Agreements Accelerator

Version 1.0.0

June 2024

Accelerate Agreements Package

Formula-based Agreement Pricing powered by the Agreements Accelerator is designed for process manufacturers involved in selling commodities and specialties. It enables the setup and renewal of customer agreements with dynamic formula-based pricing. As part of the multi-annual agreement terms, you can set the frequency of automated price recalculation to reflect changes in market conditions. It also provides forecasting functionality to see assumed agreement and business performance compared through various scenarios.

You can build a library of various formula types designed for specific purposes, such as specific pricing for territories, customer, or product groups. The formula types can be designed with visual experience. A final formula type consists of various inputs and calculation rules. Any given formulas can be extended with attributes in order to help manage and govern your formula library as well as workflow capabilities to manage an approval process on top of your formula library.

The package includes:

- Formula library
- Formula-based Agreement Types with predefined Header, Scenario, and Line Item Inputs
- Approval workflows
- Output template for generated prices
- Recalculable agreements

Key Capabilities

- Create and manage a set of formulas to enable dynamic pricing of commodities and specialties
- Establish agreements with multi-annual terms with the option to set the frequency of price recalculation
- Define calculation rules to adjust prices based on specific situations:
  - Bounding rules
  - Pricing exceptions
- Push resulting agreement price records for use within a given IT landscape (ERP, CRM, etc.)

You can also watch a video introducing Formula-based agreement pricing and its usage.

In this section:

Business User Reference (Agreements)

Agreements Accelerator provides support for formula based agreement pricing and allows you to periodically recalculate agreements.

Using this accelerator you get a head start in creating agreements and agreement renewals with regular recalculation of prices.

- Formula Management
- Working With Formula Based Agreement

Formula Management

Page under construction. To be defined: how to work with Formula Designer formulas

Formula Designer

Page under construction. To define: Formula Designer reference, how to work with it, etc.

Input-Based Formulas

Page under construction. To define: What are input based formulas and how to use them.

Define Formula

Page under **re-**construction. Feature is still supported in 1.0, but it is hidden by default.

To make the accelerator work, we need:

- **Formula type** – Stands for a calculation engine; it defines a way of calculating. Currently, the only provided type is *Cost plus*.
- **Formula definition** – Stands for a specific setup of the engine.

One formula type can be used by multiple formula definitions and formula definitions provide parameters to the engine, such as lookup parameters, weights, etc.

Formula Type

Formula Types are defined in the Company Parameter table called AGR_FormulaTypeDefinitions. There are 4 columns:

- **Formula Type Name** – Specifies the name of the formula.
- **Calculation Engine Path** – Specifies how the engine is calculated. Provided in the format *libs.<name of lib>.<name of element>.<name of method>,* for example: libs.AGR_FormulaEnginesLib. AGR_Formula_CostPlus.calculate
- **Engine Calculation Parameters** – Parameters for the Calculation Engine, some of them are predefined, most of them are input names from Input Manager.
- **Input Generation Table Name** – Name of the Company Parameter table that contains definitions of inputs to be shown on the Formula Definition (as well as broadcasted to the Agreement). The table has to follow the Input Manager input definition table setup.

Formula Definition

1. Go to **Agreements & Promotions** > **Formula Definition**.
2. Click **Create New Formula Definition**.
3. Select what **Formula Type** (calculation engine) the definition should be based on.
   Currently, the only option available is *AGR_Formula_CostPlus*, so all of these options relate to it.
   - Cost Plus general fields:
     - Select the type of the structure where your **data** is located. Currently, the only supported data source for costs is *PX (Product Extensions)*.
     - Select the name of the **table** where you have your cost data. Currently, it has to be PX table.
     - Select the column where the **Cost** is stored.
     - Select the column where the **Currency** is stored.
     - Select the column where the **UoM** is stored.
     - Select the column where **Valid From** is stored.
     - Select the column where **Valid To** is stored.
   - Cost Plus calculation specific fields:
     - Select **Adder Type** – either ABSOLUTE or PERCENTAGE.
     - **Base Adder** – Specifies the amount (either as an absolute value or percentage) which should be added to the price. It uses the default currency and default UOM defined at the header of an agreement.
       Note: You can override the adders on line item level in the agreement and using exceptions for specific products.
     - **Bounds** – Represent a mechanism to prevent too frequent price changes – prices get updated only if the price change is big enough. Bounds use comparison with the *previous result* which is the engine result of the previous period for the same agreement and the same SKU using the same currency. The previous result is then compared to the *calculated result* for the current period and their difference is calculated (previous minus calculated). If the difference is greater than the defined bounds, the newly calculated price will be used. Both bounds are specified as positive numbers. See an example below.
       - **Upper Bound** – The newly calculated price is used only if it *grew* more than the amount specified here.
       - **Lower Bound** – The newly calculated price is used only if it *dropped* more than the amount specified here.
         Note: You can override the bounds with exceptions for specific products.
     - **Rolling Period** – Allows you to calculate an average price over the defined period of time. For example, if Rolling Period is 3 months, it looks at the current date and then takes the previous 3 months *plus the current one* and uses the costs found in those 4 months for calculating the average. If Lag Period (see below) is defined too, it is applied first. That means that first the current date is moved by the defined number of months/quarters and only then the previous months/quarters are looked up.
       - **Rolling Period Type** – Either Month or Quarter.
       - **Rolling Period Amount** – Defines over how many months/quarters to calculate the average price.
     - **Lag Period** – Moves your cost lookup by the defined number of months/quarters back from the current date.
       - **Lag Period Type** – Either Month or Quarter.
       - **Lag Period Amount** – Defines how many months or quarters to move back.
4. Once you finish the Formula Definition, you need to submit it for approval.

