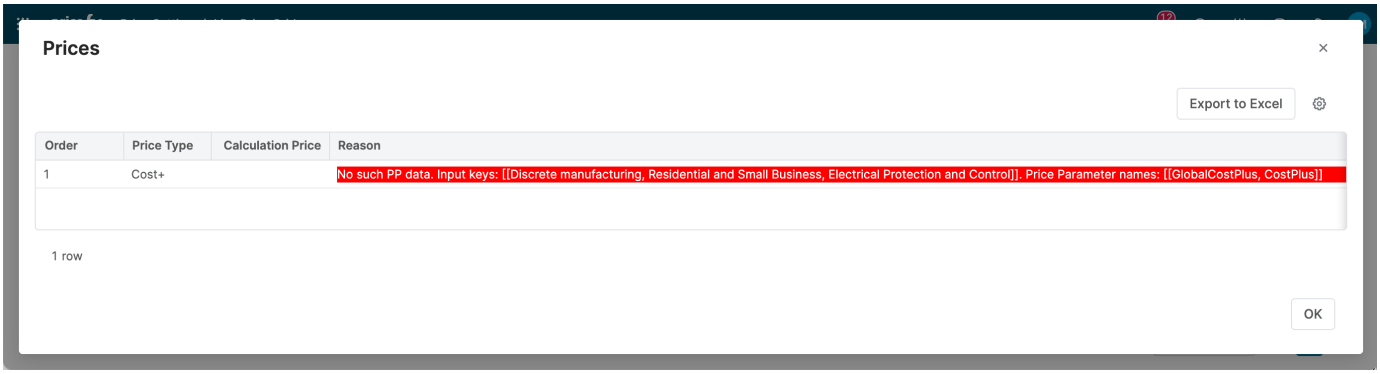


Recalculate your price grid and review the errors.

Add the "plus"

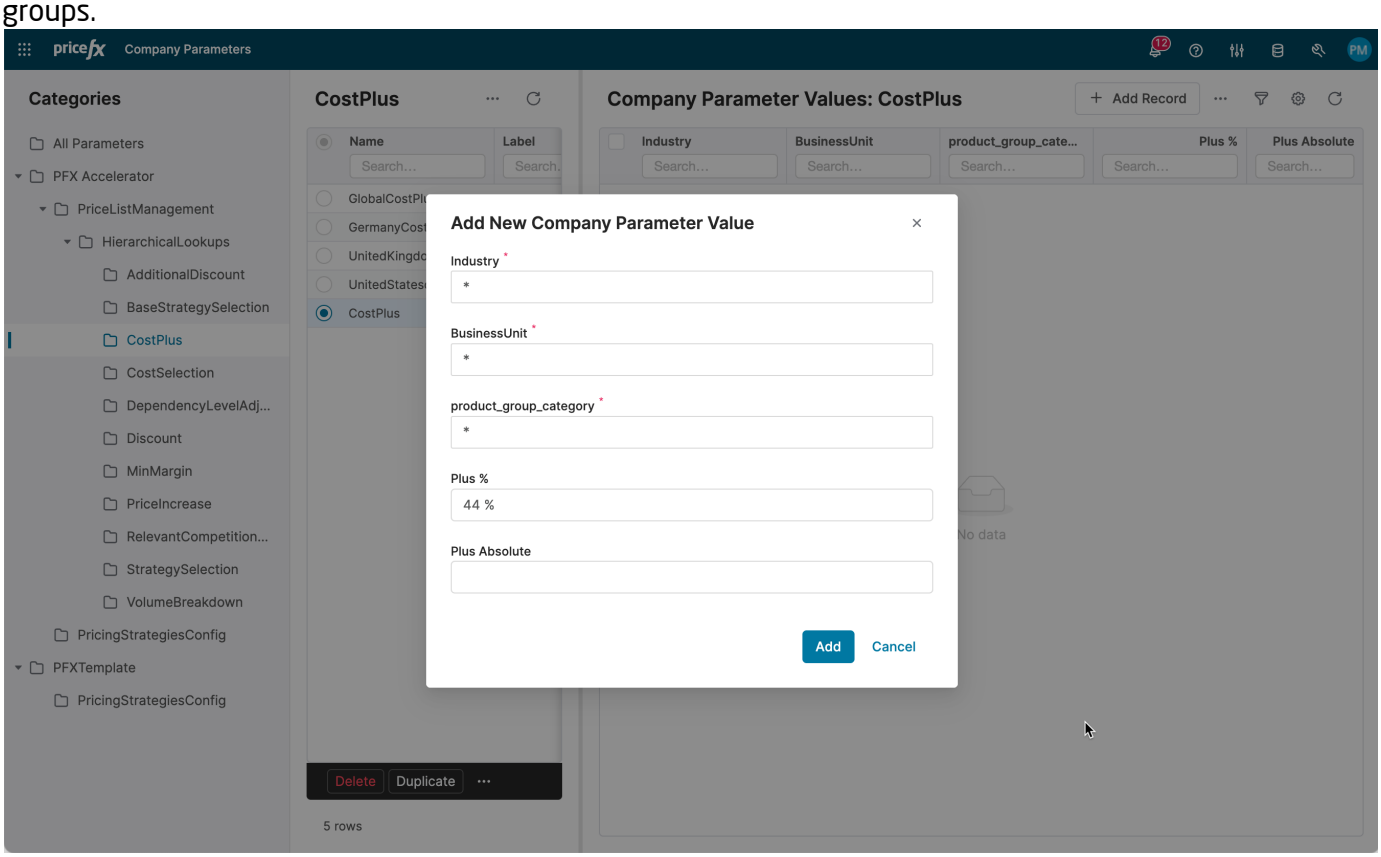
Add the Plus for All Products

We're now pretty close to having the first price calculated, but it's still not calculated and this is because there is no plus data. We have told the system what the cost is, but haven't yet told it what to add to the cost price to give us a Cost Plus price for our Price Grid.
If we want to figure out what we're missing, we have to open the price grid, and click on link **Show** in the *Prices* column of an item.



Now let's look at this error. It's saying no such PP (Price Parameter, now called Company Parameter) data row found for the combination of product attributes Industry, Business Unit and Product Group. This is the look-up key reconfigured at the beginning, and it wants to check in two price parameters: the first is the *GlobalCostPlus* and the second is the *CostPlus* so it's trying to look for the region (dependency level) specific company parameter and then for the general fallback price parameter with this look up here. This is hierarchical, so it will look for the complete key, then the key without the last lookup key and without the last part, so when going back to * * * which is the most generic one.

The error is therefore telling us we need to go to the *CostPlus* company parameter and enter a value and that should do the trick.
To do this, return to company parameters.
Under the *Hierarchical Lookups*, we now located the *CostPlus* and we click on that and see the various *Cost Plus* company parameters. This is like the *StrategySelection* parameters that we configured earlier, except this is specific to the CostPlus strategy only.
Again, we have one for the regions, one for the countries we want to use the generic CostPlus option and add our parameter here. Let's add 44% plus across all product industries, business units and product groups.



Going back to the Live Price Grid, checking first three products and hitting the **Calculate** button, we can see we have cost plus prices in here. Now more of the columns contain prices. We can say we have a final price, we have a price decision, and we have a margin.

Adding the Plus for specific products

However, most times, different industries, business units or product groups may attract a different Plus percentage. Let's look at the industry called *Discrete manufacturing*. Copy this name and return to the *Cost Plus* parameter and click on **+ Add Record**, paste the industry name and instead of 44% we want the Plus% to be 55%. Click on **Add** and we have a second record here that will now apply 55% Plus to all products within the industry *Discrete manufacturing*.
Return to the Live Price Grid, and Recalculate the first three products and we can see that the prices have changed for those Products in the *Discrete manufacturing* industry.
Now that we know the configuration that we have created is working we can recalculate the entire price grid.

Finalize the Price Grid

Calculating the Final Prices in the Price Grid

And now for almost all the product we have the pricing information we were expecting.

If there are products without price or cost, we can now investigate whether the product data extensions have cost data.

Creating a Preferred View

Before you present the final results to your stakeholders, it's also worth taking a few minutes to configure a preferred view that focuses on the information that is more relevant.
You can do this using the settings icon in the top right. Choose Select Fields to Display and you're presented with all the fields in the Live Price Grid Table. You can now work through them and select to display on those that your stakeholders expect to see.
Once you've selected the fields, don't forget to return to the settings icon and click on **Save as New Preference** and give it a name, so that you can return to it at any time.

Conclusion

You now know how to create a Price List or Live Price Grid, for a Cost Plus pricing strategy, using the Pricefx Price Setting Accelerator Package.

Competition Based Pricing

Table of Contents

Outcome

At the end of this tutorial, you should be able to:

- configure a Live Price Grid, which will utilize the Price Settings accelerator for calculations of the prices using the Competition Based strategy
- calculate the prices
- review the calculation errors from the accelerator.

Prerequisites

Training partition with:

- installed Price Setting accelerator.
- demo product data

Business Decisions

Based on the use case requirements, we should set up following capability:

> *A pricing strategy that considers competitor pricing, along with the ability to prioritize certain competitors over others. The hierarchy for the competitors is 1)ABB; 2)Legrand; 3) Eaton*

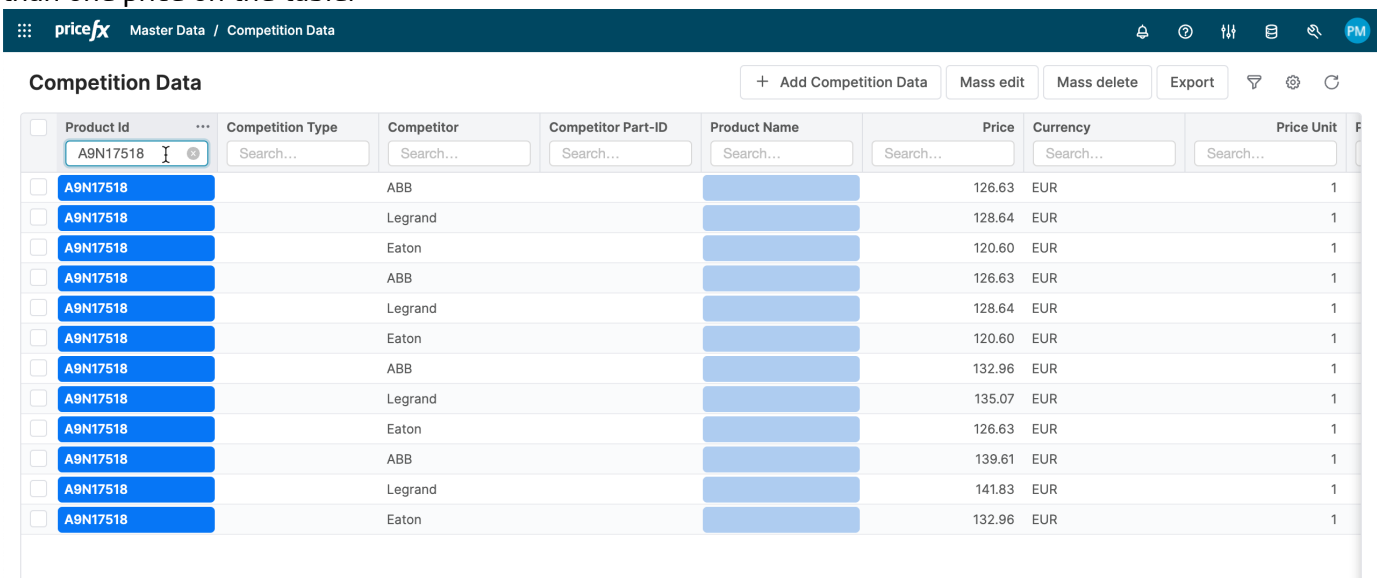
So we will need:

- to set up a Competition Based strategy
- an independent price list for the global prices. Since we already have independent global LPG configured, we will use it.

Review the Competition Data

We can view the entire competition data set within the *Competition Data* table, via the *Master Data* menu.

If we search for a specific Product ID, we're presented with all the competition data and we can see the most recent data. The system will automatically take only the newest price, where a competitor has more than one price on the table.



Product Id	Competition Type	Competitor	Competitor Part-ID	Product Name	Price	Currency	Price Unit
A9N17518		ABB			126.63	EUR	1
A9N17518		Legrand			128.64	EUR	1
A9N17518		Eaton			120.60	EUR	1
A9N17518		ABB			126.63	EUR	1
A9N17518		Legrand			128.64	EUR	1
A9N17518		Eaton			120.60	EUR	1
A9N17518		ABB			132.96	EUR	1
A9N17518		Legrand			135.07	EUR	1
A9N17518		Eaton			126.63	EUR	1
A9N17518		ABB			139.61	EUR	1
A9N17518		Legrand			141.83	EUR	1
A9N17518		Eaton			132.96	EUR	1

Configure the Product Competition Module

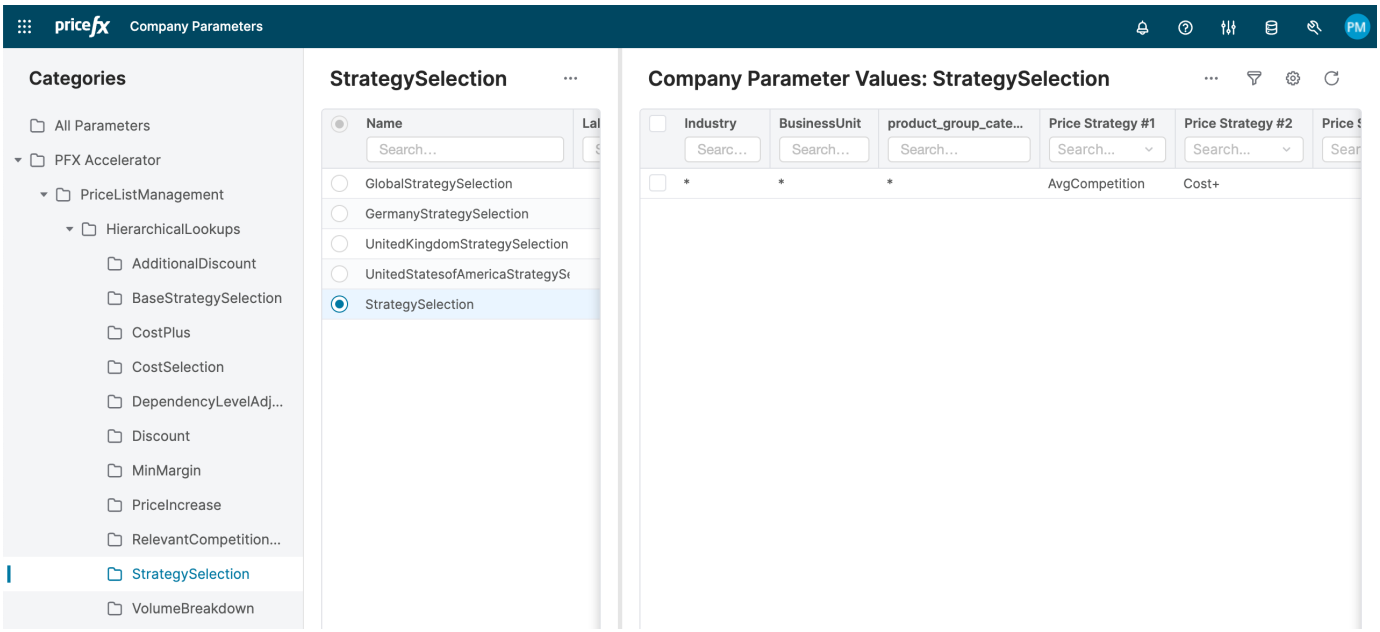
To set up a competition-based pricing strategy, we need to enable the *Product Competition* module in the configuration wizard.
Navigate to **Company Processes › Price Setting Accelerator Configuration Wizard** and select the *Product Competition* module from the dropdown below the table. Select **Configure Selected Module** and Turn it ON and **Update Module Status**.
It is enabled and will pull competition data into the Live Price Grid.
It will still require configuring, so go to configuration of the *Product Competition* module again.
Select **Configure Dependency Mapping** and we'll set it *None*, so there's currently no dependency. Click **Apply**.
We can now return to our Global Live Price Grid, select the first three products and click **Calculate**.
And we can now see that they all three have competition data available in the *Competition Data* column. If we click on link **Show**, a separate window opens with the details of the competitors, their prices, the types of prices and when the price is valid from.

> Competition Data will only be displayed for products where competition data is available in the Competition Data Table

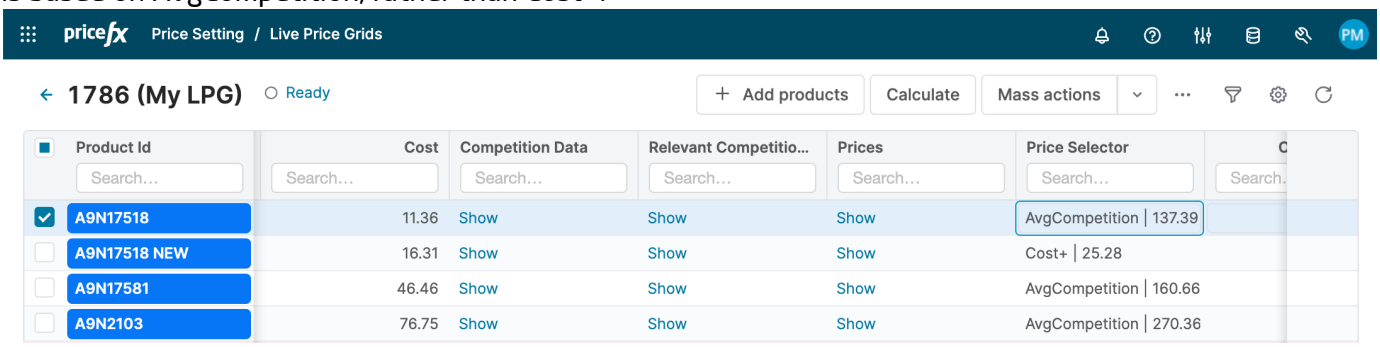
Now the competition data are looked up, but not yet used for the calculation of prices.

Set Competition Strategy Priority

If we wanted to ensure that our Competition Based pricing dtrategy had priority over the Cost+ strategy, we could set that up too. This would enable you to ensure that whenever a product has competition data available in the Competition Data table, it would use it, but for any products with no competition data, we would calculate the price using the Cost+ parameters.
To do this, navigate to **Company Parameters › PFX Accelerator › PriceListManagement › HierarchicalLookups › StrategySelection** and select the *StrategySelection* default strategy from the list.
Now on the right-hand side of the page in *Company Parameter Values*, we change value of *Price Strategy #1* from *Cost+* to *AvgCompetion* and we insert *Cost+* into value of field *Price Strategy #2*.



This will now apply to our Live Price Grid when we recalculate the prices.
If we recalculate just the first price, we can see that the *Price Selector* column now tells us that the price is based on AvgCompetition, rather than *Cost+*.



And if we click on **Show** in the *Prices* column, it provides us with information for not only the competition-based information but also the cost+ price.
We can also see a reason for the price. Here there is a Price corridor between the highest and lowest competitor price and we're at 50%, so in the center of the pricing. But how did the system determine that we wanted to set a price at the 50% point between the highest and lowest competition price?

Understanding Competition Strategy Definitions

Competition Based Pricing comprises a range of strategy definitions that will influence the calculation, and we'll now look at how we can influence these.
To see the different competition based definitions available, let's begin by adding them to the strategic priority.
To do this, navigate to **Company Parameters › PFX Accelerator › PriceListManagement › HierarchicalLookups › StrategySelection** and select the *StrategySelection* default strategy from the list.
On the right-hand side of the page in *Company Parameter Values*, we add *MinCompetition* to column *Price Strategy #2*, and we insert *MaxCompetition* into column *Price Strategy #3*.
This will now apply to our Live Price Grid when we recalculate the prices.
Return to your live price grid and if we recalculate just the first price, and click on **Show** in the *Prices* column, the pop-up window now tells us the various prices based on all three competition based strategies.

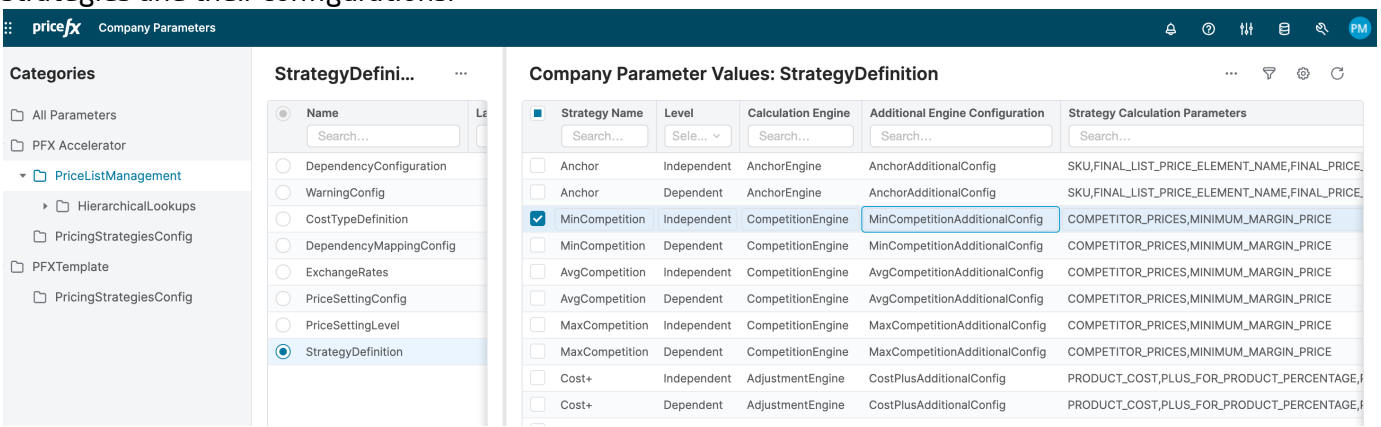
AvgCompetition = Our price is the average (50%) between the highest and lowest competitor price

MaxCompetion = Our price is the same as the highest (maximum) price across our competitors
MinCompetition = Our price is the same as the lowest (minimum) price across our competitors

These are the default system configurations and a good starting point for your implementation, however the solution is far more flexible, and can be configured to enable a greater range of strategies. For instance, instead of just an average (50%) price, you may want the price to fall within the 40% or perhaps 60% point between your competitors. You may want to be cheaper or more expensive than a competitor by a % point, or perhaps by a fixed currency amount.
These options are all possible via the *Competition Engine*.

The Competition Engine

The competition engine is an advanced configuration component of the Price Setting accelerator.
To locate it, navigate to **Company Parameters › PFX Accelerator › PriceListManagement** and find the *StrategyDefinition* parameter.
If you select it, the *Company Parameter Values* appears on the right and you'll see all the pre-built pricing strategies and their configurations.