Bounds Examples

Example 1	Example 2	Example 3

Scenario: • Previous price = 12 • New calculated price = 14 • Upper Bound = 5 • Lower Bound = 2 Result: Difference between previous and calculated price = 2 It is a price increase, so we apply Upper Bound. The price difference is not bigger than the Upper Bound value and so the previous price stays in place.	Scenario: • Previous price = 12 • New calculated price = 19 • Upper Bound = 5 • Lower Bound = 2 Result: Difference between previous and calculated price = 7 It is a price increase, so we apply Upper Bound. The price difference is bigger than the Upper Bound value and so the new calculated price is used.	Scenario: • Previous price = 12 • New calculated price = 9 • Upper Bound = 5 • Lower Bound = 2 Result: Difference between previous and calculated price = 3 It is a price drop, so we apply Lower Bound. The price difference is bigger than the Lower Bound value and so the new calculated price is used.

Working With Formula Based Agreement

Page under construction. To define: Everything the user can do inside the agreement.

Apply Formula to Agreement Line Item

Page under **re-**constructon. To define: Step by step how selecting formula is envisioned. Or split into pages about various pieces

1. Go to **Agreements & Promotions** > **Agreements & Promotions**.
2. Expand the **New Agreement & Promotion** button and select the option **Formula Based Pricing**.
3. In the header fill in the following fields:
   - **Start/End Date** – Defines validity period of the agreement. According to this period, agreements to recalculate are selected (by feeder logic in Calculations).
   - **User Group (View Details / Edit)** - If the agreement's visibility and editability should be restricted to certain user groups, enter their names.
   - **Customer** – Currently not used.
   - **Calculation Currency** / **UOM** – Define currency and UOM for this agreement.
   - **Forecast Engine** – Allows you to select a Forecast Engine which calculates forecast revenue, and, if needed, also cost or volume.
   - **Additional Calculation Output Types** – Allow you to specify additional parameters for the result. For example, you can have the price generated in another currency or using different bounds (explained below). For each combination, a set of price records is generated.
4. On the Items tab, there is a predefined scenario and predefined line item. (One of each must always be present in the agreement. If you delete the scenario and click Recalculate, it will be populated back.) Use this line or create your own.
   - In the table you have the following options for a scenario:
     - **Active** checkbox indicates that this scenario should be used for recalculations and for price record generation.
     - **Calculation Period** specifies how often the price records are recalculated: monthly or quarterly. The logic looks through all contracts, finds those with active scenarios and then

checks their calculation periods. If there is a price record generated in the current month /quarter, then no generating takes place. But if there isn't one, then a new price record for the current month/quarter is generated.
    - Note that **Calculation Currency** and **Calculation Unit of Measure** are propagated from the header. If they are changed in the header, the change will be reflected here only after recalculation.
5. Add required line items into a scenario. For each item fill in the following fields:
    - Input parameters (panel on the right side):
        - **Product(s)** – Specifies what Product Group will provide SKUs to use for the recalculation. The number of products and the number of Additional Calculation Output Types directly translate into how many price records are generated.
        - **Formula** – Allows you to select the appropriate formula definition (has to be approved).
        - The following inputs (Adder Type, Base Adder, Exceptions) are broadcasted from the Cost Plus formula, so they will not always be here, if a different formula is used.
            - **Override Adder Type** – Allows you to enter a different Adder Type than in the formula definition.
              Note that this value can still be overridden if you define an exception for selected products (see below).
            - **Override Base Adder** – Allows you to enter a different Base Adder than in the formula definition.
              Note that this value can still be overridden if you define an exception for selected products (see below).
            - **Product Exceptions** – Allows you to specify products which should be treated differently (different adders, different bounds, different currency). The priority then is: 1. Exceptions, 2. overrides, 3. base (the one in formula definition).
6. Once the agreement is recalculated, you can go to the Calculations tab of the panel on the right. It shows **Calculation Outputs Result Matrix** which summarizes all the combinations: SKU, resulting price, currency, UOM, formula detail and message.

Formula Forecasts

Page under construction. To define: How to work with forecasts, what forecasts are available OOTB

Scenario Comparisons

Page under construction: How it works, what analysis you can do with it.

Calculate or Recalculate

Page under **re-**construction: To define how recalcualtions work and how to review **Condition** Records

All (re)calculations are done using the calculation currency and calculation UOM (those are defined at the header of an agreement). That implies that you will get the results only if your PX table has the cost defined for the calculation currency/UOM.

1. Go to **Agreements & Promotions** > **Calculations** and open the "Agreement Recalculation" Calculation object.
2. Open the calculation and go through all its tabs:
    - **Schedule** – Allows you to set up how often the recalculation should run.

- **Agreements & Promotions** – Allows you to define a list of agreements to recalculate. The agreement must be approved in order to be recalculated.
  Note: Selection by the feeder logic (see below) has priority over whatever is selected here.
- **Calculation** – Allows you to select the logic and the feeder logic. The feeder logic provides a list of agreements to recalculate. These agreements must fulfill these conditions:
  - Have "Formula based pricing" as Header Type.
  - Be approved.
  - Be valid (fit in the range of from-to dates).
  - If there is a price record generated in the current month/quarter, then no generating takes place. But if there isn't one, then a new price record for the current month/quarter is generated.
  - If the agreement has a Price Record with an error, it is recalculated.

3. Once everything is set up, click the **Save** button.
4. Let the calculation run according to the schedule.
   Or, if you want to trigger the calculation manually, click **Run Calculation**.
5. After the calculation is completed, you can view the results in the Calculations & Details section on the right. After you click the link, you will get Calculation Outputs Result Matrix. It shows a union of all line items results in the active scenario and allows you to evaluate which formula to use for the price record generation. Calculation Output Types take the calculated price (based on the header currency) and use conversion rates to show prices in different currencies and UOMs.
6. Only after the (re)calculation runs, the price records are generated. (This is a change compared to the previous version – price records used to be generated immediately after an agreement was approved.)

Note the following differences between Calculation and Recalculation:

- Calculation on the agreement level does not create any Price Records, it is only a preview of what the results could be, based on the setup. Price Records are generated only when Recalculate is run for the agreement.
- The Recalculation (the periodic one) which is done by Calculation object generates Price Records based on the criteria set in the approved agreement. What gets recalculated is controlled by the feeder logic and the schedule controls how often the feeder is run.

Review Price Records

1. Go to **Agreements & Promotions** > **Price Records**.
2. There can be multiple records for a single product – depending on how many parameters combinations you have defined.
3. What fields are especially useful to check:
   - **Formula Definition Name** – Identifies which formula definition was used.
   - **Formula Type** – Identifies which formula type was used.
   - **Base Adder Type** – Identifies which adder type was defined in the formula definition.
   - **Valid From** / **Valid To** – Represent the defined calculation period (month or quarter).
   - **Output Currency/UOM** – They (together with other fields) distinguish variants of the product price.
   - **Calculated Result** – Result of the calculation with just adders. Bounds are not reflected here.
   - **Engine Result** – Result of the calculation with adders and bounds.
   - **Converted Result** – Reflects the currency conversion (from base currency to output currency). It is calculated as Engine Result times whatever value is in Exchange Rate or Conversion Rate columns.
     Note: The conversion is looked up on the day of recalculation, that is why the results may vary between the agreement and Price Records (if those are generated later on and if the conversion rates changed).

- **Formula Detail** – Shows you details of the calculation.
  For example: *50 + 5 (PRODUCT_EXCEPTION) with 3 MONTH Rolling Period with 1 MONTH Lag Period*
  That is: initial cost + defined adder of 5 (either as percentage or absolute value – as defined) where the price is looked up one month back from the current date (Lag) and averaged using values found in previous 3 months (Rolling Period; this period is moved back by Lag)
- **Calculation Status** – Either OK or Error.
- **Message** – If it says "No cost found", check whether your Product Extension table has cost defined for the given combination of currency and UoM or whether the validity fits the defined range.

Admin User Reference (Agreements)

Installation (Agreements)

This section will guide you when installing the Agreements Accelerator package in your environment.

Installation Steps

1. Get access to PlatformManager and target partition, as described in common installation prerequisites.
2. Go to PlatformManager at https://platform.pricefx.com/ and log in.
3. Go to **Marketplace** and find the *Agreements* accelerator.
4. Click the accelerator tile, select the partition where you want to deploy the accelerator package, and confirm the deployment dialogue to start.
   For detailed description of all deployment options, see PlatformManager documentation.

Post-installation Steps

To ensure proper functioning of the Agreements package, you have to go through the steps below. After completing this part, you can start configuring the calculation logics according to your needs.

Setup the Hidden Group Prefix Feature Flag

Some outputs need to be passed between Header and Line Item logics, but remain hidden from the user, to make sure this works properly the Feature Flag called "hiddenGroupPrefix" needs to be set to "Hidden_Group". The Feature Flag can be found in Administration Feature Flags.