You'll notice that for each strategy definition, there are two entries. One is for the *Dependent* level calculation and the other for *Independent* level calculation. They are configured the same, but this allows you to configure different rules for different levels within your pricing strategy. For instance, you may configure competition based pricing for country level (Dependent) calculations, but not for global (independent) calculations.
Let's make some changes to *MinCompetition* as an example.
If we begin by looking at the various columns in the table, we can see:

- the *Calculation Engine* being used and in this case we can see that we're using the *Competition_Engine* , which is dedicated to all our competition based calculations.
- *Additional Engine Configuration*, which tells us the name of the company parameter, that this strategy is checking against as it executes its calculation
- the *Strategy Calculation Parameter* which we'll leave as they are for the moment.

Let's focus on the *Additional Engine Configuration*. Copy the name found in this column, called *MinCompetitionAdditionalConfig* and on the left hand side we're going to search for the company parameter with that name. Navigate to **Company Parameters › All Parameters** and in the list of parameters search by Name for "MinCompetitionAdditionalConfig". Once found, look at the values stored inside.

Review Configuration Option

All the various configuration options for the Competition Engine can be found in the Competition Engine documentation of Price Setting accelerator.
Here we'll show just a few piece of the information available:

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`

Configure the Competition Engine

Using the configuration options we can now make changes to the *Company Parameter Values* for *MinCompetitionAdditionalConfig*.
We can change our *Competitor Position* to be `min+1` if we wanted our price to be the same as the second (min +1) cheapest competitor.
We can change the *Repositioning %*, to be `-10%` against the second-cheapest competitor.
We can do an absolute repositioning (*Repositioning Abs*) to reduce the absolut price by a number. So `- 1` would give use the second cheapest price, `-$1` or `+1` would give us the second cheapest price + $1.
You can now make changes to these parameters and view the impact on your live price grid to see the impact that each of these has.

Attribute Pricing

Table of Contents

Outcome

At the end of this tutorial, you should be able to:

- configure a Live Price Grid, which will utilize the Price Settings accelerator for calculations of the prices using the Attribute Based strategy
- calculate the prices
- review the calculation errors from the accelerator

Prerequisites

Training partition with:

- installed Price Setting accelerator.
- demo product data

Business Decisions

Based on the use case requirements, we should set up following capability:

> *An attribute based pricing strategy that enables the customer to differentiate prices based on the following 3 product attributes:*

- *Competitiveness*
- *Size*
- *Direct value*

Note, that some of those product attributes mentioned in business requirement do not exist, so you will add them.
So we will need:

- to set up an Attribute Based strategy
- an independent price list for the global prices. Since we already have independent global LPG configured, we will use it.

Review the Product Attributes Data

We can view the entire product data set within the *Products* table, via the *Master Data* menu.
For now, review the content of attributes/columns *Competitivness*.
The other attributes will be added during this exercise.

Introduction to Attribute Based Pricing

The idea of attribute-based pricing is to define the price based on product attributes like size, color, weight etc. It is based on the Anchor Follower approach, so we configure it to analyse the price of another SKU(anchor), then change it to get the final price.
Because of the infinite number of attributes across a range of different products and industries, this is the most complex pricing engine within Price Setting accelerator.

Creating an Attribute-Based Pricing Strategy

Adding a Strategy Definition

Begin in **Company Parameters › All Parameters** and search for *StrategyDefinition*. Select it and the Company Parameter Values are displayed on the right.
We can see all the strategies defined in the package on implementation. There is no Attribute Based pricing strategy, so we need to create it.
Click on **+ Add Record** and a pop-up window appears for you to set up the basic configuration.
Add a Strategy Name in this case we'll call it `AttributeTest`.
Select *Level* as `Independent` calculation level as our Live Price Grid is *Global*.
The *Calculation Engine* is the `AttributeBasedEngine` calculation engine (configuration documentation is available here)
We need to create a parameter for the *Additional Engine Configuration*, but here we can call it `Attribute BasedTest` and we'll create it using this name as we progress.
We can take the *Strategy Calculation Parameters* from the Default Strategy Calculation Parameters in the documentation - here is an example to be used here: `SKU,FINAL_LIST_PRICE_ELEMENT_NAME, FINAL_PRICE_ELEMENT_NAME,DEPENDENCY_PROPERTIES,STRATEGY_NAME`
You can leave the remaining fields blank.
Click **Add** to add it to the list and we'll see it appear with the other Strategy Definitions.

Adding a Strategy Parameter

Begin in **Company Parameters › All Parameters** and search for parameter named *StrategySelection*. Select it and the Company Parameter Values are displayed on the right and we can see the configuration we created for our Competition Based example.
Let's replace `AvgCompetition` and replace it with our `AttributeTest` strategy.
Return to the Live Price Grid and calculate the first 4 products and when we click on link **Show** under the *P rices* column, we can see an error, which shows that there are no parameters set.
These parameters relate to the *Additional Engine Configuration* we named as *AttributeBasedTest*. So let's set up and configure a new company parameter with this name.
Navigate to the **Company Parameters › PFX Accelerator › PricingStrategiesConfig** and you can now click on **+ Add Parameter** to add a new company parameter with following settings:

Name - AttributeBasedTest
Label - Attribute Based Test
Valid After - a day in the past (anything older than today)
Table Type - MATRIX
Value Type - MATRIX
Status - Active
Click **Add**.
We now need to add the fields as defined in the documentation.
To begin with, in the *Company Parameter Values*, select the settings icon and click on *Select Fields to Display* and select *Attribute 1* and **Apply**. This adds a column next to the name column for us to add information.
We add the fields and their attributes by clicking on **+ Add Record**, and populating the information as per the table below. (up-to-date configuration documentation is available here)

Name	Attribute 1
Source Type	PP
Source Name	AnchorData
Anchor Field Name	Anchor
SKU Field Label	SKU

Return to the Live Price Grid and calculate the first 4 products. When we click on link **Show** under the *Prices* column, we can see a new error, which states that the Anchor Data is missing. This means that we have not yet set any anchors for our products. So let's create an example.

Creating an Anchor Product

To create an example for our purpose, select the SKU from the first product in the Live Price Grid and copy it.
Return to Company Parameters and search for the *AnchorData* parameter and select it.
On the right, select **+ Add Record** and we set:
SKU - the ID of the first SKU from the price grid
Achor - the ID of the third SKU from the price grid
Click **Add** and return the Live Price Grid and recalculate the first product.
When the recalculation completes, we can click on the *Prices* column, and we can see the next error, which states that the Attribute Pricing rules for our strategy name has not been found, and so we can now begin to add these rules around specific attributes.

Creating Attributes and Rules

There are three distinct attribute types we can configure. These are:

1. **Value** based, such as *High*, *Medium* and *Low* with each representing a number.
2. **Interval** based, such as a number range from 1 to 5 representing a number, 6 - 10 representing a different number etc..
3. **Direct Value** based is a value that is used as it is, so in our example we'll set the *Direct Value* to $3 and when the price is calculated it will add $3 to the price after other calculations have completed.

We will create an example of each of these for our demonstration.
To locate the attribute parameter settings, return to Company Parameters and search for `attribute`, and you'll see a set of parameters related to *attributes* appear. Select *PricingAttributes*.

Example 1: Value Based, for attribute Competitiveness

Add a record to the company parameter *PricingAttributes*, with following fields values:
Pricing Attribute - Competitiveness

Type – Value
Source Type – P
Source Field – competitiveness (name of the field in the "P" - Products - table)
<other fields> – <blank>
So we have now created a *Competitiveness* parameter and now need to tell the system, what the impact is, if we set the product parameter to *High*, *Medium* or *Low* values.
Add 3 records to the company parameter *ValueAttributesConversion*:

Pricing Attribute	Pricing Attribute Value	Price Impact Value
Competitiveness	High	-1
Competitiveness	Medium	0
Competitiveness	Low	1

Add a record to company parameter *AttributeBasedPricingRules*:
Operator #1 – + (to add)
Pricing Attribute #1 – Competitiveness
Name – AttributeTest (this is the strategy name it will look up)
<other fields> – <blank>
Return to your Live Price Grid and calculate the first price again. When it is ready you can click on link **Show** in the *Prices* column and you'll see *AttributeTest* as the *Price Type* with the calculation price. The price now shown is $1 less than the price of the Anchor product we set earlier. This is because we set our competitiveness for this product to *High* and the parameter we created for *High* says to -$1 to the price of the anchor product.
We can see that if we were to change the competitiveness to *Medium* or *Low* and recalculated the price, it would change in accordance with our parameters.

Example 2: Direct Value Based

Add additional attribute column to a Product table

To make the hidden attribute column visible, navigate to **Master Data > Products**, use icon **Show Table Settings** and select **Select Fields to Display**. These are columns are designed to add additional product attributes. In the list of attributes, check one, which is not used yet, and **Apply**.
To rename this attribute, right-click on the header of the column, and select **Rename and Customize Column**. In the dialog, do following settings:
Name – DirectNumber
Label – Direct
Type – Real
Format Type – Numeric
Click **Add** to finish the attribute column configuration.
For the first product (or the product that we're working with in the Live Price Grid) set the *DirectNumber* to 3.

Add new pricing attribute

Add new record to company parameter *PricingAttributes*, with values:
Pricing Attribute – DirectNumber
Type – Direct Value
Source Type – P
Source Field – DirectNumber (name of the field in the "P" - Products - table)
<other fields> – <blank>
So we have now created a *DirectNumber* pricing parameter and now need to tell the system what the impact is of our product related settings for this parameter.

Adding AttributeBasedPricingRules

Edit the record we've added to the company parameter *AttributeBasedPricingRules*
Add the following information directly into the fields:
Operator #2 – + (to add)
Pricing Attribute #2 – DirectNumber (the name of the pricing attribute we've created in the previous step)
Return to your Live Price Grid and calculate the first product again. When it is ready you can click on **Show** in the *Prices* column and you'll see *AttributeTest* as the *Price Type* with the calculation price. The price now shown is $4 more than the price of the Anchor product we set earlier.
This is because we set our competitiveness for this product to *High* and the parameter we created for *High* says to `-$1` to the price of the anchor product, additionally we have an additional setting to add a direct value of $3 to the price.

Example 3: Interval Based, for attribute Size

Add additional attribute column to a Product table

To make the hidden attribute column visible, navigate to **Master Data > Products**, use icon Show Table Settings and select **Select Fields to Display**. These are columns are designed to add additional product attributes. In the list of attributes, check one, which is not used yet, and **Apply**.
To rename this attribute, right-click on the header of the column, and select **Rename and Customize Column**. In the dialog, do following settings:
Name – Size
Label – Size
Type – Real
Format Type – Numeric
Click **Add** to finish the attribute column configuration.
For the first product (or the product that we're working with in the Live Price Grid) set the *Size* to `10`.

Add new pricing attribute

Add new record to company parameter *PricingAttributes*, with values:
Pricing Attribute – Size
Type – Interval
Source Type – P
Source Field – Size (name of the field in the "P" - Products - table)
<other fields> – <blank>
So we have now created a *Size* pricing parameter and now need to tell the system what the impact is of our product related settings for this parameter.

Adding IntervalAttributesConversion

Add new records to company parameter *IntervalAttributesConversion*, with values:

Pricing Attribute	Pricing Attribute Valid From	Pricing Attribute Valid To	Price Impact Value
Size	0	5	1.1
Size	6	14	1.2

Adding AttributeBasedPricingRules

Edit the record we've added earlier to the company parameter *AttributeBasedPricingRules*
Add the following information directly into the fields:
Operator #3 –

- (to multiply)

Pricing Attribute #3 – Size (the name of the pricing attribute we've created in the previous step)

Return to your Live Price Grid and calculate the first product again. When it is ready you can click on **Show** in the *Prices* column and you'll see *AttributeTest* as the *Price Type* with the calculation price.
The price now shown is 20% more than the price before.
This is how the price of our product is calculated now:

1. we start with price of the *anchor product*
2. because our competitiveness for this product to *High* and the pricing parameter we created for *High* says to -$1, the price will be lowered by 1
3. because of the *DirectValue* pricing parameter, a direct value of $3 will be added to the price
4. because we set our *Size* attribute for this product to 10 and the pricing parameter we created for size 6 to 14 says to * (multiply) the price by 1.2 (or 20%), the price will be multiplied by 1.2.

Dependent and Independent Pricing

Table of Contents

Outcome

At the end of this tutorial, you should be able to:

- configure a global Live Price Grid and also country-specific price grids, where the system is configured to cross-reference prices across a range of dependencies to produce the required results.

Prerequisites

Training partition with:

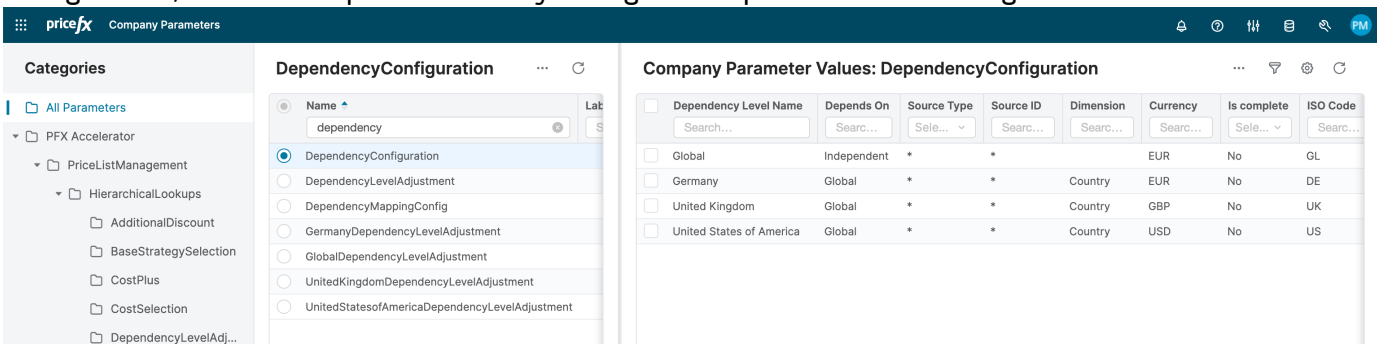
- installed Price Setting accelerator.
- demo product data

Introduction to Dependencies

For many organizations, there is a need to create discreet price lists for different regions, countries and even for different stores within the same country. Dependency mapping and configuration allows you to configure an independent list and to the configure price lists that use the information, but with their own market specific settings.

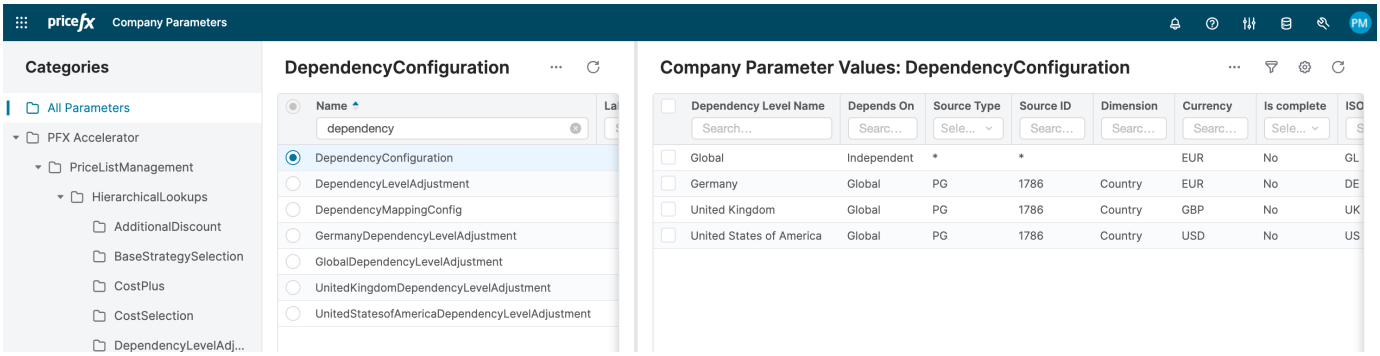
Dependency Configuration

Dependency configurations can be found in Company Parameter *DependencyConfiguration*. It holds the configuration, which was uploaded already during one step of the Price Setting Accelerator installation.



The list of values shows, that there are 4 configurations: Global, Germany, United Kingdom, United States of America.

The column labelled *Depends On*, tells the system which level it is dependent on. In this case the *Independent* level we want to use is called `Global` and therefore we need to ensure that for all three countries the *Depends On* column is named as `Global` (or matches the *Independent* level exactly, in case is named differently).
You also have the *Source Type* (PL/PG/...) and *Source ID* (ID of pricelist) to identify the price list on which this one will depend on. The *Global* row is *Independent* and so will not require a *Source ID* or *Source Type*, but the other rows will.



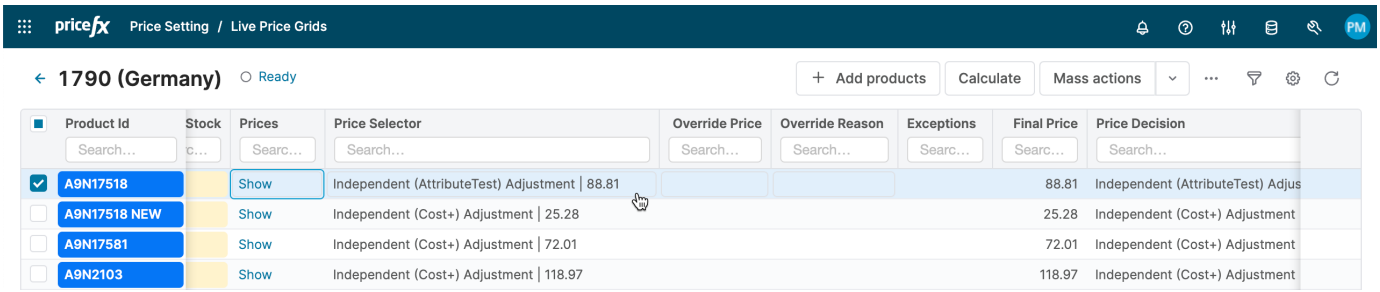
The *Source Type* will be the type of List (Price Grid or Live Price Grid) that we are using for the global level, so in this case it is the Live Price Grid (PG) that we complete in this field.
The *Source ID* is the Price Grid ID then for the Live Price Grid. In our example, it is 1786, but you will have it different.
Find the ID for your Live Price Grid and complete these two fields for the countries Germany, United Kingdom and United States of America.
Once that is configured, we need to create the individual price lists or price grids for each country.

Create the Dependent Live Price Grid

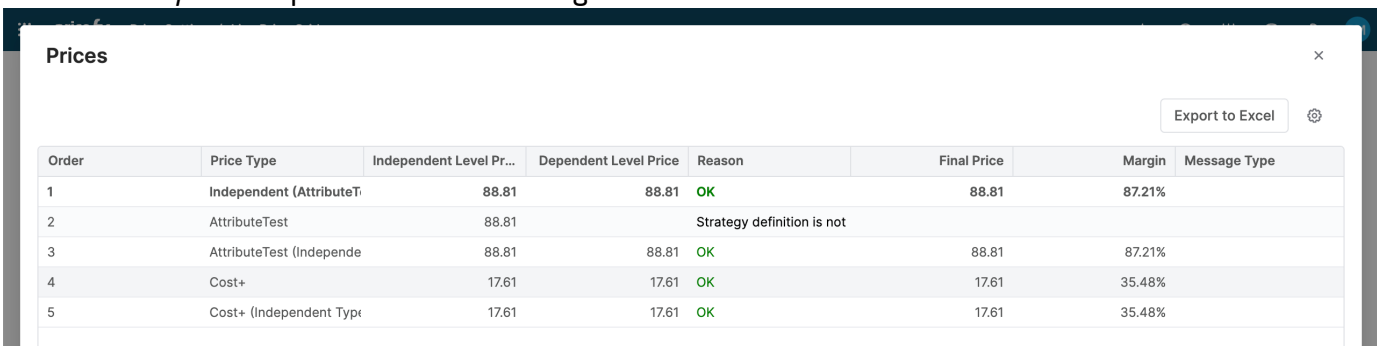
Repeat the following instructions for all countries.
Navigate to **Price Setting › Live Price Grid**
Use action **+ New Price Grid** and select *(default)* type.

1. Define Live Price Grid
   - Label: <give it name based on the country name>
   - Type: SIMPLE
2. Select Products
   - use button **Select All** to select all products
3. Set Parameters
   - Allow column type change
   - Default pricing logic: *DependentPriceListLogic* (this is the logic provided by Price Setting accelerator). Local price lists will have the dependent price list logic, which allows us to select an independent price list as the starting point in our pricing calculations.
   - Result Price: *Final List Price*
   - Dependent Level Name: <country name>. Because we selected the dependent level pricing logic, we have to select a dependency level. We select the configuration that matches the country we set up earlier.
   - Output Elements: <select all>
   - Default fields: <hide all in item list>

Hit **Save**
In the list of Live Price Grids, select the one you just created, and use action **Calculate**.
Now we are ready and we can enter the Live Price Grid.



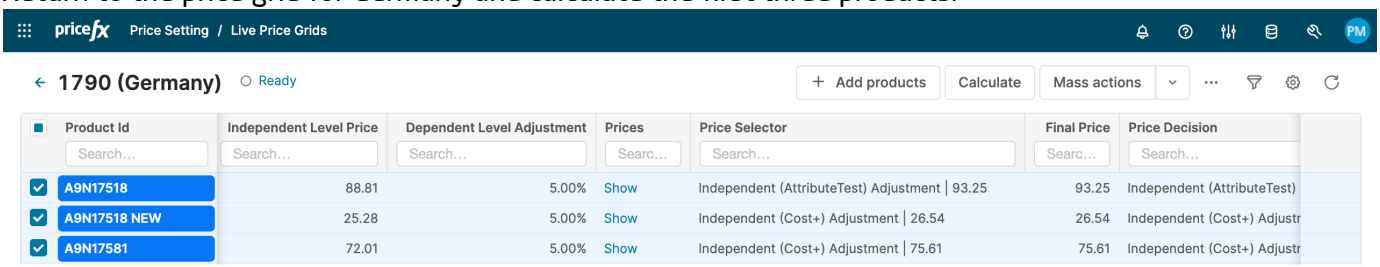
If we locate the *Price Selector* column and sort to find a product with the "Independent (AttributeTest) Adjustment" price, and then click on **Show** in the *Prices* column to open the pop up with the various prices. We can see all the strategies available so far and we can see that the price that has been selected comes from the *Independent* price list with no change.