Setup proper create behavior of Custom Forms

Navigate to Administration Configuration Custom Forms General Settings and set the "Create Custom Form Option" to "Save and Calculate pre-phase for new Custom Form immediately". This is necessary to properly pass values between the Custom Form tabs.

Setup Historical Data

In order to properly utilize forecasting features available in the package the Historical Data Datamart needs to be properly setup.

First prepare a Datamart that will contain the Historical Data that will be used for Forecasts. The Datamart has to have certain columns:

- ProductId - columns under which Product Id (sku) can be found
  - Data Type: Text
- CustomerId - columns under which Customer Id can be found
  - Data Type: Text
- Margin - columns under which Margin can be found
  - Data Type: Money
- Cost - columns under which Cost can be found
  - Data Type: Money
- Currency - columns under which Currency Code (EUR, USD etc.) can be found
  - Data Type: Currency
- UnitOfMeasure - columns under which Unit of Measure Code (KG, LB, PCS, etc.) can be found
  - Data Type: UOM
- PricingDate - columns under which Pricing Date can be found
  - Data Type: Date
  - Function: Pricing Date
- Volume - columns under which Volume can be found
  - Data Type: Quantity
  - Function: Per Unit Basis

Navigate to Advanced Property called "AGR_Historical_AdvancedConfiguration" under Administration Configuration Advanced Configuration Options. There will be a JSON file with properties, that need to be filled in accordingly to the setup of the Datamart.

The fields are as follows:

- sourceType - for now only "DM" value is supported, this points that Historical Data is stored in a Datamart
- sourceName - name of the Historical Data Datamart
- productIdFieldName - name of Product Id (sku) column
- customerIdFieldName - name of Customer Id column
- marginFieldName - name of Margin column
- costFieldName - name of Cost column
- currencyFieldName - name of Currency Code column
- unitOfMeasureFieldName - name of Unit of Measure Code column
- pricingDateFieldName - name of Pricing Date column
- volumeFieldName - name of Volume column
- defaultCalculationCurrency
  - Defines the default Calculation Currency (should reflect the Base Currency of the DM)
- defaultCalculationUOM
  - Defines the default Calculation Unit of Measure (should reflect the Base Unit of Measure of the DM)

Once this setup is complete then various Forecast Engines can be used, for example included with the package Average Forecast (but not Cost Plus, this one needs more setup described below).

Setup Cost Plus Forecast Data (optional)

In order to properly utilize Cost Plus forecasting features available in the package the Forecast Cost Data Price Extension as well as Volume Data Price Extension needs to be properly setup.

First create a Product Extension for Cost Data with the following mandatory columns:

- Product Id (sku)
- Cost
- Currency
- Unit of Measure
- Period
  - The values in this column should be stored in the DM Price Date Month format, which is yyyy-MM

Navigate to Advanced Property called "AGR_CostForecast_AdvancedConfiguration" under Administration Configuration Advanced Configuration Options. There will be a JSON file with properties, that need to be filled in accordingly to the setup of the Price Extension.

The fields are as follows:

- sourceType - for now only "PX" value is supported
- sourceName - name of the Price Extension
- productIdFieldName - name of the column containing the Product Id (sku)
- costFieldName - name of the attribute that holds Cost value
- currencyFieldName - name of the attribute that holds Currency Code value
- uomFieldName - name of the attribute that holds Unit Of Measure value
- periodFieldName - name of the attribute that holds Period value

Then create a Product Extension for Volume Data with the following mandatory columns:

- Product Id (sku)
- Volume
- Unit Of Measure
- Period
  - The values in this column should be stored in the DM Price Date Month format, which is yyyy-MM

Navigate to Advanced Property called "AGR_VolumeForecast_AdvancedConfiguration" under Administration Configuration Advanced Configuration Options. There will be a JSON file with properties, that need to be filled in accordingly to the setup of the Price Extension.

The fields are as follows:

- sourceType - for now only "PX" value is supported
- sourceName - name of the Price Extension
- productIdFieldName - name of the column containing the Product Id (sku)
- volumeFieldName - name of the attribute that holds Volume value
- uomFieldName - name of the attribute that holds Unit Of Measure value
- periodFieldName - name of the attribute that holds Period value

Proper setup of these two PX makes it possible to use Cost Plus Forecast Engine on Agreements.

Setup Cost Plus Data (optional)

In order to properly utilize Cost Plus Input-Based Formula available in the package the Cost Data Price Extension needs to be properly setup.

First create a Product Extension with the following columns:

- Product Id (sku)
- Cost
- Currency

- Unit of Measure
- Valid From
  - Holds the start of validity period for given Cost/Currency/UOM pairing
- Valid To (Optional)
  - If left empty (or not mapped) then there's no "expiration date" set on Cost/Currency/UOM pairing

Once the PX with Cost Data has been setup it can be utilized in the Cost Plus Input-Based Formula, which is described here [Cost Plus formula article link]

Setup additional configuration options (optional)

There is a number of additional configuration options available in the package. To configure them navigate to Advanced Property called "AGR_AdvancedConfiguration" under Administration Configuration Advanced Configuration Options. There will be a JSON file with properties that can be modified to suit the needs.

The properties are:

- defaultCalculationCurrency - defines the default Currency in which all Agreements will be calculated. This influences the preselection on Header of an Agreement.
- defaultCalculationUOM - defines the default Unit of Measure in which all Agreements will be calculated. This influences the preselection on Header of an Agreement.
- forecastMonths - defines the forecasting period in months. Requires an integer value, ex. 12 means all forecast engines will be run 12 times producing a yearly forecast.
- maximumAmountOfRecalculationAttempts - if a Condition Record fails (ends up in ERROR status, for example due to missing cost) then it will be picked up again by the Feeder logic, up to maximum times that this value defines. For example 3 means that the Record will be recalculated 3 times until it will be marked with MAXIMUM_RECALCULATION_ATTEMPTS_REACHED status and no longer picked up by Feeder logic.

Setup Formula Designer Additional Information

Formula Designer allows to return custom information from all formulas. The additionalInfoFields parameter should be filled based on project needs (more info here: https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/5225775151/Formula+Designer+Agreements#Advanced-Configuration).

The parameter additionalInfoFields is a list of maps that follow certain structure:

```
{
  "name": "formulaDetail", <--- provides the name of the returned value
  "label": "Formula Detail", <--- provides the label displayed on the
block
  "type": "string" <--- defines the type of the value, can also be set
to "number"
}
```

This mechanism is designed to allow passing values from Formula Designer to Condition Records, if necessary [article how to].

Setup Approval Workflow

The package workflow is fully based on Approval Workflow Package: Overview (Approval Workflow)

There are three Workflow logics which utilize the following Workflow Types that are required for Approval Workflow setup (ApprovalWorkflowSetup table):

- Agreement workflow uses "FormulaBasedPricing"
- Formula workflow uses "Formula"
- Input-Based Formula workflow uses "InputBasedFormula"

Assign Business Roles

To give users view and edit access to parts of the accelerator, use the available business roles or incorporate them in your own permission setup.

Business Roles (Agreements)

The Agreements Accelerator is delivered together with the following business roles:

Business Role	Included Permissions
[AGR] Sales Agent	• Manage Agreements & Promotions • View Agreements & Promotions • View Formula Designer • View Custom Form • View Dashboards
[AGR] Formula Manager	• Edit Formula Designer • Manage Formula Designer • Manage Agreements & Promotions • View Agreements & Promotions • View Formula Designer • Manage Custom Form • Edit Custom Form • View Custom Form • View Dashboards

Technical User Reference (Agreements)

- Company Parameters (Agreements)
- Architecture Diagram (Agreements)
- Calculation Parameters (Agreements)
- Formula Designer (Agreements)
- How To Articles (Agreements)
- Error Management (Agreements)

Company Parameters (Agreements)

- AGR_FormulaTypes
- AGR_InputBasedFormulaTypes

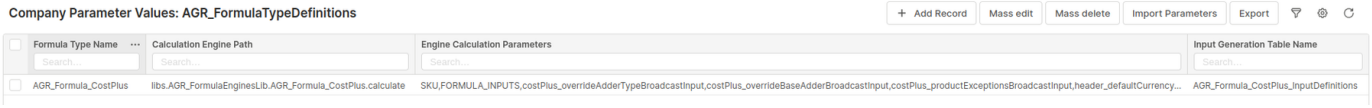
- AGR_ForecastTypes
- AGR_FormulaBasedPricingHeader_InputDefinitions
- AGR_FormulaBasedPricing_InputDefinitions
- AGR_Formula_CostPlus_InputDefinitions
- AGR_Formula_ExceptionsConfigurator_InputDefinitions
- AGR_Forecast_CostPlus_InputDefinitions
- AGR_Forecast_Average_InputDefinitions
- AGR_FormulaHeaderAttributesConfigurator_InputDefinitions
- AGR_Dashboard_ScenarioComparison_InputDefinitions
- AGR_WarningConfig

AGR_FormulaTypes

Defines Formula Types and how they calculate the price. Automatically populated by Formula Designer on formula submission.

Fields:

- **Formula Type Name** - Stands for the name of an individual Formula Type. These values can then be selected in the **Formula type** dropdown menu in the Formula Definition. Currently, the only type provided out of the box is *Cost Plus*. You can add your own type, for details see How to Add New Input-Based Formula Type with Calculation Method and Inputs.
- **Calculation Engine Path** - Path to the calculation engine in the format `libs.<name of lib>.<name of element>.<name of method>`.
- **Engine Calculation Parameters** - Parameters which values will be passed to the calculation as parameters of method provided in `Calculation Engine Path`. The static ones are: SKU (the product being calculated) and FORMULA_INPUTS (all of the inputs from the Definition step).
- **Input Generation Table Name** - Refers to another Company Parameter table which defines the Input Manager input generation.