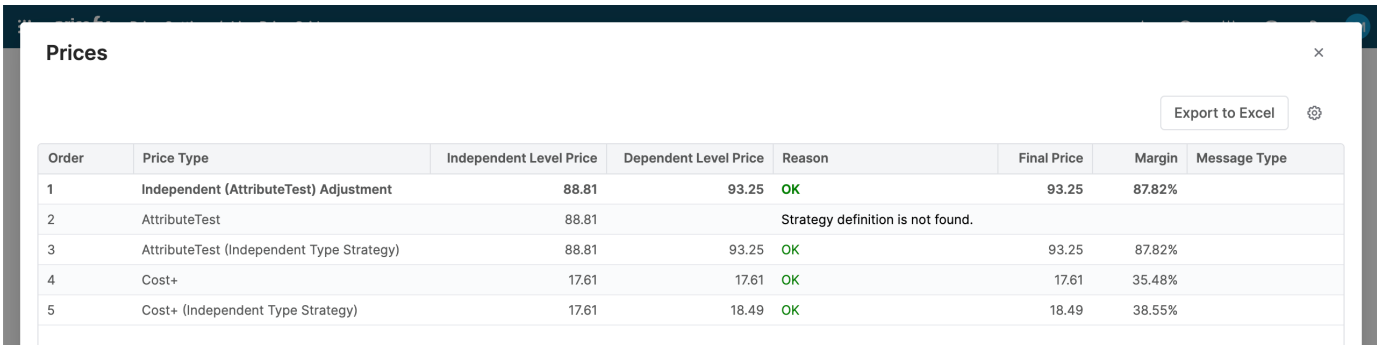
If we wanted the prices to differ, we should configure the *Dependency Level Adjustments* to tell the system what adjustments to make to the prices.

Configuring Dependency Level Adjustments

Navigate to **Company Parameter › PFX Accelerator › PriceListManagement › Hierarchical Lookups › DependencyLevelAdjustments**.
We're now presented with company parameters holding adjustments for the various Dependency Levels we've set up, so in this case Global, Germany, United Kingdom, Unites States of America and the base configuration called *DependencyLevelAdjustments*.
To add adjustments to any level, we need to open select appropriate company parameter and add some values.
Select *GermanyDependencyLevelAdjustment*.
Add new record with following settings:
Industry - *
Business Unit - *
Product Group (product_group_category) - *
Adjustment % - 5%
Return to the price grid for Germany and calculate the first three products.



We can now observe in the *Dependent Level Adjustment* column that there is a 5% adjustment applied and if we click on **Show** in the *Prices* column, we can observe that there is a difference between the *Independent Level Price* and the *Dependent Level Price*.



We can also see the priority of the pricing strategies based on the order of the results. The highest priority that returned a result (in this case *AttributeTest* strategy) is presented at the top with the final price and the pricing for the other strategies are also returned and shown here, but are not displayed as a final price.

Prioritizing the Independent Level Strategy

When calculating dependent price list, then by default the first strategy used is *Independent Level Adjusted Price*. It is calculated as a Final Price from the independent price list adjusted by the dependency level adjustment.
For details, see the accelerator documentation about Dependent Price List Calculations.
However, if you wanted to alter this behavior, you could.
This can be altered in company parameter *StrategySelection* (found in menu **Company Parameters › PFX Accelerator › PriceListManagement › Hierarchical Lookups › StrategySelection**) by setting the value of column titled *Prioritize Independent Level* to no.

Custom Price Strategy Configuration

Table of Contents

Purpose

The purpose of this first walk-through is to show how the Price Setting accelerator can be customized to deliver a custom pricing strategy.

Pre-Requisites

This tutorial assumes that

- You have completed Day 1 of Price Setting accelerator training.
- 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.

Introduction

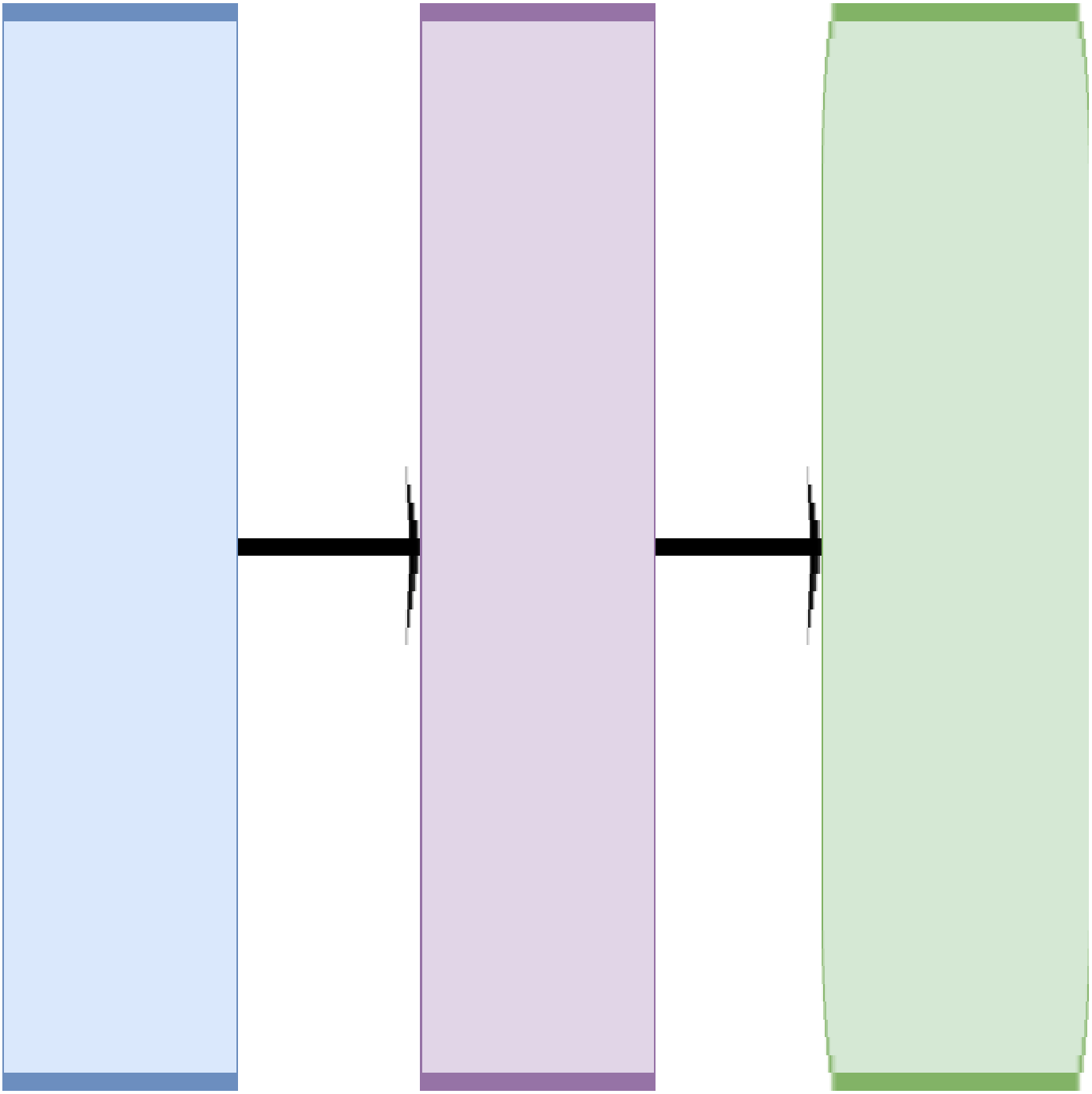
As you will remember from the first day of training, the Price Setting accelerator has a range of pre-defined pricing strategies and calculation engines to meet the standard pricing needs of many use cases, however the flexibility of being able to go even further and add custom strategies makes the Pricefx platform one of the most versatile and powerful pricing solutions on the market today.
In this training, we will walk you through the steps to build a custom calculation engine in Pricefx Studio and link it through the accelerator to run customized strategies based on the needs of your end customer.

Configuration Overview

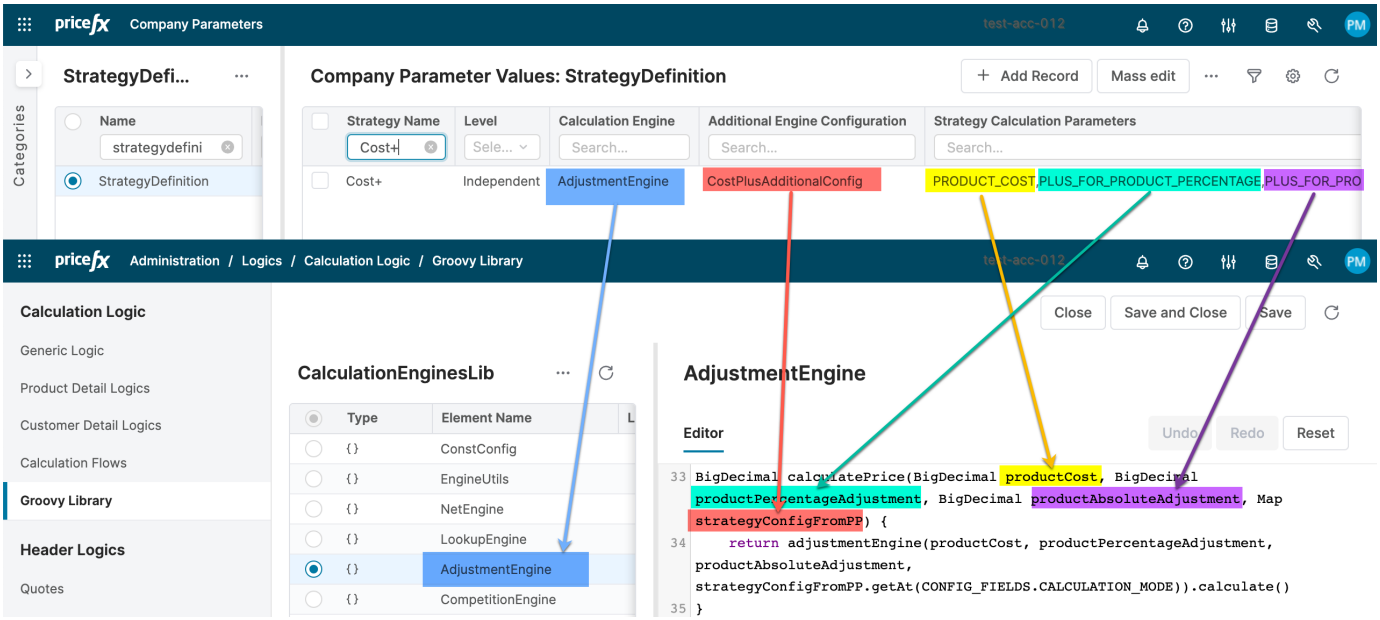
Each strategy is configured on 2 places:

- **strategy implementation** - a piece of Groovy code stored in a method in a *Library* logic. This is truth for both out-of-the-box and custom strategies. For details, see Custom Engines.
- **strategy definition** - record in company parameter *StrategyDefinition*, which defines the name and parameters to be provided when the Price Setting accelerator calls the method. For details, see Strategy Definition parameter.

Then the admin can set up the Price Setting accelerator to use the strategy for the calculations of pricelist 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 relation between the *definition* and the *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.

Create Custom Pricing Strategy

Create a New Calculation Engine Implementation

Now we can begin creating 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* logic
   - with name "TrainingCalculationEngineLib"
   - set *Valid After* to a date in the past (e.g.beginning of the year)
   - set *Status* to *Active*
2. Add new *groovy* element "TrainingEngine" with *calculatePrice()* method and very simple implementation

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

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

Define a New Pricing Strategy

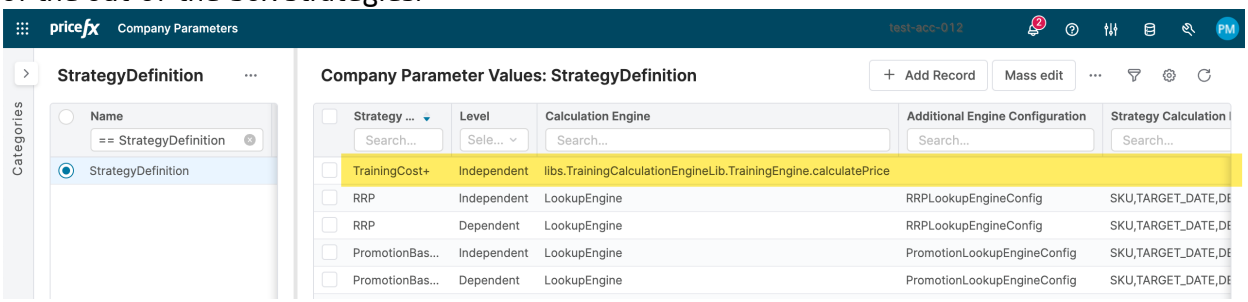
The company parameter *StrategyDefinition* contains a list of available pricing strategies that are available as standard in the price setting package, along with details about the strategic hierarchy, calculation engine and strategy calculation information.
Let's quickly explain some fields:

- 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 - *Independent* for global and *Dependent* level for regions, countries or even branches set in your hierarchy.

- Calculation Engine - The next thing is the *Calculation Engine*. This is an attribute of the Pricing Strategy and you can see we have existing Calculation Engines, like the *AnchorEngine*, *CompetitionEngine* etc.
- For more details, see Strategy Definition parameter.

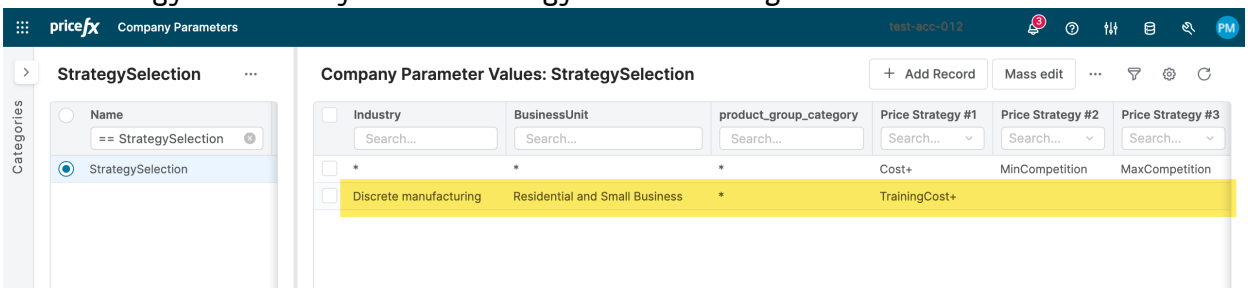
To define the new strategy:

1. Find the company parameter *StrategyDefinition*
2. Add new record to define the new strategy and link it to the new strategy implementation in groovy library:
   - Set *Strategy Name* as "TrainingCost+" (mandatory field)
   - Select the *Level* as *Independent* (mandatory field)
   - Set the *Calculation Engine* to the 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.



Linking the Pricing Strategy to Business Units or Products

1. Navigate to company parameter *StrategySelection*. Here we have the various business units and product groups as well as the various pricing strategies assigned to them.
2. Add new record for:
   - Industry *Discrete manufacturing*
   - Business Unit *Residential and Small Business*
   - set the other keys as "*"
   - Price Strategy #1 - select your new strategy named *TrainingCost+*

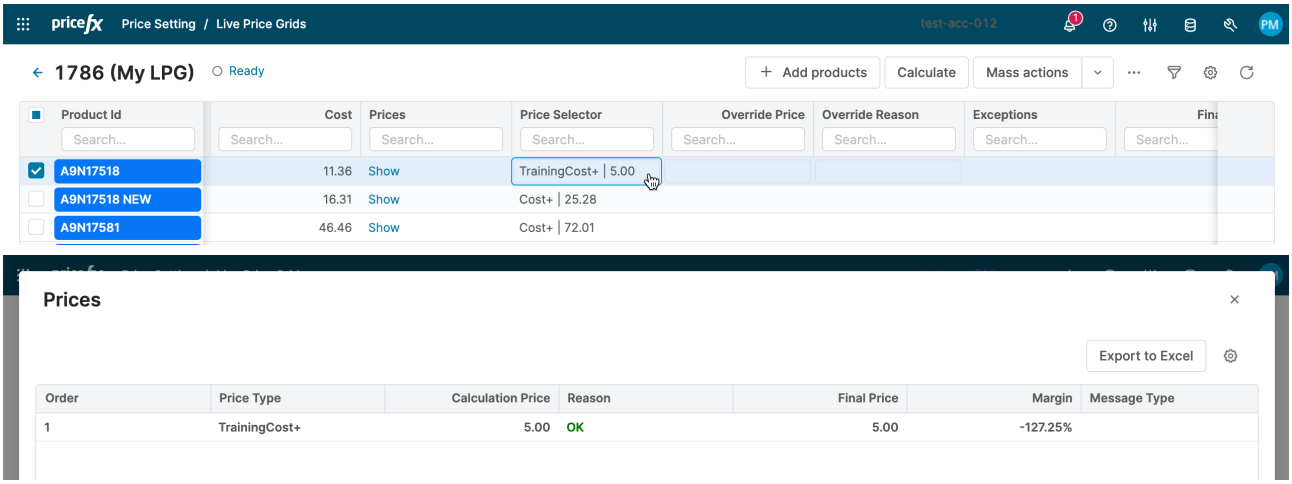


Test Calculation using the Custom Pricing Strategy

We can now calculate the result price for *Discrete manufacturing| Electrical Protection and Control*. When our pricing calculation runs for any product within that Industry and Product Group, it will take this look up key and with our new strategy to calculate a new price, which, based on the element we created, should return a price of (result) of 5.

1. Navigate to one of the live price grids created in the first part of the training and select an *Independent* one, as we defined the new strategy for *Independent* level pricing.
2. Ensure that 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, if you see the expected result, based on your implementation. Click on link *Show* in Prices field to see the result in detail.



Messaging from the calculation logic

There are two types of results we can return, you can find out more about that in the Custom Engine docs.
What we returned right now is a big decimal with a price.
When we are not able to calculate and return a price, we can report the problem from the logic:

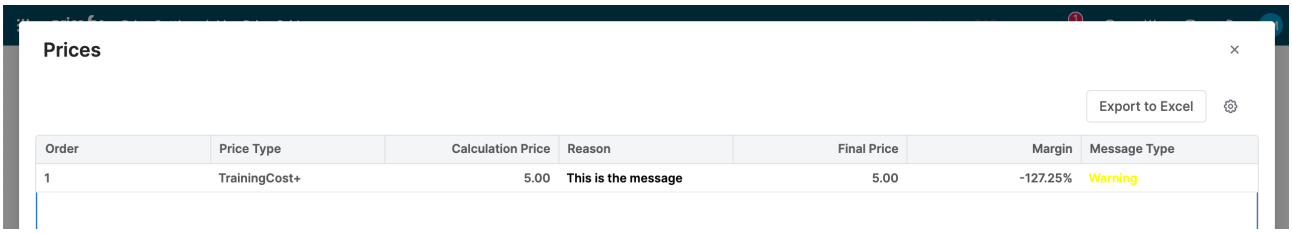
- throw an exception with an error message
  - the reason message will be displayed as white text on the red background
- return a more detailed result with message and type.
  - the reason message will show the provided message, and you will see the message type with appropriate color on the side.
  - 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 it *Map*.
   For possible message types, see Custom Engine docs.
2. Remember to deploy the modified logic.
3. Recalculate the product in the live price grid
4. Review the results in column *Prices*, and ensure, that you see not only the price but also the message in the appropriate color.



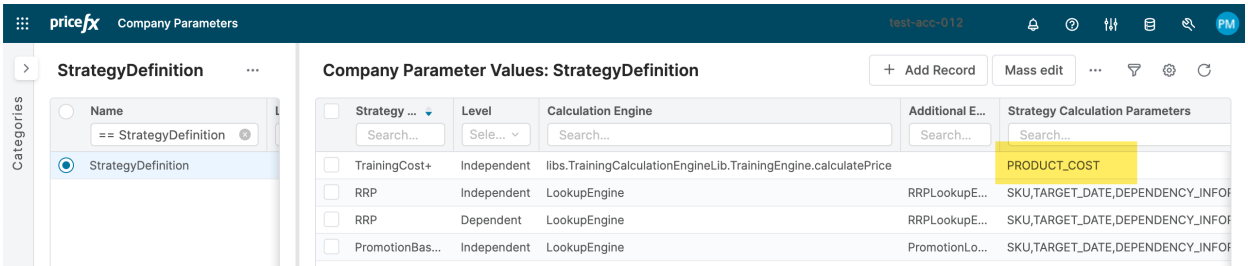
Add Built-in Parameter to Strategy Call

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

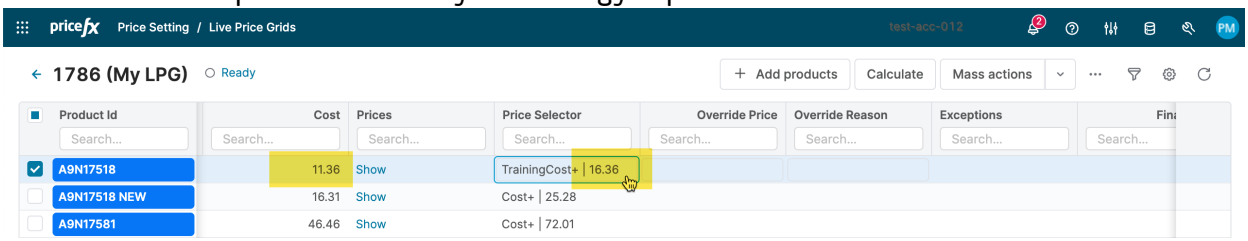
```
Map calculatePrice( BigDecimal cost ) {

    BigDecimal price = cost + 5.0

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

   - Note, that this example has no validation, so it won't check if we have a cost or not. In more complex examples you can build up logic to check if you have the cost, and if not, you can return a warning message for the pricing manager to investigate.
2. Set the Strategy Calculation Parameters:
   a. Navigate to company parameter *StrategyDefinition*
   b. Review the content of *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 full list, you can review code of element *PriceManagerUtils* of calculation logic *PriceBuilderCommonElementUtils*, which is part of the installed Price Setting accelerator. You can find them also in the Accelerator documentation.
   c. For your strategy *TrainingCost+*, set the value for field *Strategy Calculation Parameters* to "PRODUCT_COST"



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 ensure, that the price is calculated as expected based on your strategy implementation.



Add Custom Parameter to Strategy Call

You have learned how to set up the price setting calculation process to pass the cost to our calculation 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 set up, that we want to pass one more parameter to our calculation method call:
   a. Navigate to company parameter *StrategyDefinition*
   b. In definition of your strategy *TrainingCost+*, set the value for field *Strategy Calculation Parameters* to "PRODUCT_COST,OWN_PARAMETER"
2. Modify the implementation of your strategy to accept one more parameter - we call it *ownParamete*

```
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 *IndependentPriceListLogic*, which is part of Price Setting accelerator installation, which you have on your partition.
   b. Review code of element *AdditionalCalculationParameters*. It defines two maps, used for additional custom parameters - *additionalParameters* and *additionalOptionalParameters*.
   c. Modify the definition of *additionalParameters*, to provide a constant value for *OWN_PARAMETER*

```
Map additionalParameters = [
        OWN_PARAMETER : 10.0
]
```

   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 shows ok.
   b. review fields *Cost* and *Final price*, if the calculation is done based on expectations.

Conclusion

You now know how to create your own custom strategy, how to pass parameters between Pricefx Studio and the Pricefx Partition that you're working on, and how to create additional parameters that you can define through StrategyDefinition.
Additionally, you've learnt that if you need more control over messages, both info messages and error messages, you can return this more complex map structure with info, warning and critical types of messages.
You've 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.