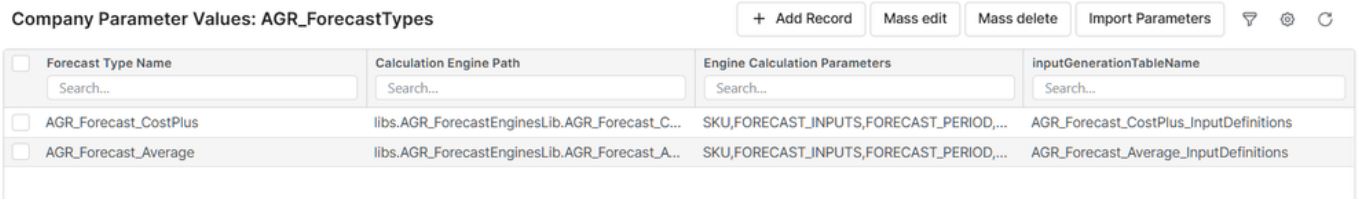


AGR_InputBasedFormulaTypes

Works exactly the same way as `AGR_FormulaTypes`, but manages Input-Based Formulas, so it needs to be filled in manually.

AGR_ForecastTypes

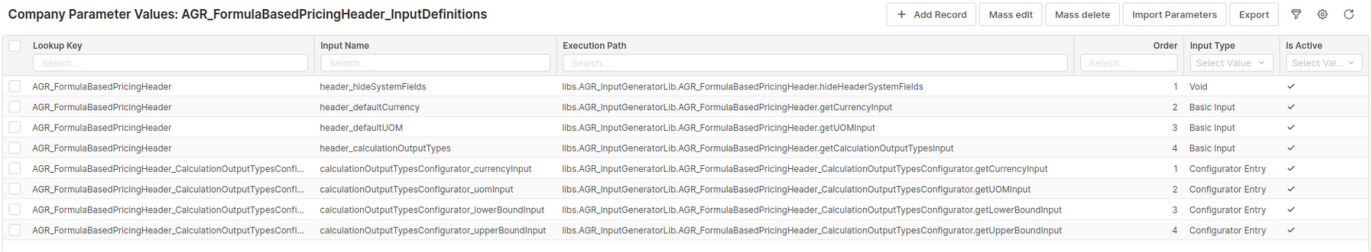
Provides values to select from in the Forecast Engine dropdown in the agreement header. Works the same way as `AGR_Formulas`. Only manually configured Forecast Types are supported.



AGR_FormulaBasedPricingHeader_InputDefinitions

Allows you to customize the inputs in the header. Details abut usage can be found in How to Modify, Add and Remove Inputs.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.



Company Parameter Values: AGR_FormulaBasedPricingHeader_InputDefinitions

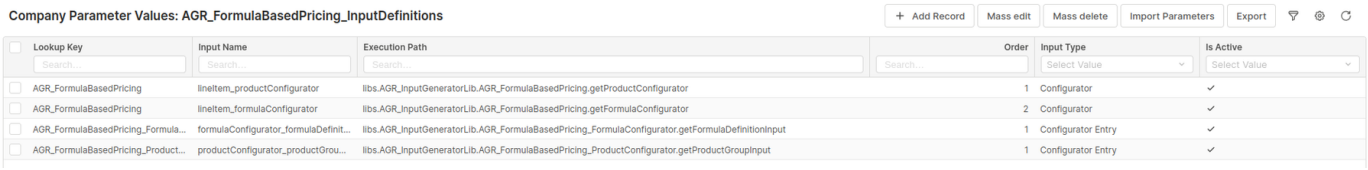
+ Add Record | Mass edit | Mass delete | Import Parameters | Export

Lookup Key	Input Name	Execution Path	Order	Input Type	Is Active
AGR_FormulaBasedPricingHeader	header_hideSystemFields	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader.hideHeaderSystemFields	1	Void	✓
AGR_FormulaBasedPricingHeader	header_defaultCurrency	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader.getCurrencyInput	2	Basic Input	✓
AGR_FormulaBasedPricingHeader	header_defaultUOM	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader.getUOMInput	3	Basic Input	✓
AGR_FormulaBasedPricingHeader	header_calculationOutputTypes	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader.getCalculationOutputTypesInput	4	Basic Input	✓
AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfi...	calculationOutputTypesConfigurator_currencyInput	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfigurator.getCurrencyInput	1	Configurator Entry	✓
AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfi...	calculationOutputTypesConfigurator_uomInput	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfigurator.getUOMInput	2	Configurator Entry	✓
AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfi...	calculationOutputTypesConfigurator_lowerBoundInput	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfigurator.getLowerBoundInput	3	Configurator Entry	✓
AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfi...	calculationOutputTypesConfigurator_upperBoundInput	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricingHeader_CalculationOutputTypesConfigurator.getUpperBoundInput	4	Configurator Entry	✓

AGR_FormulaBasedPricing_InputDefinitions

Allows you to customize the inputs on the line item level. Details abut usage can be found in How to Modify, Add and Remove Inputs.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.



Company Parameter Values: AGR_FormulaBasedPricing_InputDefinitions

+ Add Record | Mass edit | Mass delete | Import Parameters | Export

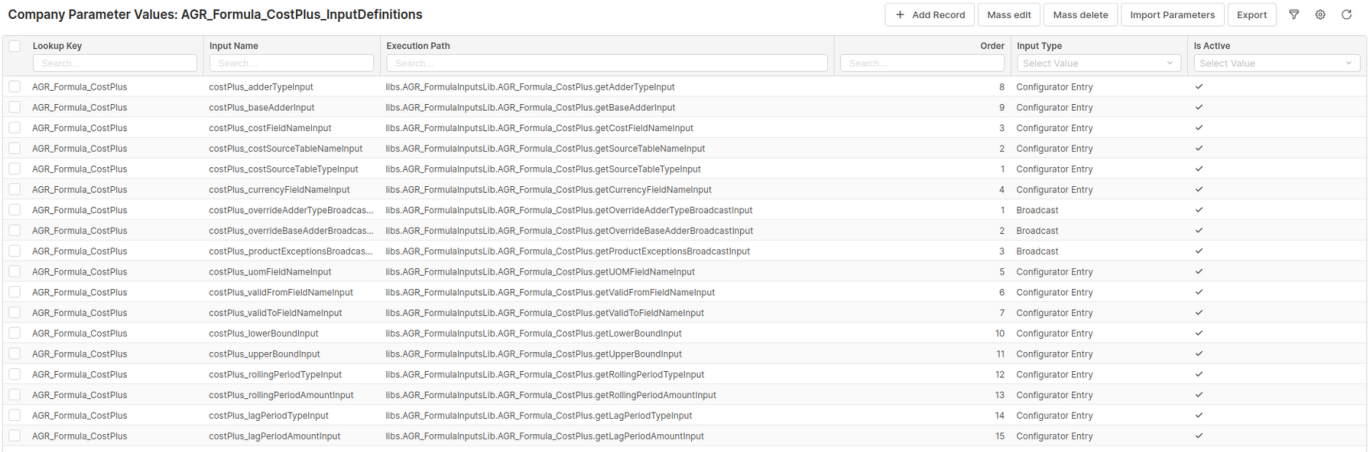
Lookup Key	Input Name	Execution Path	Order	Input Type	Is Active
AGR_FormulaBasedPricing	lineItem_productConfigurator	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricing.getProductConfigurator	1	Configurator	✓
AGR_FormulaBasedPricing	lineItem_formulaConfigurator	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricing.getFormulaConfigurator	2	Configurator	✓
AGR_FormulaBasedPricing_Formula...	formulaConfigurator_formulaDefinit...	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricing_FormulaConfigurator.getFormulaDefinitionInput	1	Configurator Entry	✓
AGR_FormulaBasedPricing_Product...	productConfigurator_productGrou...	libs.AGR_InputGeneratorLib.AGR_FormulaBasedPricing_ProductConfigurator.getProductGroupInput	1	Configurator Entry	✓

AGR_Formula_CostPlus_InputDefinitions

Allows you to customize the inputs in the Definition step: their order, visibility etc.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.

Details abut usage can be found in How to Modify, Add and Remove Inputs.



Company Parameter Values: AGR_Formula_CostPlus_InputDefinitions

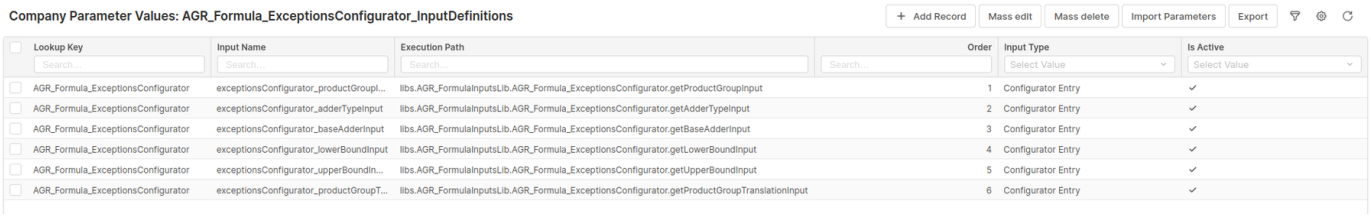
+ Add Record | Mass edit | Mass delete | Import Parameters | Export