To modify implementation of out-of-the-box Calculation Engines, our recommendation is to copy the complete calculation engine to your own library, and modify it there.

Custom Fields in Price List/Grid

Table of Contents

Purpose

This walk-through shows how you can use the logic inheritance functionality to create new pricing logics without editing the existing logic used within the accelerator.

Concept Used for Custom Fields

In Pricefx there is 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 2022-09, Pricefx Studio support of logic inheritance is on the way, so in this article, we refer to Pricefx user interface for building logics.

Add a Custom Field to a Price List/Grid

Create Inherited Logic with Custom Field

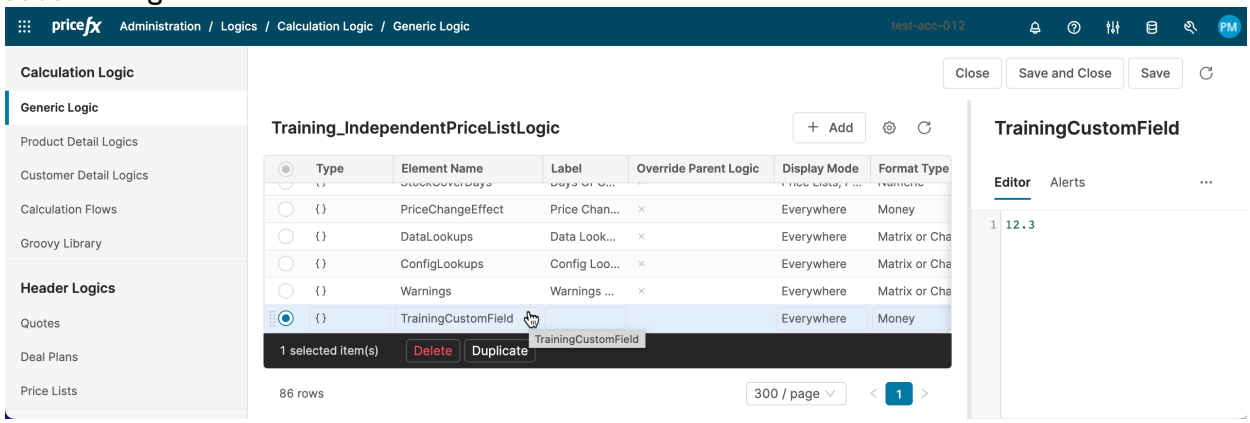
1. Navigate to **Administration › Logics › Generic Logics**
2. **Add** new generic logic with
   - Name: "Training_IndependentPriceListLogic"
   - Parent Logic: *IndependentPriceListLogic*
     - Note, that this dropdown field contains all the existing calculation logics available within the system and 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 practicing, simply use January 1st of the actual year.

- Status: 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 code of some inherited element, you see a message, that this element is inherited, i.e. it's code is coming from the original parent logic, and we can eventually do an override of that code within this new logic.
4. Add new element
   - Note, that it will appear at the end of the list of elements
   - Type: Groovy (Function)
   - Name: "TrainingCustomField"
   - Display Mode: *Everywhere*
     - Caution - remember to uncheck the option *Never*
   - Format Type: *Money*
   - with code:

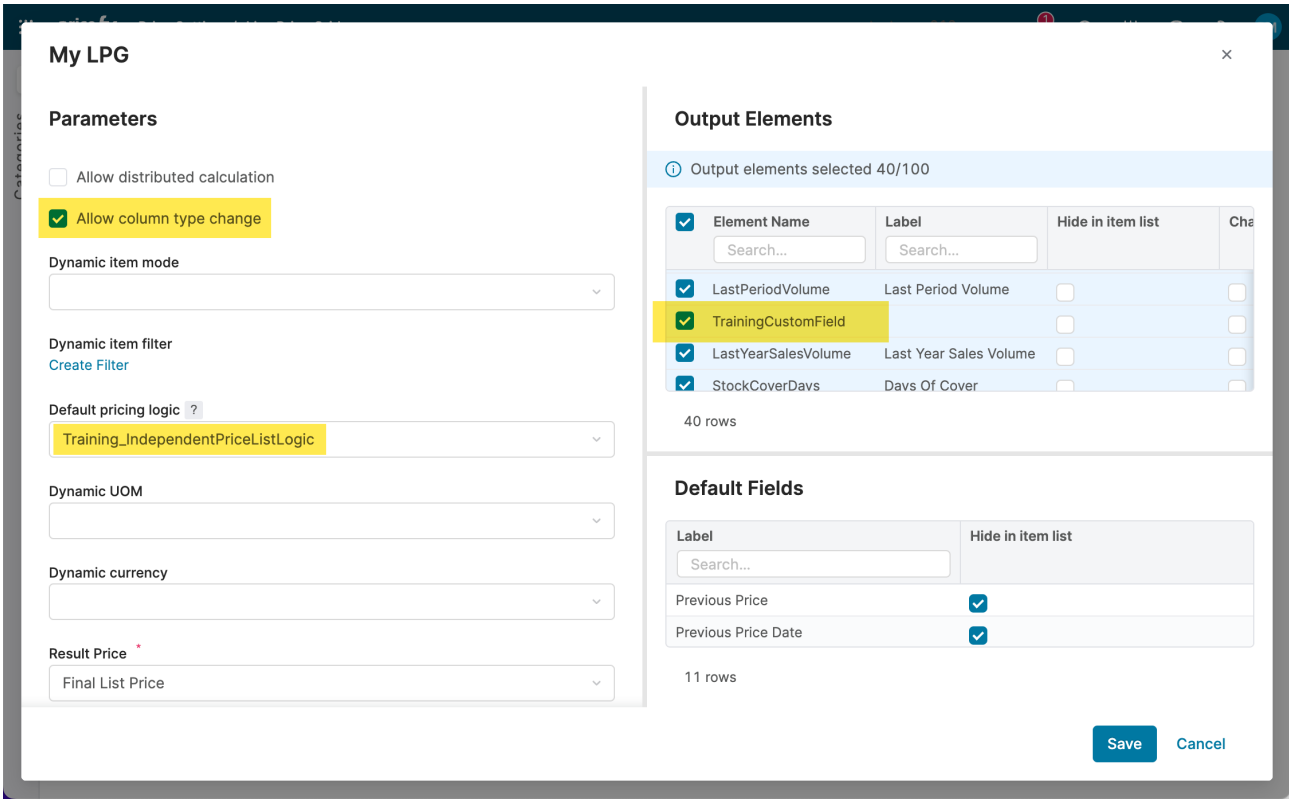
```
12.3
```

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



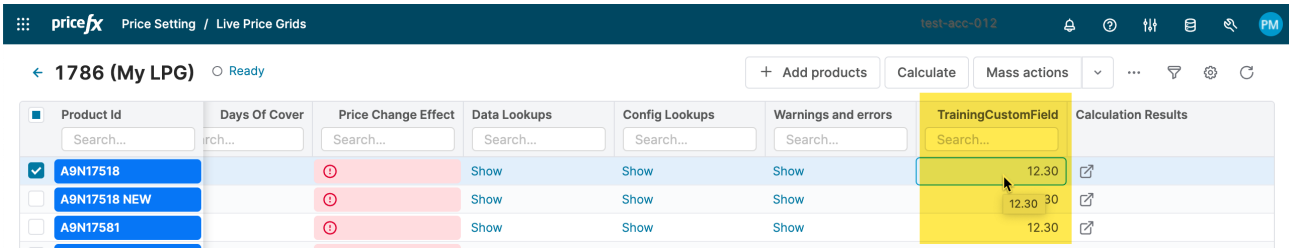
Use the Inherited Logic in Live Price Grid

1. Locate your Live Price Grid from previous exercises. 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: YES.
         - Note, that this is needed, because we added new field and so need to regenerate the meta-data about columns during full recalculation of the price grid.
      ii. Default Pricing Logic: *Training_IndependentPriceListLogic*
      iii. Independent Level Name: *Global*
         - Note, that 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: ensure, that the new element named *TrainingCustomField* is checked too
   c. **Save** the changes in price grid configuration



Test the new Custom Field in the Price Grid

1. Do a full recalculation of the price grid
   a. In the list of price grids, select your price grid and use **Calculate**.
   b. Wait for recalculation.
2. Open your price grid and review the columns - you should be able to find column named *TrainingCustomField* with value `12.3`.



Modify/Override Behaviour of Existing Field

When you open your inherited logic *Training_IndependentPriceListLogic*, there's an element named *Cost*, which has the code/behavior inherited from logic *IndependentPriceListLogic*.
In this part, you will override the out-of-the-box behavior of the accelerator, and instead calculate the *Cost* in a different way. But the out-of-the-box accelerator logic will stay unchanged, because the change will be only in your inherited logic.

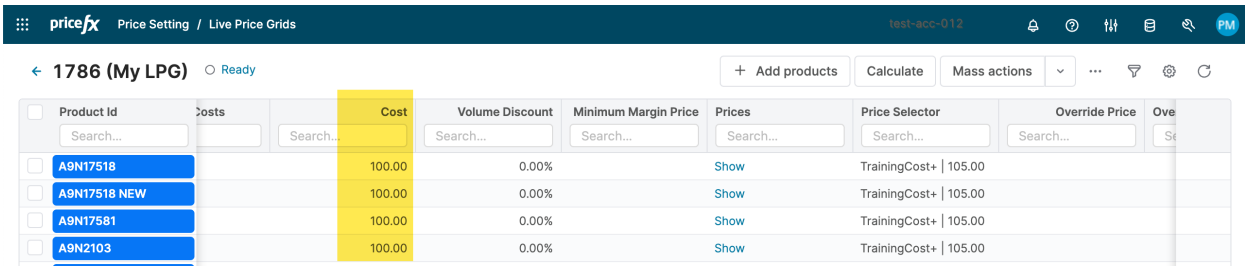
1. Find and **Edit** your generic logic *Training_IndependentPriceListLogic*.
   a. Select the element *Cost*.
   b. In the *Editor*, use action **Override**.
      - Note, that this will copy the element content from the parent logic to your logic. So you can modify it and perform any operation and lookup you need to gather the *Cost* in the way you need.
   c. Comment out the original code.

d. To 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*.



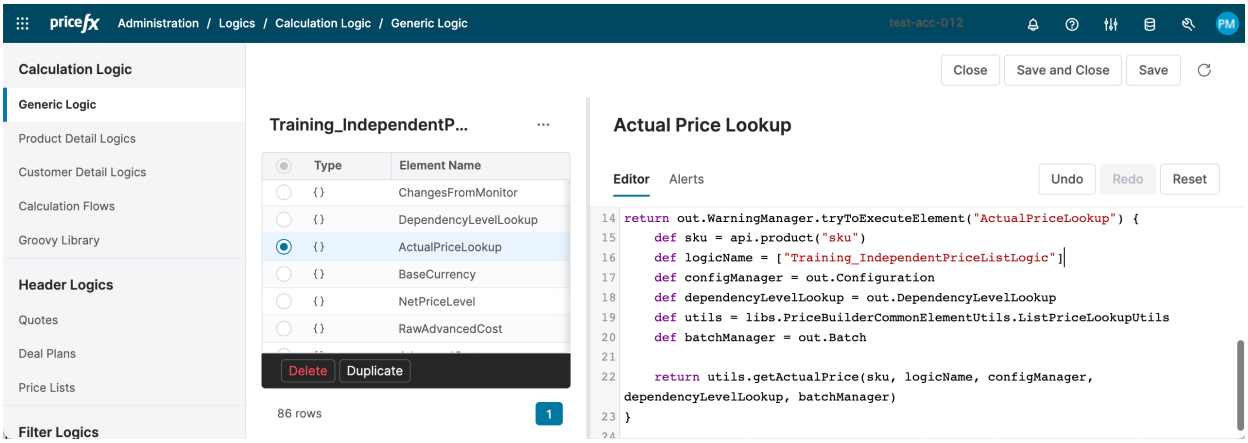
Key Considerations when making custom logics using inheritance

When creating your own pricing logic based on parent logic from Price Setting accelerator, it is important to pay attention to the element *ActualPriceLookup*, which retrieves the actual price of the product. There are different modes how to get the actual price:

- Live Price Grid mode - commonly used mode in live price grids. It takes last approved price of the live price grid, and if there is no approved one (e.g.because you're 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 new season or the 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 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 the generic logic *Training_IndependentPriceListLogic*
   b. Select the element *ActualPriceLookup*
   c. Use action **Override**
   d. Find the line of code, which defines the logic name to be *IndependentPriceListLogic* and change the logic name to your "Training_IndependentPriceListLogic".



e. **Save** the logic

2. Test the change - to be able to test, you would need to simulate the scenario above, which is actually using this logic name. So in our use case (live price grid) it will not have any effect.

Conclusion

You learned how customize the price list/grid calculations, when using Price Setting accelerator - particularly how to add new or modify existing fields available in the price list/grid line items.
This was done using a feature called *Logic Inheritance*, which allows you to safely modify the Price Setting accelerator calculation logic, by making your own logic, which inherits all behavior from the out-of-the-box logic of Price Setting accelerator.

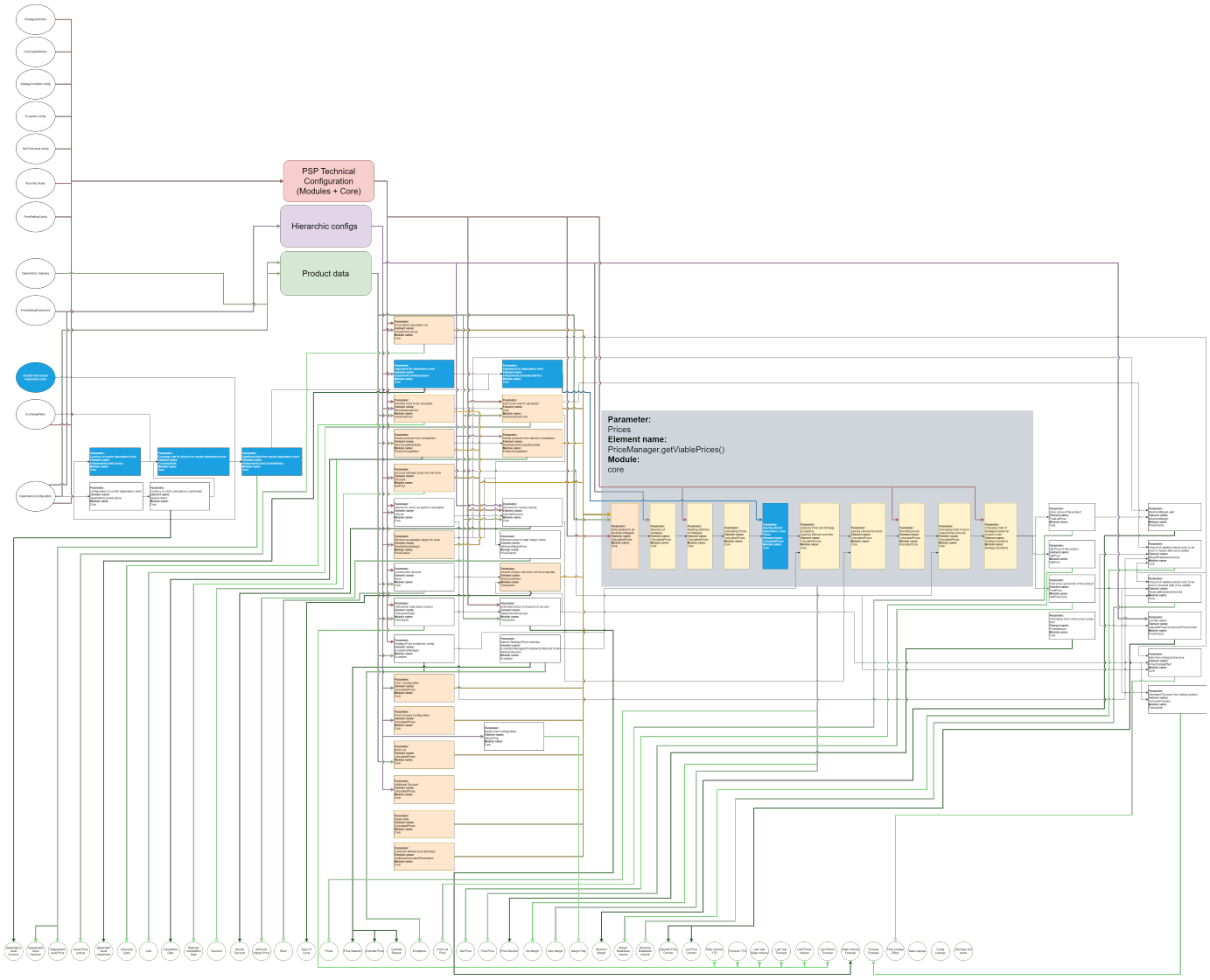
Configuration (Price Setting)

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.

Strategy Importance

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 independent 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 Independent and Dependent Levels Are Calculated

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

Independent Price List Calculations

Independent 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 "Independent Level Adjusted Price", which is a Final Price from the independent price list for a given product adjusted by the dependency adjustment. It can be overridden by setting "No" for the "Prioritize Independent Level Price" column in the Strategy Selection Lookup PP. In this case, "Independent 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 independent level adjusted prices, if the same strategy was calculated for a given product in the independent price list.
- Prices from the independent price list can be ignored on the dependent level by setting "Independent Level Only" to "Yes" in the StrategyDefinition PP.
- Dependent prices come in pairs with independent level adjusted prices (with dependent before independent), unless "Independent Level Priority" is set in the StrategyDefinition PP . Independent level priority is taken into consideration only when the strategy is defined for both dependent and independent level. It has to be set by an entry on the independent 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 independent level, it will take the independent 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 independent levels and you have the same cost, it will result in two different prices. One freshly calculated and one taken from the independent level and with an applied markup factor - which might seem unexpected.

Summary

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

Flag	Where you find the flag	What it does
Prioritize Independent Level Price	PP Strategy Selection	Default is "Yes". When you change to "No", this will force the system to put the "Independent Level Adjusted Price" (= final approved price from independent level with the markup) to the end of the priority list.
Level	PP Strategy Definition	You can have "Independent" 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 independent level. When you only want to take some price from the independent level and add a markup factor, you do not have to configure the strategy for the dependent level.
Independent Level Only		You can only set up this flag for "Level" = "Independent". When this is done like this, it will prevent the inheritance of this independent level price to the dependent level. So with this you force a price not to be taken over to the independent level and applied with the markup factor.

Independent Level Priority		When one strategy is calculated both on the independent and dependent levels, it will appear twice in the dependent level: 1. (Re)calculated on the dependent level base. 2. Taken from the independent 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:

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 one 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. Independent price is always taken from the first found master in the hierarchy but there are two exceptions:

- Independent 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.

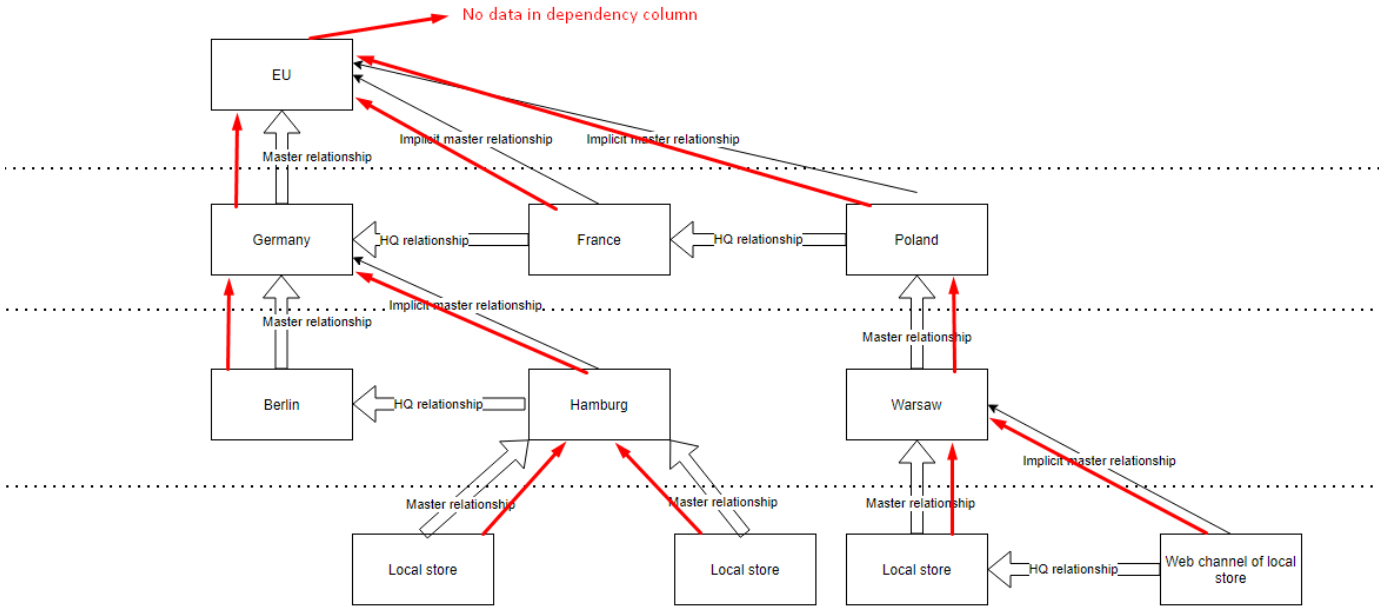
⚠ Currently, it is not possible to create a virtual dependency level on top of the hierarchy, as anything below the top uses the dependent logic – and expects to have a master price somewhere in the tree.

Data Lookup Fallbacks

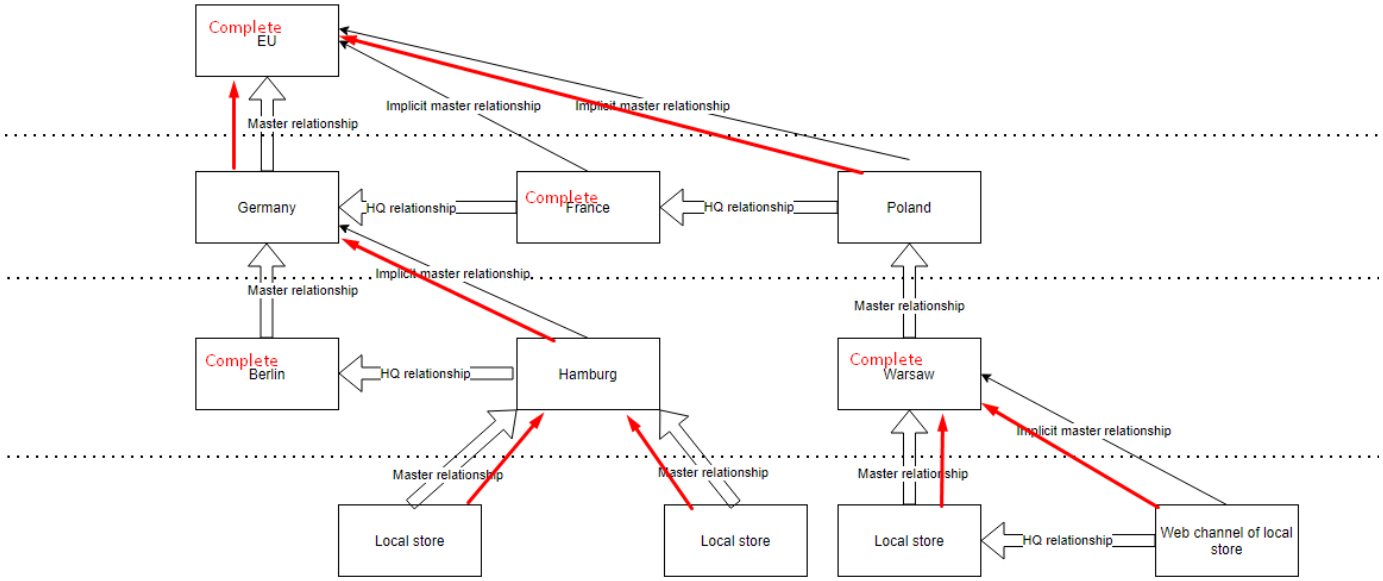
Fallbacks follow the following rules:

- 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:

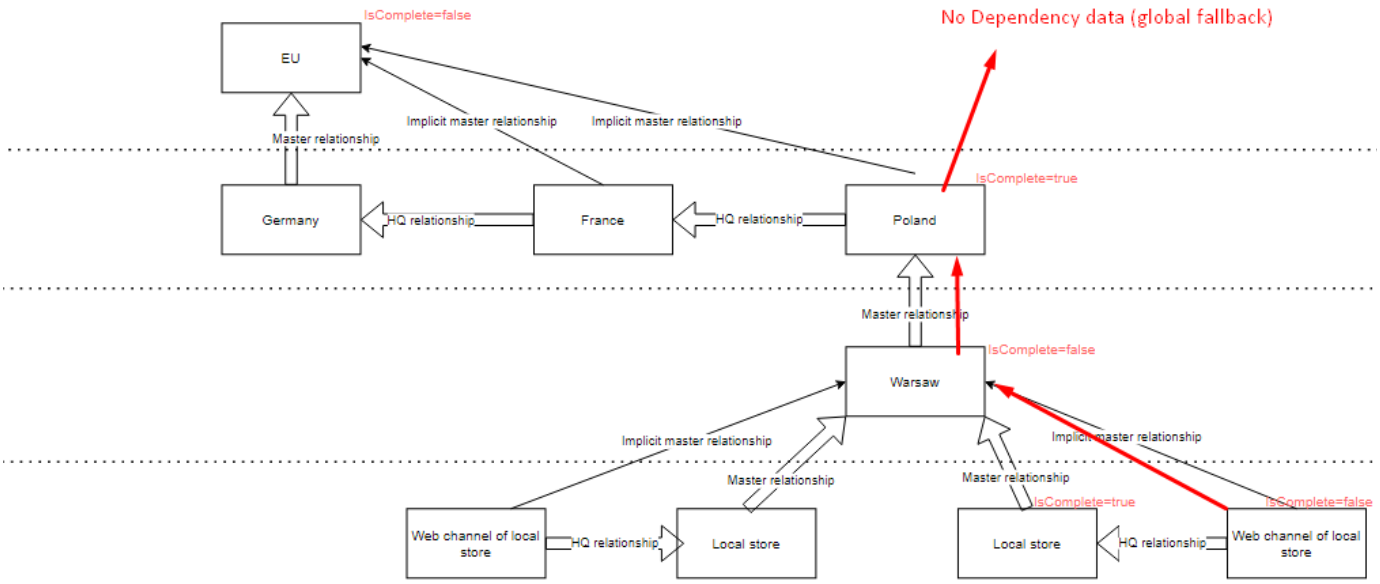


Lookup Keys Config Fallbacks

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.

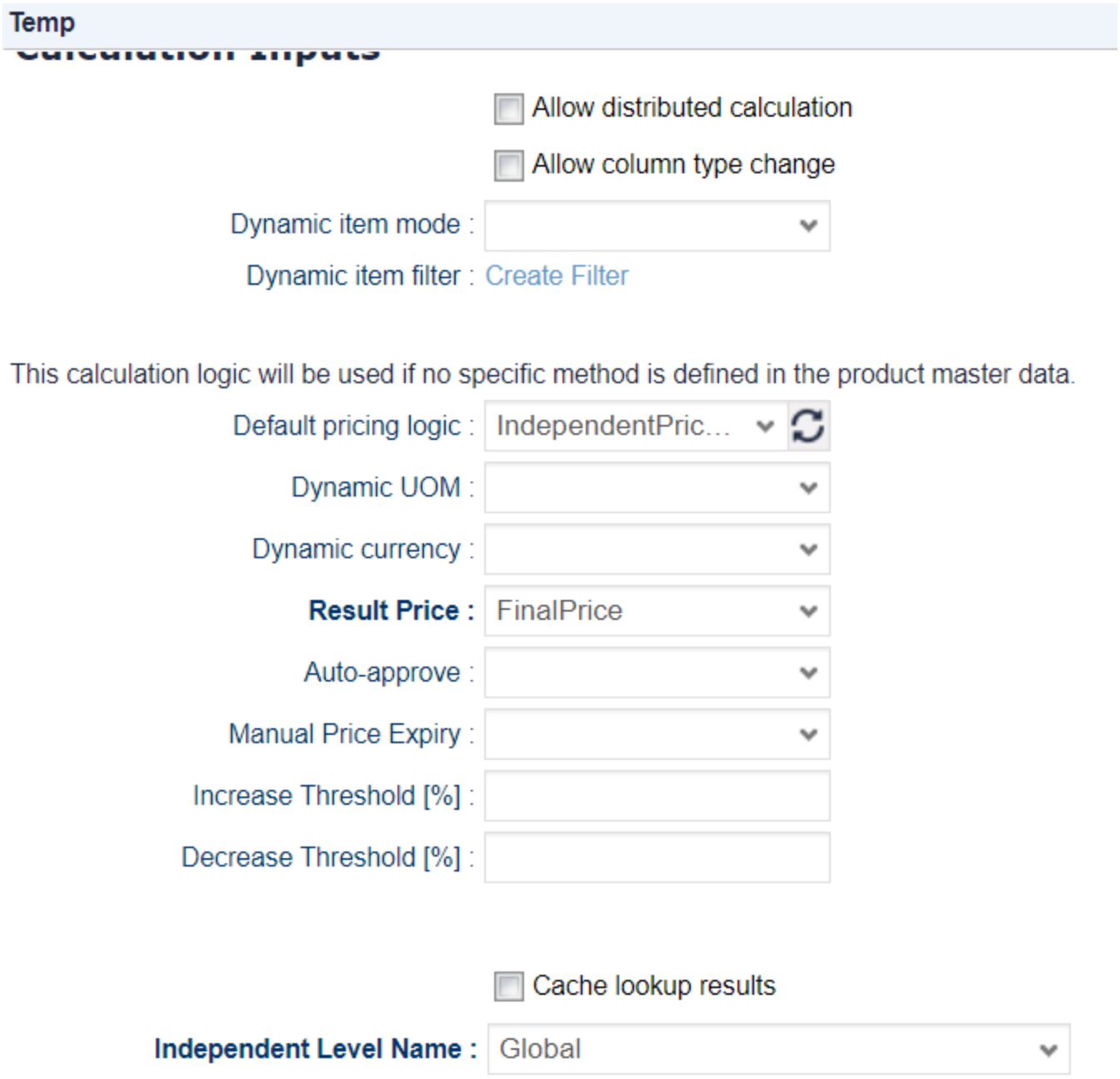
> ⚠ The Dependency Field and Mapping Source Field in DependencyMappingConfig PP are omitted in this type.

Dependency Mapping Mechanism

Here you will find an example with Cost mapping.

Select Dependency

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



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

Company Parameter Values : DependencyConfiguration [1]

	global								
	Dependency Level Na...	Depends ...	Source Type	Source...	Dimens...	Currency	Is com...	ISO C...	Sal...
	Global	Independent	*	*	Area	USD	No	GG	SO00

With Lookup Mapping Type

In DependencyMappingConfig PP, set the following:

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

With Table Mapping Type

In DependencyMappingConfig PP, set the following:

cost			
Name	Dependency Field	Type	Mapping Sourc...
Cost	DependencyLevel...	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.

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

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

With None Mapping Type

In DependencyMappingConfig PP, set the following:

cost			
Name	Dependency Field	Type	Mapping Sourc...
Cost		None	

- **Type** is set to *None* to indicate the mapping type.

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 in the CostData PX table and get a value from the column pointed in the Source Field as cost value (Cost field).

Data Lookups

Price Setting Package uses LookupManager 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
- Hierarchical Lookups

Standardized Lookups

The flow of standardized lookups:

1. Once for a batch (by default 200 products), a DB lookup is performed.
    a. User entered source type is converted to the proper database type (e.g. PP to MLTV4).
    b. The first key of the database type is read; in standardized lookups, it is always SKU.
    c. Validator closures are loaded. These will be used later.
    d. SortBy parameter is prepared. It is there only for technical purposes (finding data with proper valid after date), otherwise, no sorting is performed.
    e. Filters are constructed.
        i. ValidityDates (if used) are in proper ranges. Price Setting Package supports "ValidAfter" mode and "ValidAfter-ValidTo" mode.
        ii. Data load is prepared for any SKU in the batch.
        iii. Data is placed in a proper table. It might be a simple filter, or it might be a complex one when using Dependency Mapping with the "Table" type (the whole hierarchy is looked up).
        iv. Data is prepared for proper dependency levels (the whole hierarchy). It is used only when the Dependency Mapping with "Lookup" type is used.
    f. Data is looked up.
    g. Data is grouped per SKU (if lookup did not fail).
    h. For every SKU the group validators are run (if lookup did not fail).
        i. Validators on the data entry level – each Dependency Level in the hierarchy has its own closure. It marks every entry if it belongs to the given dependency level. It is especially important in the "Table" Dependency Mapping where entries from multiple tables are mixed together.
        ii. Validator on the entry set level – marks the whole set of entries and informs if data overlap. It also considers the fallback mechanism. If there is any data for the first dependency level, it checks only these entries.
    i. Data lookup is registered for debugging (registering keeps data for other products in the batch).
2. It is checked if data lookup threw any exception. Any exceptions are propagated further for every product in the batch so that config issues are easier to debug.
3. Data for SKU is read.
4. A warning is registered if data has overlapped (only in the "ValidFrom-ValidTo" mode).
5. Proper data for the product is returned.

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

Rounding Rules

We do not perform lookup per SKU.

Preference: Rounding Rules Lookup

Competition Data

- No user-config ("ValidFrom", "ValidFrom-ValidTo" modes are not supported).
- Only PCOMP as a source (single table, no table name configuration).
- No Dependency Mapping with the "Table" type supported (single table, cannot create table per dependency level).

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

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 O ther 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:

- Additional Engine Configuration
- Default Strategy Calculation Parameters
- Define Attribute Data (with Sample Data)

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:

Valid	+	AttributeP	+	AttributePX	+	AttributePP	
Invalid	+	AttributeP	+	AttributePX		AttributePP	
Invalid1	+	AttributeP	+	AttributePX	+	AttributePP	+

- 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:	Selects the target competitor to compare to. Case insensitive.

	• `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 per centile of the whole competitor range.	
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 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. It can be done by supplying one of these:

- Hardcoded value calculated earlier (standardParameter).
- Closure with a code which will be executed when the first element is empty (optionalParameter). This way this calculation can be lazy-initialized.

Possible return values of the engine-like function:

- Price as BigDecimal
- Throwed exception (will be handled and showed 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".

Example configuration of custom engines can be found at https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/2098036839/Recipe+Custom+Strategies?NO_SSR=1 (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 independent 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:

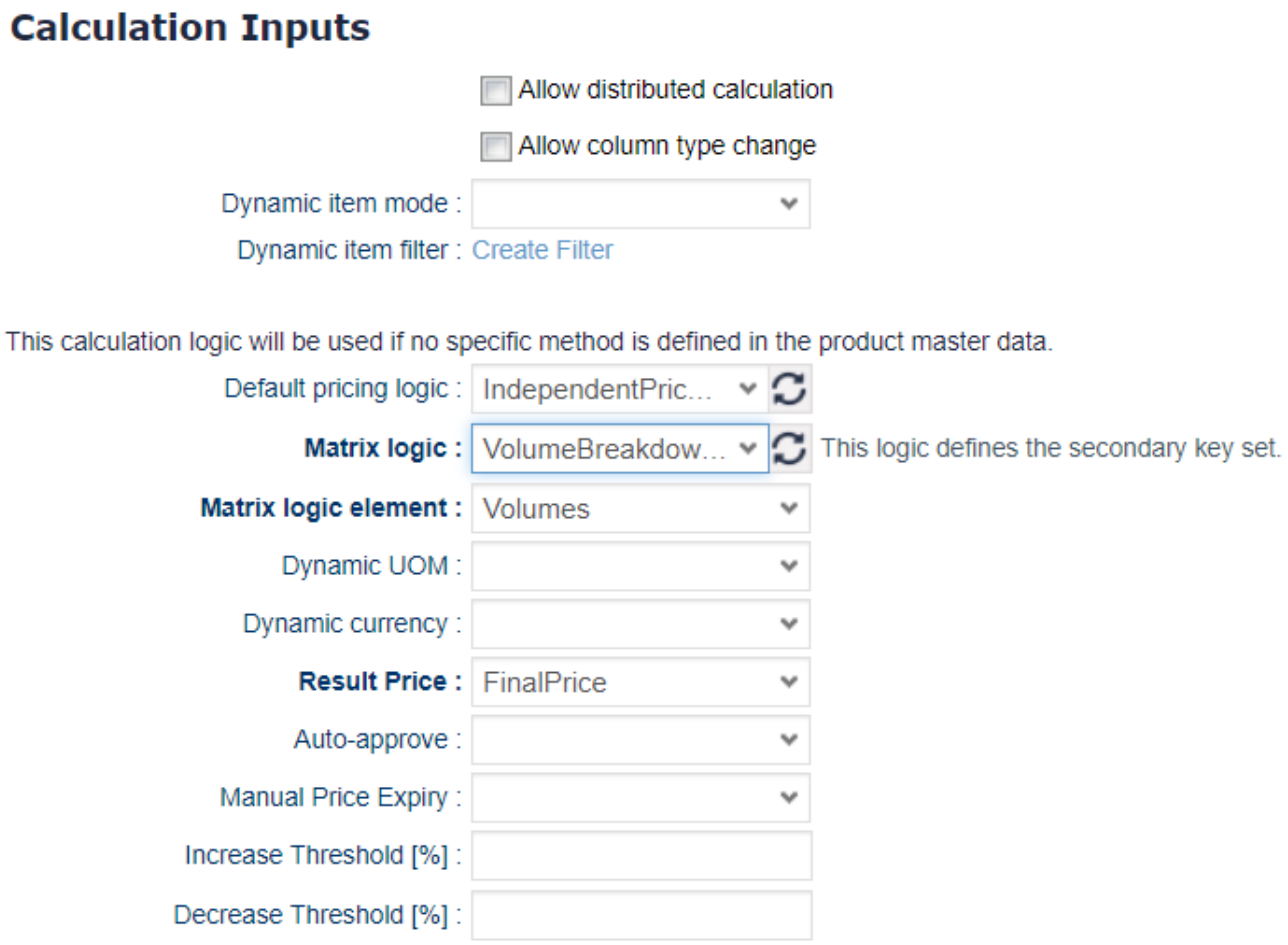
- Break-even volume for revenue %: (New Price minus Previous Price) / Previous Price) * -100%
- Break-even volume for margin %: (((Old Price - Cost) * Volume) / (New Price - Cost)) / Volume) - 1

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:



Product Segmentation

> ⚠ 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 Independent 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 Independent 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 Independent 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 Price Setting Package Configuration Wizard or manually by turning them on in PriceSettingModules PP and configuring 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. Stock data is independent from transactions, but calculation of StockCoverDays is dependent on this module.	PSP_TRANSACTION_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 independent 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 Stock 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.
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 Config (PriceSettingConfig).
2. Set the dependency mapping in the DependencyMapping PP table with the key `Actual Price`. Details can be found at DependencyMappingConfig PP.

PSP Stock 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 Stock

Used to calculate Stock Cover Days.

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 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	Independent 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	Select a strategy from the `Price`	Set the strategy in a configuration table	Can do both Current line and Exception tables methods	Override is not allowed

	prices as the selected strategy	`Selector` dropdown list			

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 name of the price strategy chosen in the exception.	Choose a strategy from the PriceSelector drop-down
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 Independent 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 independent pricing in the PriceSettingConfig PP table with the keys `Independent Manual Override Allowance|Price`.
2. Set the product strategy override level for independent pricing in the PriceSettingConfig PP table with the keys `Independent 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.

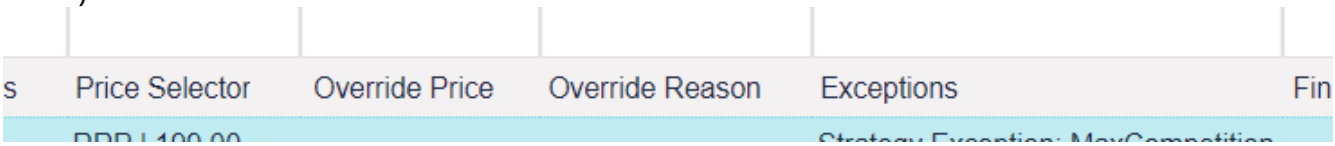
2. Set the dependency mapping in the DependencyMapping PP table whose key is `Strategy Exception`. Details can be found at DependencyMappingConfig PP.

PSP Rounding Rules Module

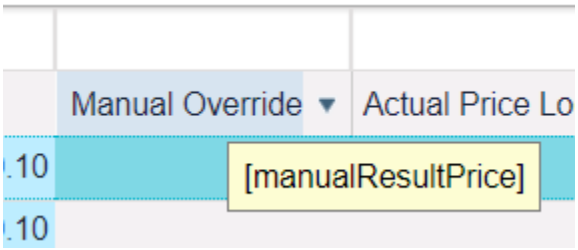
The Rounding module allows users to round prices to business-friendly values. Rounding is done by using a defined set of rules. This set can be expanded if needed.

⚠ **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 Independent 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 mode is Gross/Net and the module is turned on	BigDecimal	List price with a discount applied.
Discount	Discount	Yes	Yes		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 Price Setting Level.

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-AdditionalDiscount) / (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-AdditionalDiscount)`	Null	Null
Off	N/A	Some value	`target price / (1-AdditionalDiscount)`	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. Stock data is independent from transactions, but the calculation of StockCoverDays depends on this module.

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			Description

			Independent PL/PG	Dependent PL/PG	
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.

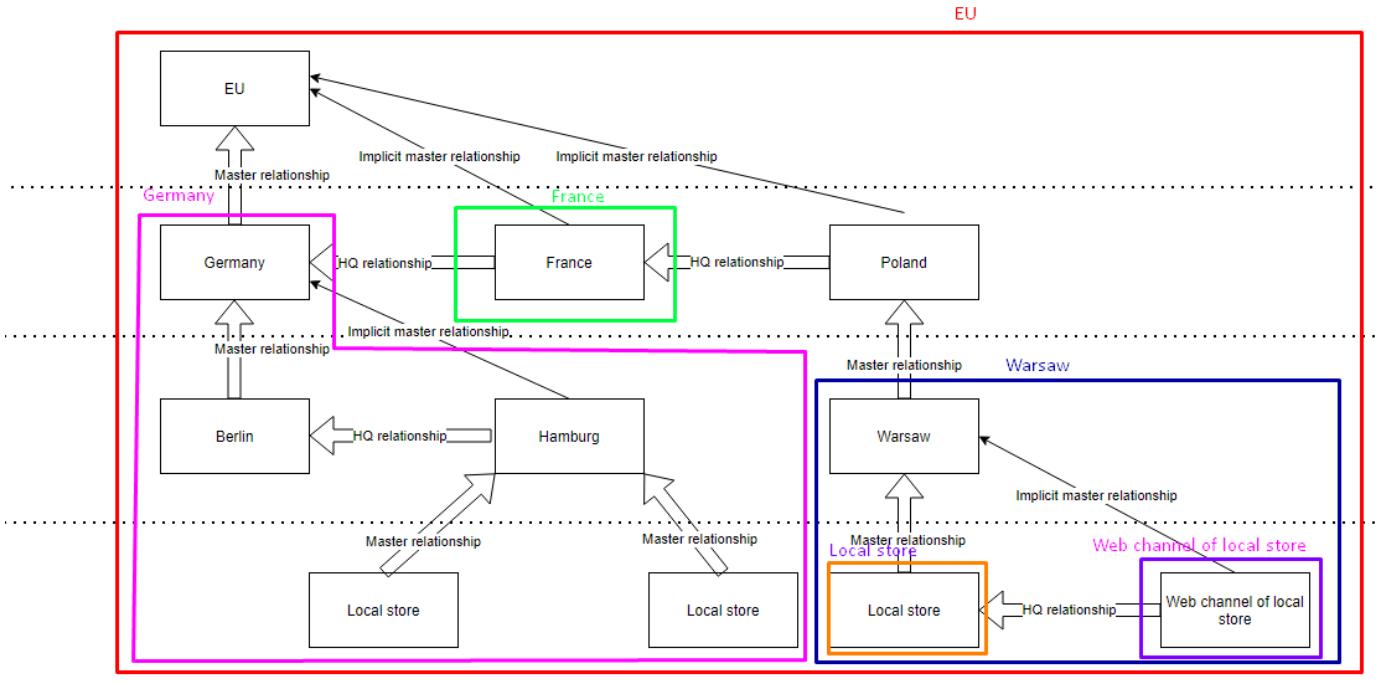
When the transaction data source points to Product Extensions, the currency is converted using the ExchangeRate PP. For details, see ExchangeRates PP.

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. Users are expected to fill 3 more columns when they use Price Insights Dashboard. Those are Min, Max, and Avg turnovers of all data aggregated into that entry.

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 independent price and the dependent price in a dependent PL/PG.

Module Elements

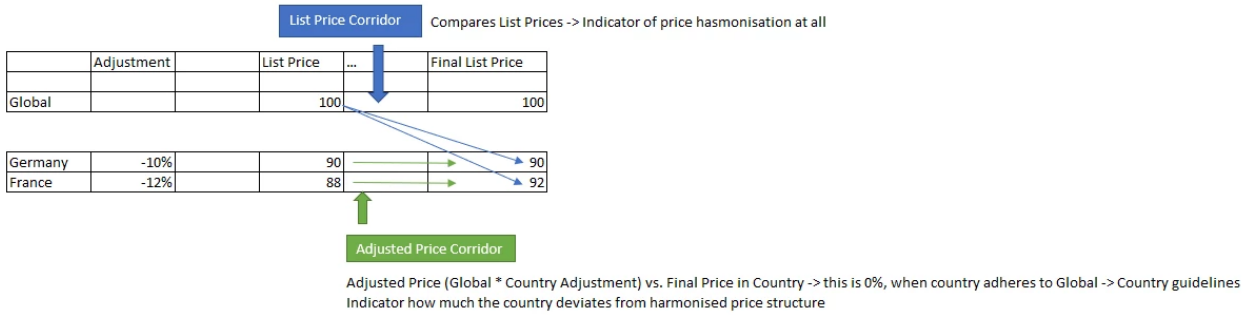
Technical name	Label	Available in Independent PG/PL	Available in Dependent PG/PL	Description
MinimumMargin	Minimum Margin	Yes	Yes	The minimum margin specified for the product.
ListPriceCorridor	List Price Corridor	No	Yes	Delta between the independent price (*) and final list price. On independent price, refer to Element Data Sources > List Price Corridor section below.
AdjustedPriceCorridor	Adjusted Price Corridor	No	Yes	Delta between the adjusted independent price and final list price. On independent 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` (%).

Element Data

Values for price checks are configured in respective price parameter tables or calculated results.

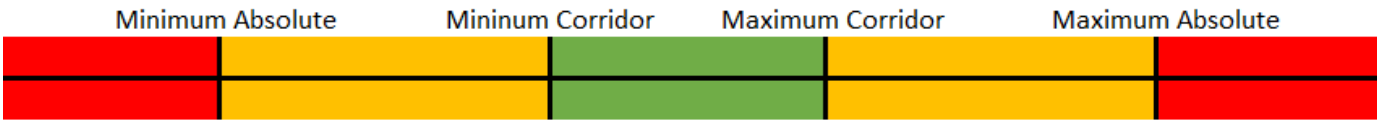
- **Minimum Margin – MinMargin PP table**
  From table with naming convention: `<<dependency>>MinMargin`.
  Examples: MinMargin, AsiaMinMargin, GlobalMinMargin,...
- **List Price Corridor (Dependent PG/PL only)**
  The value is taken from the final list price of the referred independent PG/PL.
  To set the referred independent 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 independent 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 independent 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 independent 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 independent and dependent prices in a dependent price list. Independent 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.

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+ (Independent 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	Move the left-hand side strategy to the last position of the strategy list.	(fallback)
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.

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 Independent 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 Names
- Adjustments after Changing the PP PriceSettingLevel
- Change Product Segmentation

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 "Independent" 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

Rename Dependency Level Names

⚠ 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 Independent 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

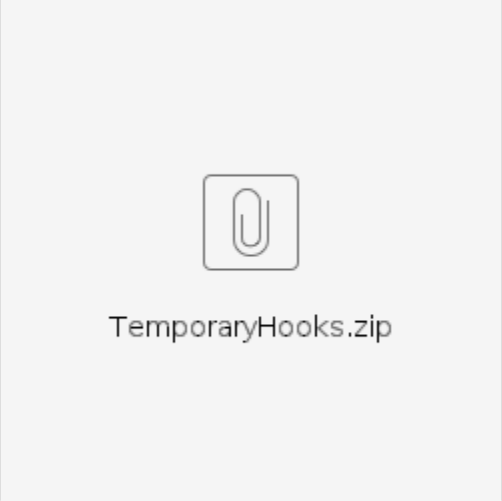
1. These configurations need to be updated:
   - PriceSettingDimensions PP
   - DependencyConfiguration PP
2. Choose one of these two options:
   - Run "Price Setting Package - Upgrade" in PlatformManager Marketplace if you have the newest version of Accelerator. If you do not have the newest version, 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.



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.

- Standard Configuration
- Configuration Company Parameters (Price Setting Package)
  - Bootstrapped Tables
    - Additional Discount Lookup
    - Adjusted Price Corridor Lookup
    - Base Strategy Selection Lookup
    - Cost Plus Lookup
    - Cost Selection Lookup

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
  - Configuration for dependency mapping in some of the included data containers and product extensions
  - Sample of complete configuration that has to be completed
- Sample for Pricing Strategy Definition
- Sample for Rounding Rules Configuration
- Sample for Pricing Exceptions

Configuration Company Parameters (Price Setting Package)

- 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

- 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

- 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

- Price Strategy #1
- Price Strategy #2
- Price Strategy #3
- Price Strategy #4
- Price Strategy #5

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

Name

{nameOfDependency}CostPlus

Attributes

- Plus %
- Plus Absolute

Description

It is one of the supportive parameters coming with pre-configured strategies. Depending on additional configuration of Cost+ strategy, Plus % OR Plus Absolute will be used.

Cost Selection Lookup

Name

- {nameOfDependency}CostSelection

Attributes

- 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

Name

- {nameOfDependency}DependencyLevelAdjustment

Attributes

- Adjustment %

Description

Valid only for Dependent PL/PG. Value of Price Adjustment between master Dependency Level Price and current Dependency Level.

Discount Lookup

Name

{nameOfDependency}Discount

Attributes

- 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

- 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

- Critical Threshold
- High Threshold
- Medium Threshold

Description

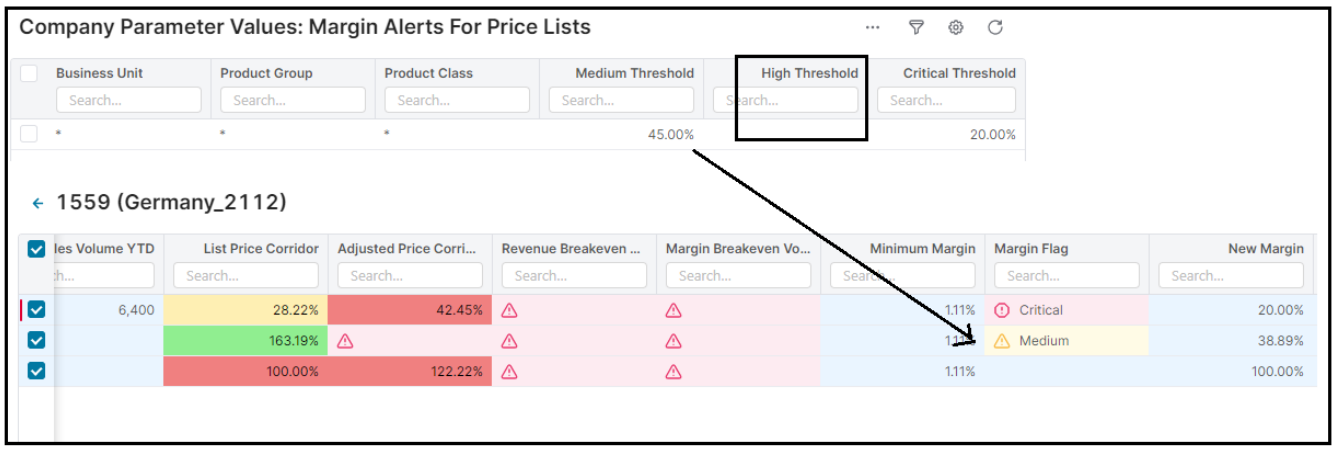
It is the value of Margin Alerts thresholds in percent for the Price Checks module. The `Margin Flag` element in the PL/PG has the values Critical, High, or Medium, depending on what margin falls into what range. For Critical and High ranges, it will color the row in red. For the Medium range, it will 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

- Min Margin %

Description

Value of the minimum margin in percent for the Price Checks module.

Price Increase Lookup

Name

{nameOfDependency}PriceIncrease

Attributes

- Plus %
- Plus Absolute

Description

It is one of the supportive parameters coming with pre-configured strategies. Depending on additional configuration of PriceIncrease strategy, Plus % OR Plus Absolute will be used.

Relevant Competition Data Lookup

Name

- {nameOfDependency}RelevantCompetitionData

Attributes

- 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

Name

{nameOfDependency}StrategySelection

Attributes

- Price Strategy #1
- Price Strategy #2
- Price Strategy #3
- Price Strategy #4
- Price Strategy #5
- Prioritize Independent Level Price

Description

Order of used strategies. Other strategies might appear before, if exceptions/overrides/base strategies are used. “Prioritize Independent Level Price” is relevant only for Dependent PL/PG – it defines if a price from master Dependency Level should be placed at the first place or at the end.

Volume Breakdown Lookup

Name

- {nameOfDependency}VolumeBreakdown

Attributes

- 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.

- Cost Lookup
- Actual Price Lookup

- Stock Lookup
- Exceptions and Manual Override Allowance Config (PriceSettingConfig)
- Exception Lookup
- Rounding Rules Lookup
- Transaction Lookup
- Forecast Config (PriceSettingConfig)
- Forecast Lookup
- Last Period Config (PriceSettingConfig)

Cost Lookup

This configuration represents a single row of the PriceSettingConfig Price Parameter.

This configuration row can be set up separately for independent and dependent price lists. It means that there can be maximum 3 rows with the key "Cost", each with a different value in the "Condition" column.

Column	Value	Description
Key	Cost	Holds the product cost configuration.
Condition	*	Condition is not needed here. Leave it at the default value.
Type	Lookup	
Source	PX	Currently, only a PX lookup is supported.
Source Table	{PX name}	Remember to use "name" from the configuration, not a label.
Source Field	{name of the column with the product cost}	
Source Field 2	Currency	If there is a currency in the PX Cost table, get it and do the conversion. If there is none, then get the LPG/PG BaseCurrency.
Source Field 3	{name of the column with the start date of the validity period, typically ValidFrom}	Used for searching with a validity period. • When the ValidFrom field (Source Field 2) is left empty, the ValidTo field (Source Field 3) will be ignored and the normal search approach will be applied. • When the ValidFrom field is set but the ValidTo field is left empty, it will find the value with the closest validity date. • When both ValidFrom and ValidTo fields are set, it will find the value falls in the time range between ValidFrom source value and ValidTo source value.

Source Field 4	{name of the column with the end date of the validity period, typically ValidTo}	

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	• PX • PL • PG	Source type for the lookup. PG might be used only when using Price Grids. It will not work for Price List.
Source Table	{name of the PX}	Needed only when we use PX as a source. For PL it is always the last approved price for a given dependency level name.
Source Field	{name of the column with price information in the source data}	Needed only when we use PX as a source.
Source Field 2	{name of the column with the start date of the validity period}	Needed only when we use PX as a source.
Source Field 3	{name of the column with the end date of the validity period}	Needed only when we use PX as a source.
Source Field 4	{name of the column with the currency}	Needed only when we use PX as a source.

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.

Stock Lookup

This configuration represents a single row of the PriceSettingConfig Price Parameter.

This configuration row can be set up separately for independent and dependent price lists. It means that there can be maximum 3 rows with the key "Stock", each with a different value in the "Condition" column.

Column	Value	Description
Key	Stock	Holds the product stock configuration.
Condition	*	Condition is not needed here. Leave it at the default value.
Type	Lookup	
Source	• PX • PP	Type of the source table.
Source Table	{name of PX or PP}	Remember to use "name" from the configuration, not a label. ⚠️ For PP lookups, we assume two things: 1. When dependency mapping is not applied, there is only one key which is the SKU code 2. For other cases: a. key1 - SKU code b. key2 - Dependency mapping value c. (optional) key3 - Valid from d. (optional) key4 - Valid to
Source Field	{name of the column with the stock data for a product}	
Source Field 2	{name of the column with the start date of the validity period, typically ValidFrom}	Used for searching with a validity period. • When the ValidFrom field (Source Field 2) is left empty, the ValidTo field (Source Field 3) will be ignored and the normal search approach will be applied. • When the ValidFrom field is set but the ValidTo field is left empty, it will find the value with the closest validity date. • When both ValidFrom and ValidTo fields are set, it will find the value which falls in the time range between the ValidFrom source value and ValidTo source value.
Source Field 3	{name of the column with the end date of the validity period, typically ValidTo}	

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	• Independent Manual Override Allowance • Dependent 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.

Exception Lookup

This configuration describes two rows of the PriceSettingConfig Price Parameter.

⚠ Exception Lookup will not be performed, if allowance config forbids it: Exceptions and Manual Override Allowance Config (PriceSettingConfig).

One configuration row should exist for every key-condition pair mentioned in the table below.

Column	Value	Description
Key	• Strategy Exception • Price Exception	Holds the exception configuration. There is one row for each type.
Condit	*	Condition is not needed here. Leave it at the default value.

ion		
Type	Lookup	
Source	• PX • PP	Type of the source table.
SourceTable	{name of PX or PP}	Remember to use "name" from the configuration, not a label. ⚠️ For PP lookups, we assume two things: 1. The only existing key is the SKU code, dependency mapping is not applied. 2. If there are two keys: a. key1 – SKU code b. key2 – Dependency mapping value
SourceField	{name of the column with the exception data for a product}	The type of this column depends on the exception type. • Strategy – String with the name of an existing strategy definition. **Important note**: Strategy exception can be selected only from strategies configured in the StrategySelection PP for a given product. It means that if we try to use a strategy that is not calculated by default for the product, this exception will be ignored. • Price – Numeric value with a price override.
SourceField2	{name of the column with the start date of the validity period, typically ValidFrom}	Used for searching with a validity period. • When the ValidFrom field (Source Field 2) is left empty, the ValidTo field (Source Field 3) will be ignored and the normal search approach will be applied. • When the ValidFrom field is set but the ValidTo field is left empty, it will find the value with the closest validity date. • When both ValidFrom and ValidTo fields are set, it will find the value which falls in the time range between the ValidFrom source value and ValidTo source value.
Sourc	{name of the column with the end date of the validity period, typically ValidTo}	

eField3		
SourceField4	{name of the column with currency}	Configuration for optional currency conversion. No data means no conversion.

Rounding Rules Lookup

This configuration represents a single row of the PriceSettingConfig Price Parameter.

Column	Value	Description
Key	Rounding Rules	Defines how to look up values for the Rounding Rules lookup.
Condition	*	Condition is not needed here. Leave it at the default value.
Type	Lookup	Hardcoded value.
Source	PP	Hardcoded value.
Source Table	{name of the PP}	Name of the PP table where rules are stored
Source Field	{name of the column with Rounding Rule}	Remember to use "name" from the configuration, not a label.
Source Field 2	{name of the column with Rounding Mode}	Remember to use "name" from the configuration, not a label.
Source Field 3	{name of the column with the start date of the validity period, typically ValidFrom}	Used for searching with a validity period. • When the ValidFrom field is left empty, the ValidTo field will be ignored and the normal search approach will be applied. • When the ValidFrom field is set but the ValidTo field is left empty, it will find the value with the closest validity date.
Source Field 4	{name of the column with the end date of the validity period, typically ValidTo}	

		• When both ValidFrom and ValidTo fields are set, it will find the value which falls in the time range between ValidFrom and ValidTo source values.

Why This Lookup Is Not Standardized

We do not perform lookup per SKU.

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.

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

Module Configuration Tables

- PriceSettingModules PP
- For PSP Advanced Cost Module
- For PSP Strategy Conditions Module
- For PSP Override Module

PriceSettingModules PP

Column Name	Name	Status
Values	• 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	• On • Off
Description	Technical name of the module. ⚠ All values needs to be present in the PP for the package to work correctly.	Defines whether any given module is enabled or disabled.

Example:

Price Parameter Values : PriceSettingModules [9]

Name	Status
PSP_PRODUCT_COMPETITION_MODULE	On
PSP_TRANSACTION_MODULE	On
PSP_NET_PRICE_MODULE	On
PSP_OVERRIDES_MODULE	On
PSP_PRICE_CHECKS_MODULE	On
PSP_PRICE_FLEXIBILITY_MODULE	On
PSP_STRATEGY_CONDITION_MODULE	On
PSP_ROUNDING_RULES_MODULE	Off
PSP_ADVANCED_COST	Off

For PSP Advanced Cost Module

- Advanced Cost Lookup (CostTypeDefinition PP)

Advanced Cost Lookup (CostTypeDefinition PP)

Column name	Cost Type	Calculation Engine Suffix	Type	Calculation Method	Source Table	Source Field	Valid From	Valid To	Dependency Field	Dependency Type	Mapping Source Field	Currency
Values	Any string	Any string	"Look up"	• SINGLE • AVG • SUM	Name of the PX	Name of the PX column with cost	Name of the PX column with a valid From date (optional)	Name of the PX column with a valid To date (optional)	Name of the column in Dependency Configuration (country mapping feature)	• Table • Lookup (country mapping feature)	Name of the PX column (country mapping feature)	Name of the PX column with currency code
Description	Name of the definition, used in cost selection.	String concatenated with "COST" and passed to engines as a parameter.	Other types will be added in the future.	For details see Advanced Cost.	Note: Name is required here, not a label.							

For PSP Strategy Conditions Module

- StrategyConditions PP

StrategyConditions PP

For additional details see also PSP Strategy Conditions Module.

Column name	Order	Condition	Rule	CheckException
Values	{incrementing integer}	{condition}	{rule}	• Yes • No
Description	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.	Expression to be evaluated (boolean value to activate the rule). For details see PSP Strategy Conditions Module.	Defines what to do with the strategy when the condition is evaluated to true. For details see PSP Strategy Conditions Module.	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.

For PSP Override Module

- PricingExceptions PP

PricingExceptions PP

Column name	SKU	Dependent Level Name	Price Exception	Strategy Exception
Values	{SKU from Product master}	{name of dependency level} as stated in DependencyConfiguration PP	Defined price exception for given parameters	Defined strategy (as stated in StrategyDefinition PP) exception for given parameters
Description	e.g. "MB-0001"	e.g. "Global"	e.g. "20.0"	e.g. "Cost+"

Example:

Other Configs

The configuration options described here may be required to set up to enable individual Price Setting modules and features.

- DependencyMappingConfig PP

- DependencyConfiguration PP
- ExchangeRates PP
- PriceSettingDimensions PP
- PriceSettingLevel PP
- StrategyDefinition PP
- VolumeBreakdownExceptions PP
- WarningConfig PP
- CompetitionAdditionalConfig PP

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 Co	Defines how to filter out values from different dependencies. There must be one row for every key defined in this table.

	mpetition • Rounding • Stock	
Dependency Field	{name of the label in the DependencyConfiguration PP}	The logic will lookup the Dependency Configuration PP for a dependency being calculated and take a value from the 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 our 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.

Mappin	{name of the column with the depend	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

gSourceField	ency information in the source data}	not return anything even though a given product has competition data, because the name of the field is "country".

DependencyConfiguration PP

Column Name	Dependency Level Name	Depends On	Source Type	Source ID	Dimension	Currency	IsComplete	Preference #01-27
Values	{name of dependency}	{independent level name}	{type of source data for independent lookup}	{ID of the source data for independent lookup}	Data used to help describe what the dependency is used for. If one level depends on another, that relation will be treated as HQ mode.	{currency code}	Flag indicating if all data is supposed to be filled at this point. If the dependency level contains "Yes", fallback hierarchy will be stopped at this point.	User defined data. These values are 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.
Description	e.g. "Germany"	e.g. "Global"	Allowed values: • PG • XPG • PL • XPL	e.g. "155"	e.g. "Country", "Warehouse"	e.g. "EUR"	Allowed values: ▪ Yes ▪ No	e.g. "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

ExchangeRates PP

Column name	From	To	ValidDate	Rate
Values	{code of the base currency matching the DependencyConfiguration data}	{code of the target currency matching the DependencyConfiguration data}	{from which date the currency is valid}	{exchange rate value}
Description	e.g. "PLN"	e.g. "EUR"	In case of multiple entries with the same currency, the first entry after the valid date is chosen.	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

PriceSettingDimensions PP

Column name	Dimension	Order	Feature Name	Field Name
Values	• Products • Custom	• 1 • 2 • 3 • 4 • 5	▪ Fallback ▪ StrategySelection ▪ BaseStrategySelection ▪ MinMargin ▪ CostPlus ▪ PriceIncrease ▪ AdditionalDiscount ▪ RelevantCompetitors	• Products: {name of the column from the products table} • Custom: {name of the custom

			▪ DependencyAdjustment ▪ VolumeBreakdown ▪ AdjustedPriceCorridor ▪ ListPriceCorridor ▪ CostSelection ▪ Discount	value provided in Groovy code}
Description	The dimension value will be taken from: • Products: Product master data • Custom: Custom value provided by Groovy code	Hierarchy order where "1" is the most important attribute, the following ones are used for more specific lookups.	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".	

Example:

Price Parameter Values : PriceSettingDimensions [7]

Dimension	Order	Key3	Field Name
Products	1	Fallback	Business Unit
Products	2	Fallback	Product Group
Products	3	Fallback	Product Class
Products	1	StrategySelection	Business Unit
Products	2	StrategySelection	Product Group
Products	1	BaseStrategySelection	Product Class
Products	2	BaseStrategySelection	Business Unit

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.

PriceSettingLevel PP

⚠ See also Adjustments after Changing Price Setting Level.

	Pricelist	Price level

Column name		
Values	• {name of dependency level } as stated in DependencyConfiguration PP	• Gross • Gross / Net
Description		If an additional discount from a gross to net price should be calculated for a specific dependency level.

Example:

StrategyDefinition PP

Column name	Strategy Name	Level	Calculation Engine	Additional Engine Configuration	StrategyCalculationParameters	Independent Level Only	Independent Level Priority	Overridable
Values	{user friendly name of the strategy}	• Independent • Dependent	• {name of the predefined engine} or • {path to the function in groovyLibrary which will perform calculations} To learn about the	{name of PP with the engines customization}	{list of parameter names which are sent to calculation}	• Yes • No	• Yes • No	• Yes • No

			built-in pricing engines, see Price Strategies.					
Description	e.g. "CostPlus"	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.	Path should look like this: "libs.MyLib.MyElement.MyFunction"	Not all engines are customizable, so this field is nullable.	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. The parameters are passed as an input to engines, so the order is important and should not be changed for default engines.	If Yes, independent level results of this strategy will not be shown on dependent level PG/PL.	If Yes, this strategy will be above the dependent level results on a dependent level PG/PL.	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.

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	SKU	Dependent Level Name	Volume #01...#15	Discount #01...#15
Values	{sku from Product master}	{name of dependency level} as stated in DependencyConfiguration PP	Value representing the product volume that will use the given discount	Discount for the given volume
Description	e.g. "MB-0001"	e.g. "Global"	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.	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	Key1	Key2	Message	Solution	Type	Alert	Matrix	Warning
Values	{error code}	• * • {DependencyLevelName}	Any text	Any text	Any text	• Empty • "Message" • "Yellow" • "Red" • "Critical"	• Yes • No	• Yes • No
Description	Code of the warning, e.g. "NO_DIMENSIONS".	Defines which dependency this configuration is for. There should be a fallback for each error code by leaving	Message displayed to the user describing what went wrong, e.g. "Price parameter called	Solution displayed to the user describing how to fix the issue.	Type of the issue displayed to the user. It is shown	Defines how the warning will be classified: on the	Defines if the warning will be register	Defines if the warning will

	See their full list.	an entry with an asterisk. This field may be ignored if every dependency shares same warning handling.	'PriceSettingDimensions' does not exist or is empty."	It is shown only in a ResultMatrix.	only in a ResultMatrix, e.g. "Business issue"	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.	ed in a Result Matrix.	be registered in the default Warnings column.

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

CompetitionAdditionalConfig PP

Column name	Name	Mode
Values	{Setting Name}	{Setting Value}

Description	To learn about the built-in pricing engines, see Calculation Engines.

⚠ 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	

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 https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/2608955463/Error+Handling.

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	Adjust the final price.	Runtime	Red	Yes	Yes

	due to division by zero error.					
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	Error	Red	Yes	Yes

		configuration table and its data.				
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 Independent Price and this Pricing Level configured.	Check if you want to directly follow the Independent 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		Business Warning	Yellow	Yes	Yes

	manual selection.					
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 dimension configuration table.	AnyType	Red	Yes	Yes
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_INDEPENDENT_SOURCE_ID	The source ID of the specified independent 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 input (s) for margin breakeven	Verify the required parameter(s).	Runtime	Red	Yes	Yes

MISSING_INPUTS_MARGIN_BREAKEVEN_VOLUME	volume. Required: cost, base price, volume, final price.					
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.	Verify discount level configuration	Data	Red	Yes	Yes

	Unable to calculate related element(s).	table and its data.				
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_EXCHANGE_RATE_FOUND	No valid exchange rate value found.	Verify exchange rate configuration table and its data.	Data	Critical	Yes	Yes
NO_INDEPENDENT_LEVEL_CALCULATED_PRICES	No valid calculated prices from the independent level found.	Verify Prices of the independent item.	Error		Yes	Yes
NO_INDEPENDENT_LEVEL_FINAL_PRICE	No valid final price from the independent level found. Unable to calculate related element(s).	Verify Final Price of the independent item.	Error	Red	Yes	Yes
NO_INDEPENDENT_LEVEL_PRICE	No valid price from the independent level found. Unable to calculate related element(s).	Verify Final Price of the independent item.	Error	Red	Yes	Yes
NO_INDEPENDENT_LEVEL_PRICE_DECISION	No valid price decision from the independent level found.	Verify Price Decision of the independent item.	Error	Red	Yes	Yes
NO_INDEPENDENT_LEVEL_RECORD_FOUND	No price record found for independent level.	Verify the corresponding price record at the independent level source.	Error	Yellow	Yes	Yes
NO_INPUT_FOR_CORRIDOR	Unable to calculate the		Error		Yes	No

	price corridor due to missing parameter(s).	Verify the required parameter(s).				
NO_INPUT_FOR_DISCOUNT	Unable to get product discount data.	Check Discount Data.	Data		Yes	Yes
NO_INPUT_FOR_INDEPENDENT_LEVEL_ADJUSTED_PRICE	Unable to get Independent level adjusted price.	Check Independent Level Price and Dependency Level Adjustment.	Error		Yes	No
NO_INPUT_FOR_INDEPENDENT_LEVEL_PRICE_PRIORITY	Unable to get Independent 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
NO_TURNOVER_LAST_YEAR	There is no turnover in last year.	Verify the transaction data.	Data		Yes	No
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
				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			
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_INDEPENDENT_SOURCE_TYPE	The source type of the specified independent 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:

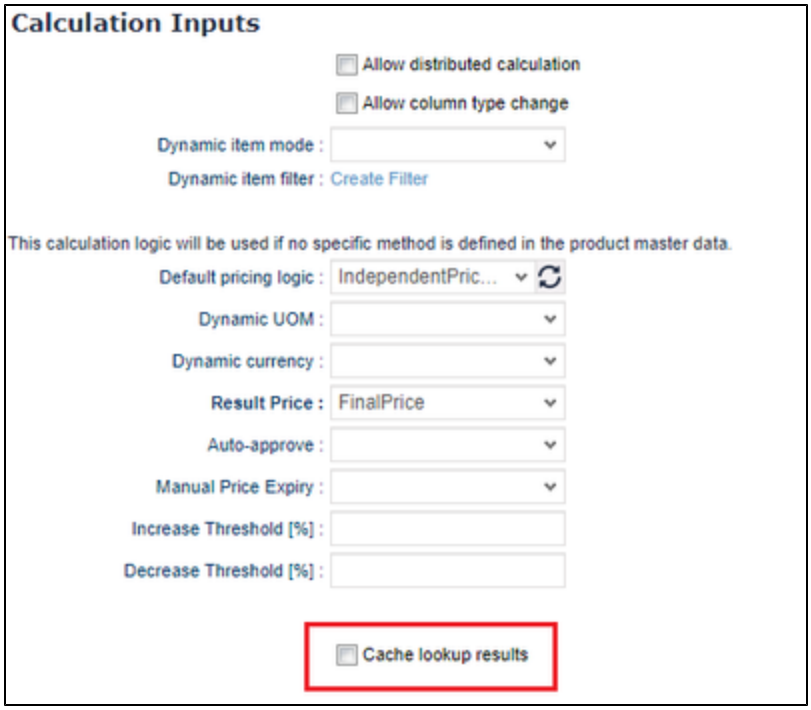
- VALIDATION
- MODULE_UNUSABLE
- NO_CONFIG
- EMPTY_CONFIG

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.

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.

Batching

All lookups are batched as described in Data Lookups.

Price Setting Configuration Wizard - Technical Design

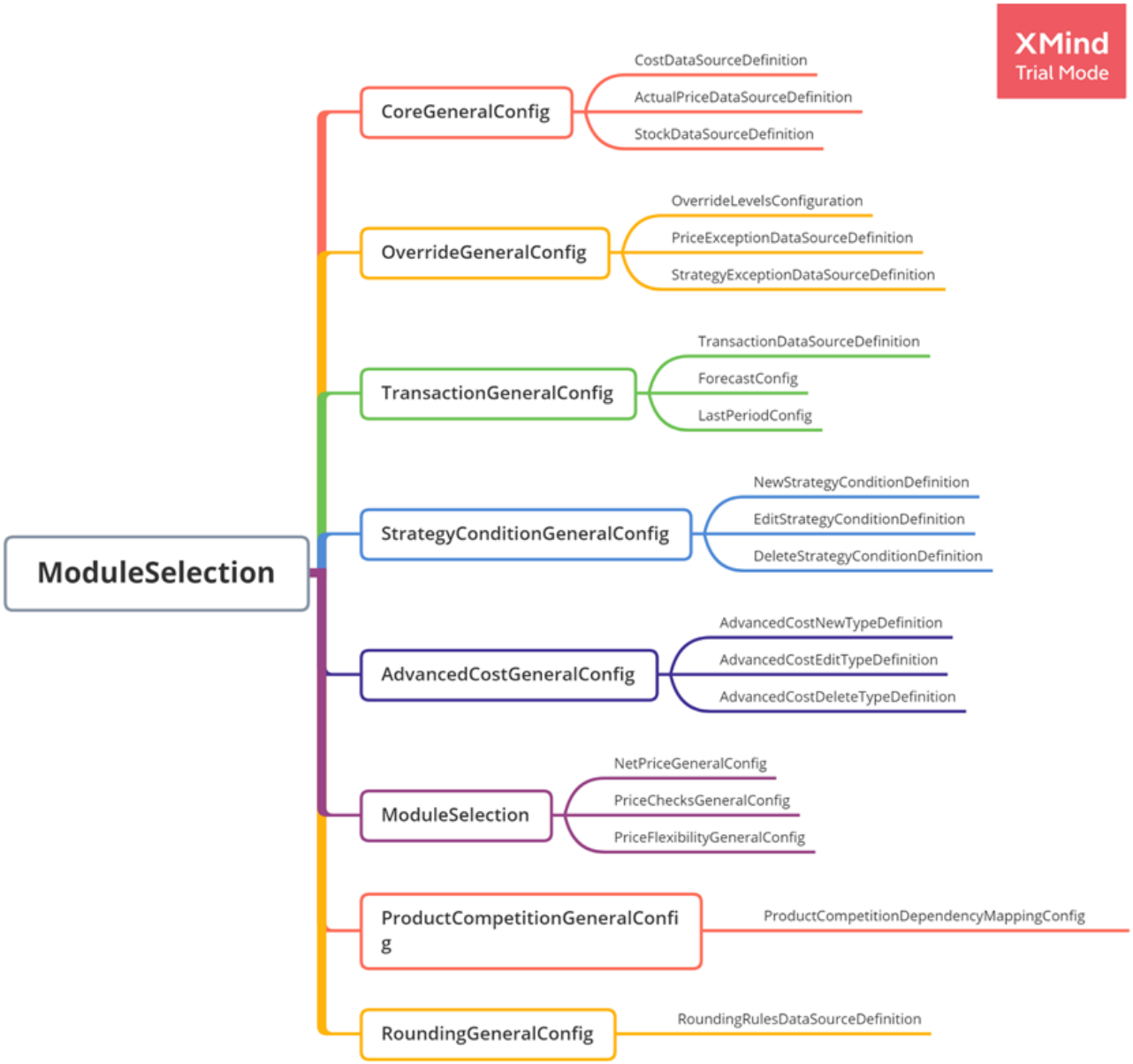
⚠ This document is for maintenance and upgradability. Do not publish it to users.

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

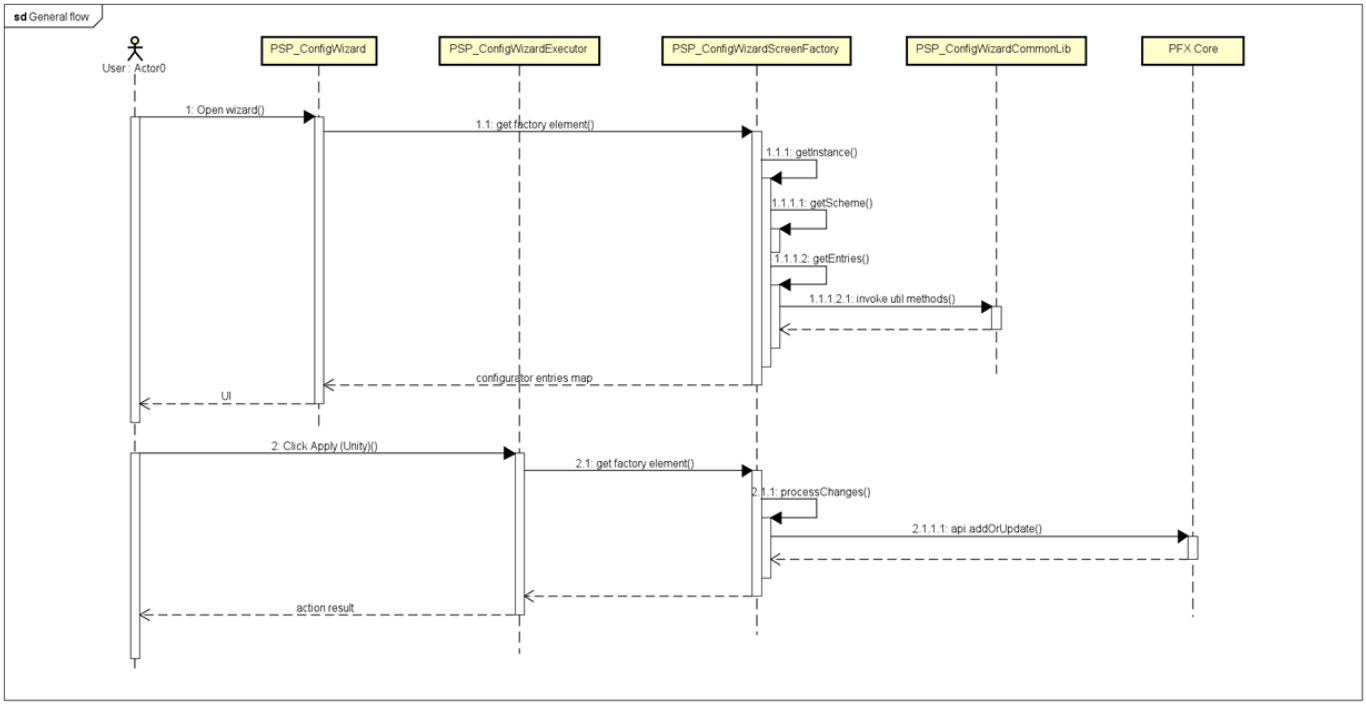
Generic logics

- PSP_ConfigWizard
- PSP_ConfigWizardExecutor

Libraries

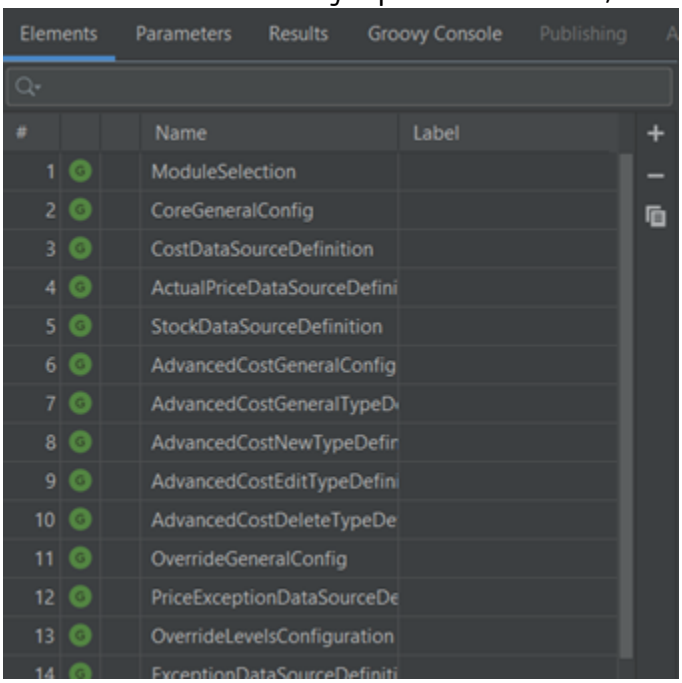
- PSP_ConfigWizardScreenFactory
- PSP_ConfigWizardCommonLib

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:

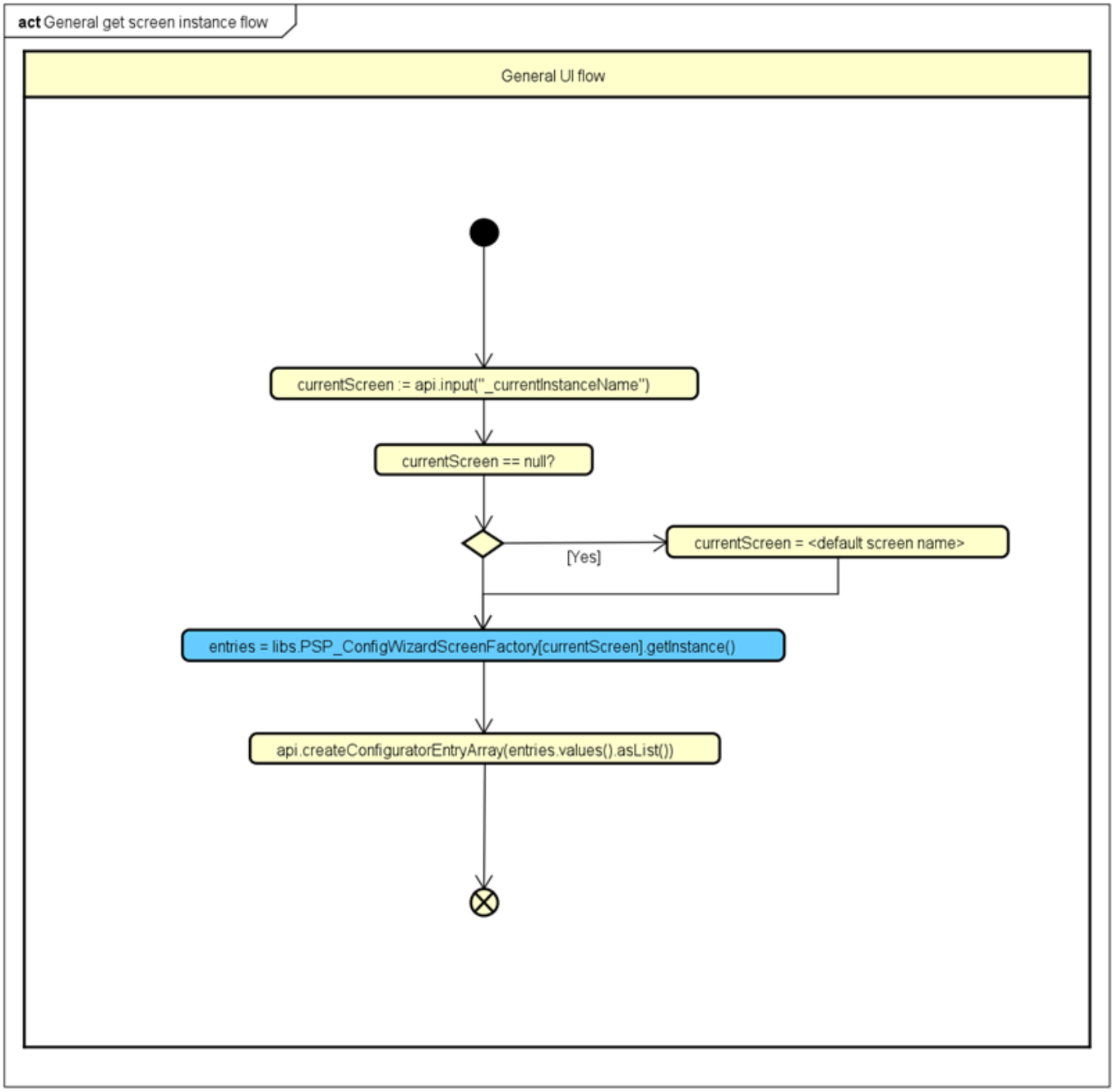
```
@Field String SCREEN_NAME = "ModuleSelection"
@Field String CORE_MODULE_NAME = "coreModule"
@Field String CORE_MODULE_LABEL = "Core Elements"

// 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() {...}
```

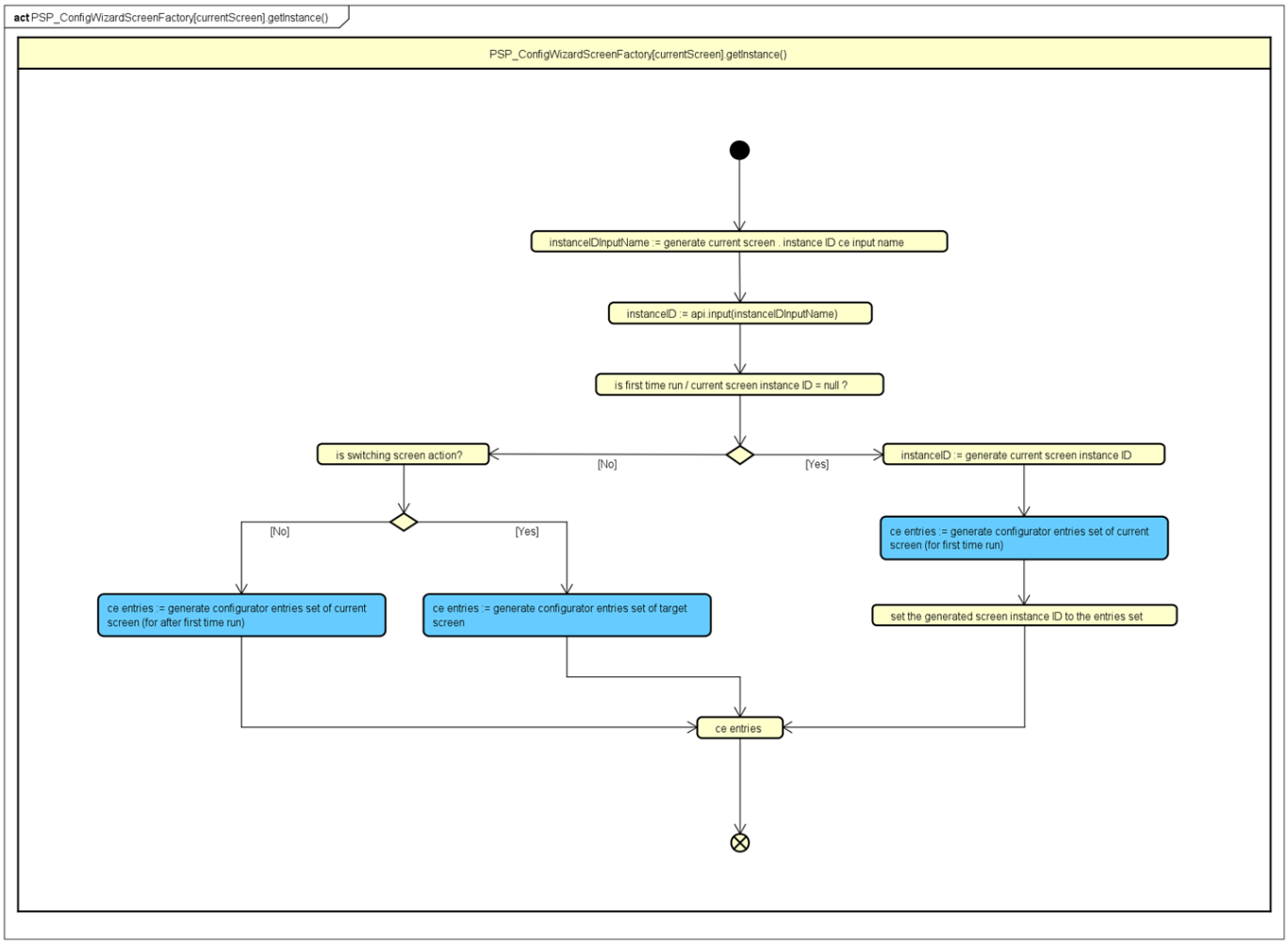
PSP_ConfigWizardCommonLib Library Structure

A set of predefined utilities which consists of:

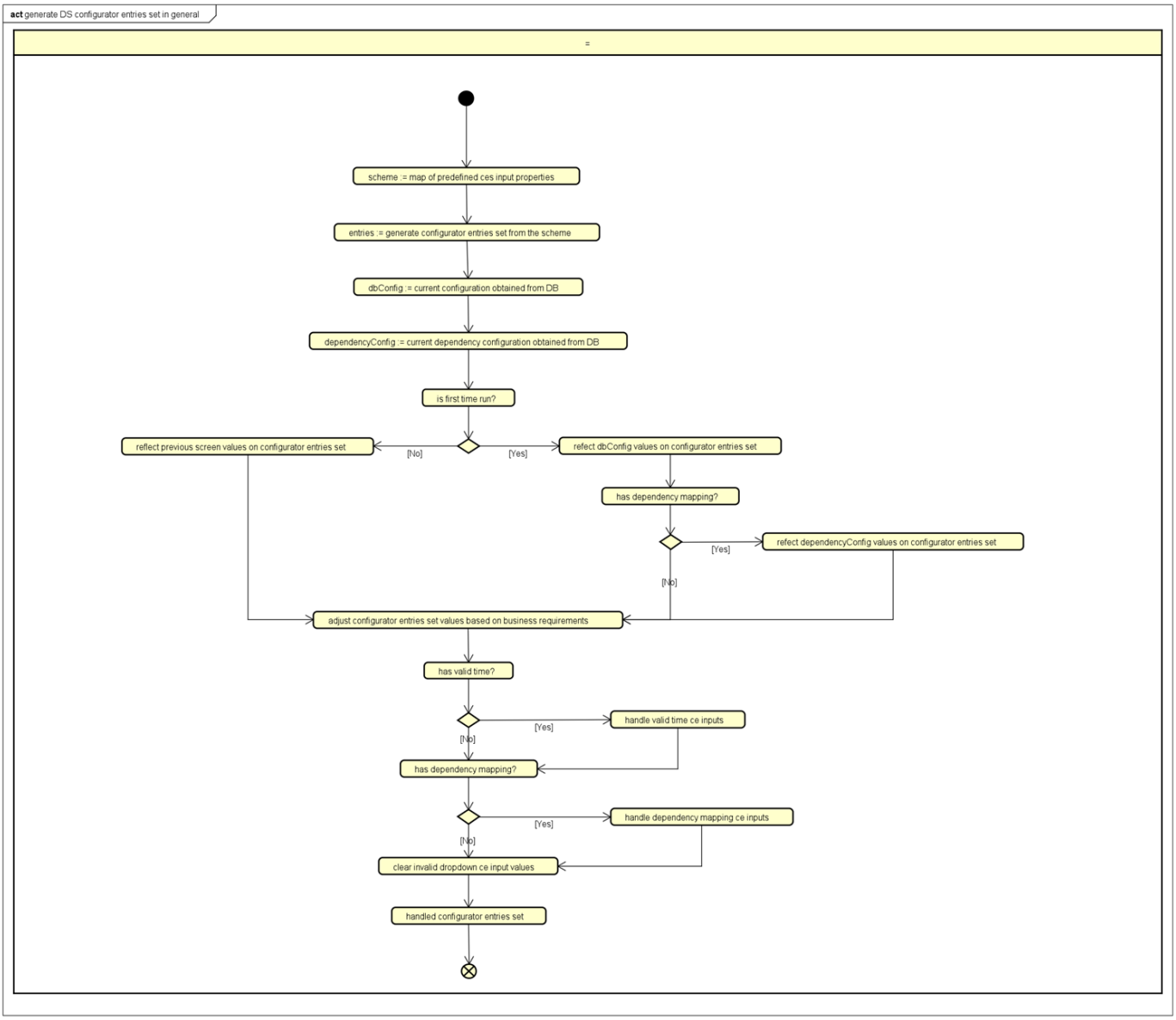
- **Constants** - Contains constants used throughout the wizard such as common input labels,...
- **HTMLDescriptions** - Contains HTML constants used throughout the wizard.
- **HTMLDescriptionUtilities** - Contains HTML handler methods.
- **Errors** - Contains error handler methods (temporarily unused).
- **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 Package)

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

How to Customize (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 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 BLOCKED).

Prerequisites

This article assumes that:

- Price Setting Accelerator is deployed to the partition and you have a working Live Price Grid configured with `IndependentPriceListLogic` where you can test your changes. Check Installation (Price Setting) 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 `IndependentPriceListLogic` 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 `IndependentPriceListLogic` 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 independent 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 BLOCKED).

In this section:

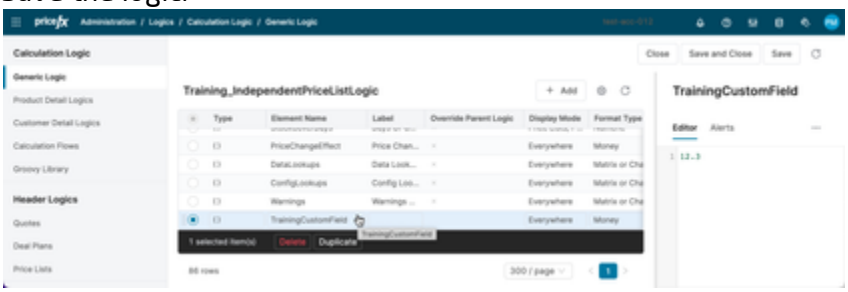
- Add Custom Field to Price List/Grid
- Modify/Override Behavior of Existing Field

Add Custom Field to Price List/Grid

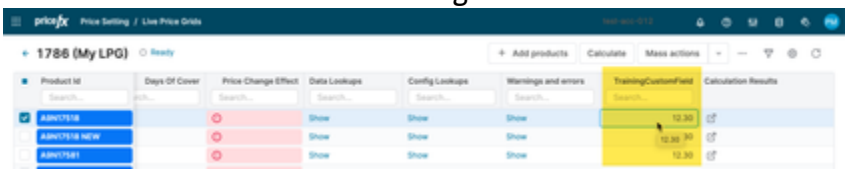
1. Navigate to **Administration > Logics > Generic Logics**.
2. Add a new generic logic:
   - **Name** - Select *Training_IndependentPriceListLogic.*
   - **Parent Logic** - Select *IndependentPriceListLogic.*
     Note: This dropdown field contains all the existing calculation logics available within 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

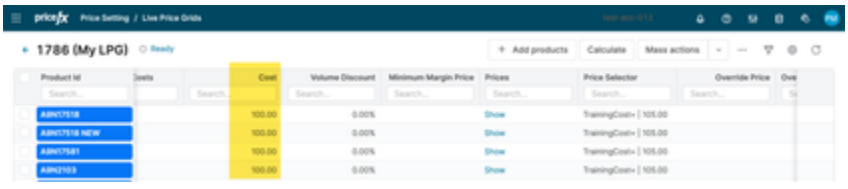
When you open your inherited logic *Training_IndependentPriceListLogic*, there is an element named *Cost* which has the code/behavior inherited from the logic *IndependentPriceListLogic*.

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_IndependentPriceListLogic*.
   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

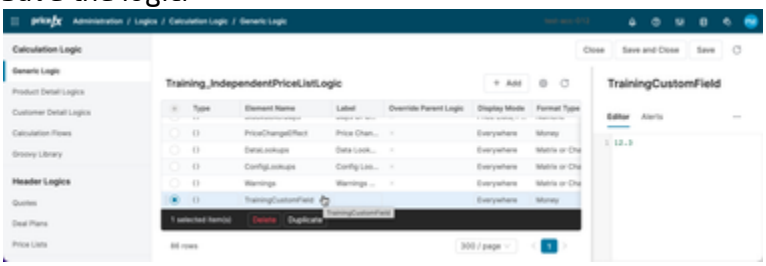
- Add Custom Field to Price List/Grid
- Modify/Override Behavior of Existing Field

Add Custom Field to Price List/Grid

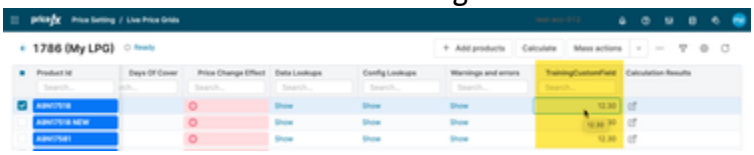
1. Navigate to **Administration > Logics > Generic Logics**.
2. Select *IndependentPriceListLogic* and duplicate it.
   - Rename it to "Training_IndependentPriceListLogic".
   - **Valid After** - Set it based on guidelines in your project, or now, for test purposes, simply use January 1st of the actual 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_IndependentPriceListLogic*, there is an element named *Cost*.

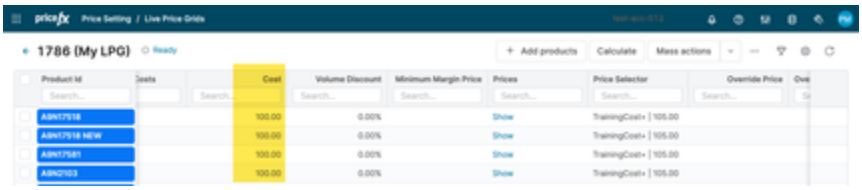
1. Find and edit your generic logic *Training_IndependentPriceListLogic*.
   a. Select the element *Cost*.
   b. Comment out the original code.

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*.

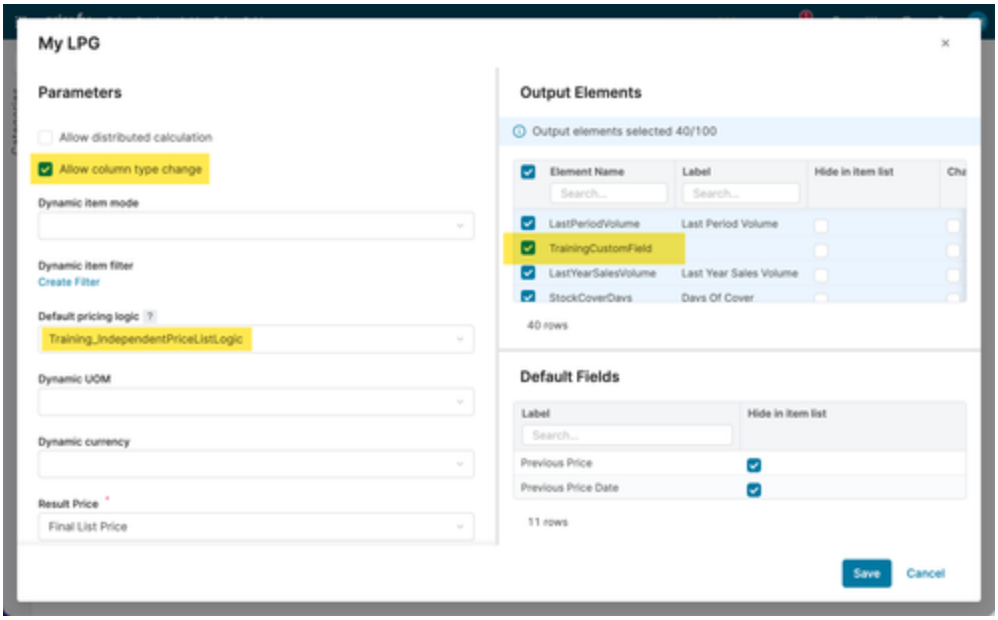


Tips on Logic Customization

- Configure Your Custom Logic
- Approach to Modifications
- Element Structure
- Actual Price Lookup - Special Case

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_IndependentPriceListLogic.*
      iii. **Independent 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 an independent/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

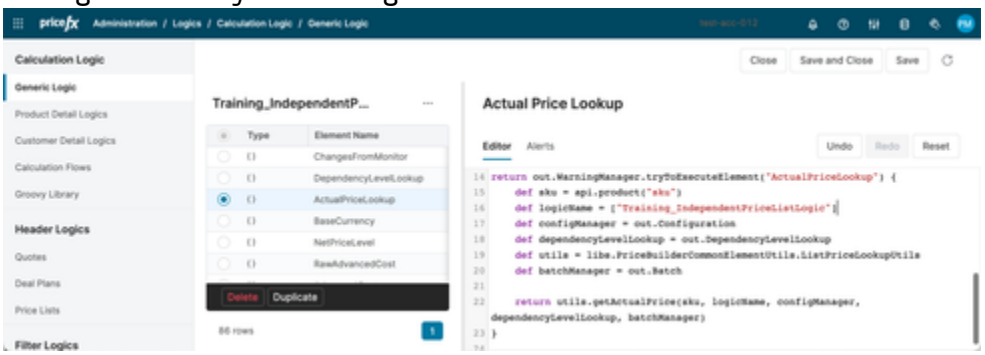
Both approaches suggest cloning Price Setting Accelerator logic. While doing so, it is important to pay attention to the element *ActualPriceLookup* which retrieves the actual price of the product.

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 *IndependentPriceListLogic.*
   b. Select the element *ActualPriceLookup.*
   c. Find the line of code which defines the logic name to be *IndependentPriceListLogic* 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:

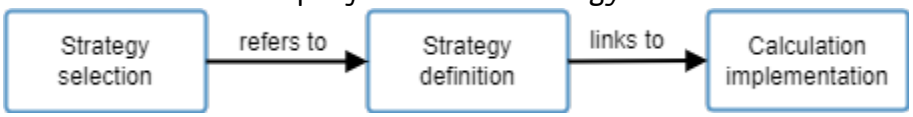
- Price Setting Accelerator is deployed to the partition and you have a working Live Price Grid configured with `IndependentPriceListLogic` where you can test your changes. Check Installation (Price Setting) for more info.
- You are a Configuration Engineer or have sufficient knowledge of Groovy coding,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

Configuration Overview

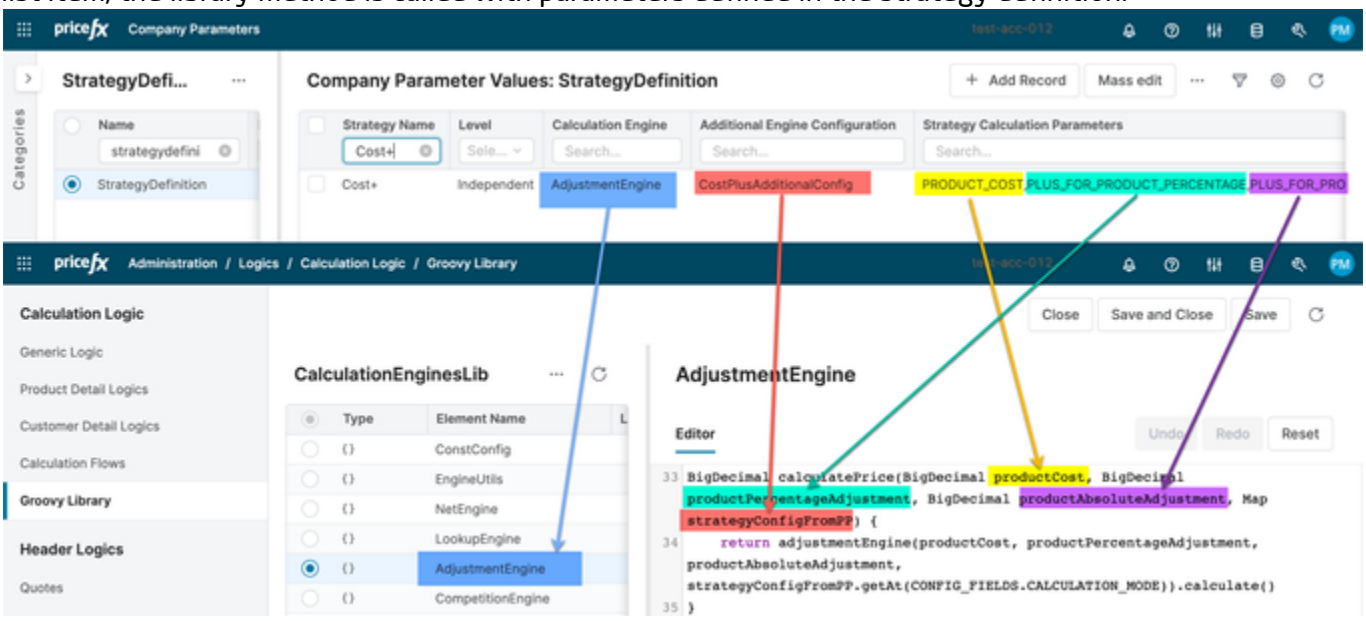
Each strategy is configured at two places:

- **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* logic˘.
   - 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

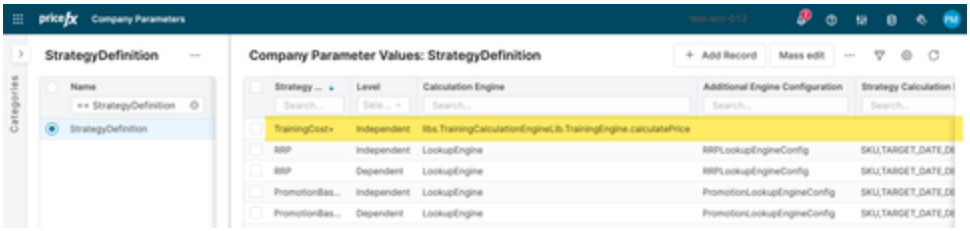
The Company Parameter *StrategyDefinition* contains a list of pricing strategies that are available in the price setting package, along with details about the strategic hierarchy, calculation engine and strategy calculation information.

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 - *Independent* 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.*
2. Add a new record to define a new strategy and link it to the new strategy implementation in Groovy library:
   - Set **Strategy Name** as *TrainingCost+* (mandatory field).
   - Set **Level** to *Independent* (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.



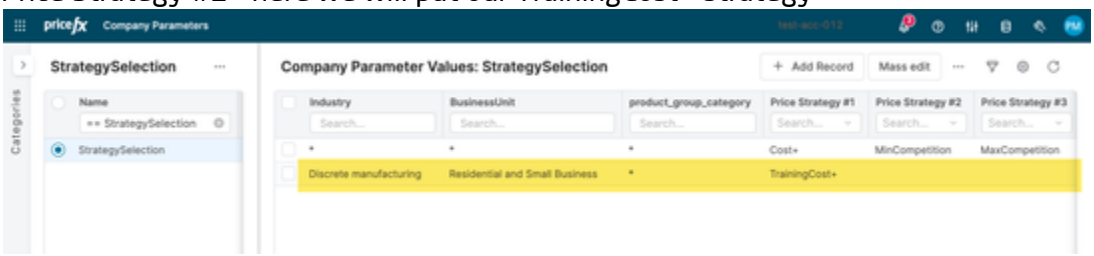
Previous step: Create New Calculation Engine Implementation

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



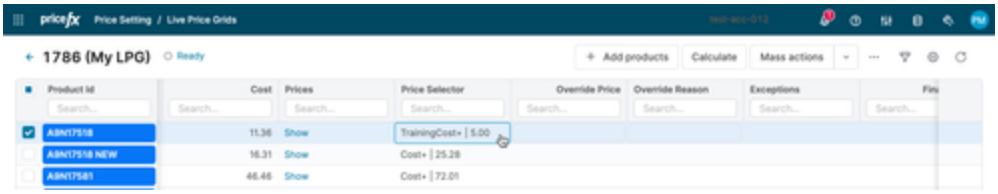
Previous step: Define New Pricing Strategy

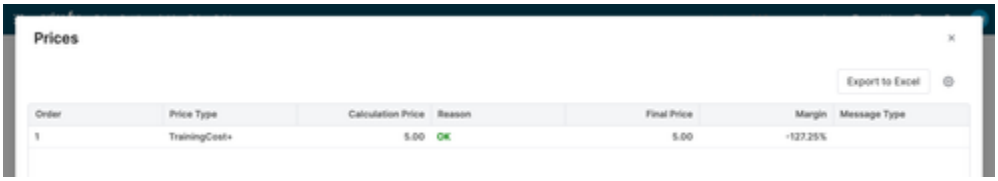
Next step: Test Calculation using Custom Pricing Strategy

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.





Previous step: Link Pricing Strategy to Business Units or Products

Next step: Set Messaging from Calculation Logic

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.

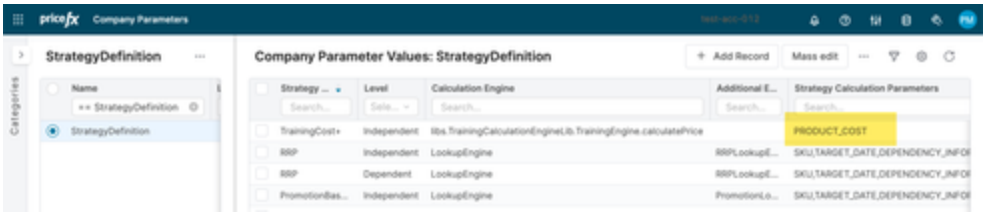
Let's assume that we want to have the cost of the product and we want to calculate a price, which should be the cost plus €5 in this case.

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

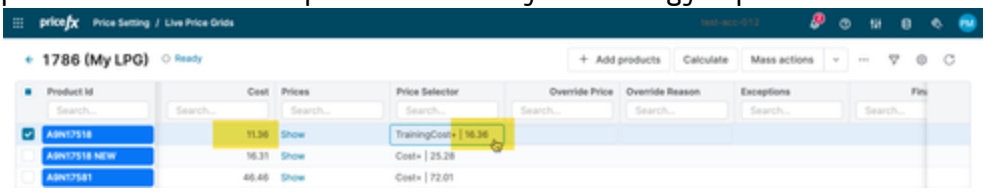
```
Map calculatePrice( BigDecimal cost ) {

    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 *PriceBuilderCommonElementUtils*, 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 *IndependentPriceListLogic* 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 independent one.
   b. Review the code of the element *AdditionalCalculationParameters*. It defines two maps, used for additional custom parameters – *additionalParameters* and *additionalOptionalParameters*. They

are explained in detail inside the element. The key information is that *parameters* should be used for static values like constants that are already available in the calculation logic and *optionalParameters* should be used for any dynamic parameters because they are lazy loaded only when an engine using them is being used.

c. Modify the definition of *additionalParameters* to provide a constant value for *OWN_PARAMETER.*

```
Map additionalParameters = [
        OWN_PARAMETER : 10.0
]
```

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.

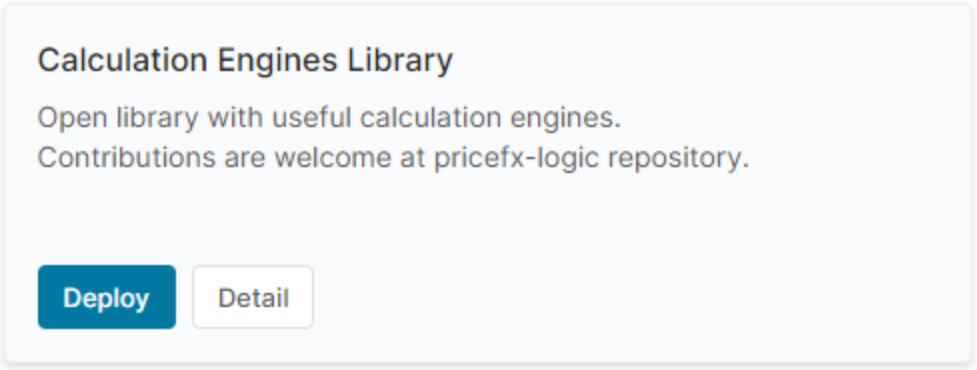
Prerequisites

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 `CalculationEngines Lib`.

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 StrategyDefinition PP.

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 `IndependentPriceListLogic` and `DependentPriceListLogic` logics.

Reference documentation worth checking out:

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

Glossary (Price Setting)

This glossary summarizes various terms used in Price Setting Accelerator.

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. Used e.g. 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 2.1.11

This document summarizes fixes introduced in the Accelerate Price Setting Package release version.

Version	2.1.11
Release Date	Jul 13, 2023

Fixed Issues

Bug Description	ID
Strategy Designer configuration has been updated with a new URL.	PFPCS-7182