Lookup Key	Input Name	Execution Path	Order	Input Type	Is Active
AGR_Formula_CostPlus	costPlus_adderTypeInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getAdderTypeInput	8	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_baseAdderInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getBaseAdderInput	9	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_costFieldNameInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getCostFieldNameInput	3	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_costSourceTableNameInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getSourceTableNameInput	2	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_costSourceTableTypeInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getSourceTableTypeInput	1	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_currencyFieldNameInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getCurrencyFieldNameInput	4	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_overrideAdderTypeBroadcas...	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getOverrideAdderTypeBroadcastInput	1	Broadcast	✓
AGR_Formula_CostPlus	costPlus_overrideBaseAdderBroadcas...	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getOverrideBaseAdderBroadcastInput	2	Broadcast	✓
AGR_Formula_CostPlus	costPlus_productExceptionsBroadcas...	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getProductExceptionsBroadcastInput	3	Broadcast	✓
AGR_Formula_CostPlus	costPlus_uomFieldNameInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getUOMFieldNameInput	5	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_validFromFieldNameInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getValidFromFieldNameInput	6	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_validToFieldNameInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getValidToFieldNameInput	7	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_lowerBoundInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getLowerBoundInput	10	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_upperBoundInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getUpperBoundInput	11	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_rollingPeriodTypeInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getRollingPeriodTypeInput	12	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_rollingPeriodAmountInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getRollingPeriodAmountInput	13	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_lagPeriodTypeInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getLagPeriodTypeInput	14	Configurator Entry	✓
AGR_Formula_CostPlus	costPlus_lagPeriodAmountInput	libs.AGR_FormulaInputsLib.AGR_Formula_CostPlus.getLagPeriodAmountInput	15	Configurator Entry	✓

AGR_Formula_ExceptionsConfigurator_InputDefinitions

Allows you to customize the exceptions configurator used by out of the box CostPlus Input Based formula. Details abut usage can be found in How to Modify, Add and Remove Inputs.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.

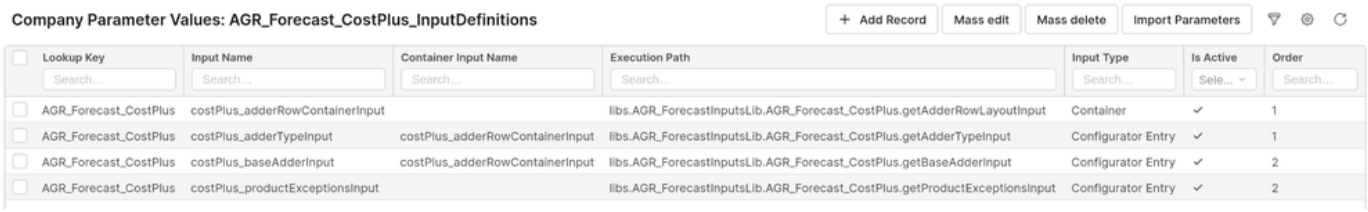


AGR_Forecast_CostPlus_InputDefinitions

Allows you to customize the inputs in the CostPlus forecast definition: their order, visibility etc. Usage of this parameter is configured in `AGR_ForecastTypes` Company Parameter.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.

Details abut usage can be found in How to Modify, Add and Remove Inputs.

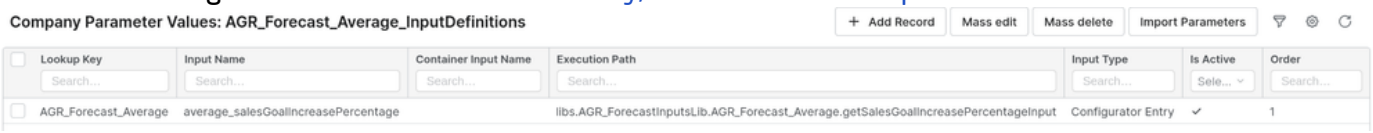


AGR_Forecast_Average_InputDefinitions

Allows you to customize the inputs in the out of the box Average forecast type definition: their order, visibility etc. Usage of this parameter is configured in `AGR_ForecastTypes` Company Parameter.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.

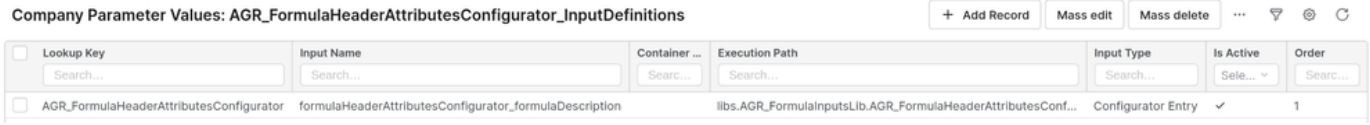
Details abut usage can be found in How to Modify, Add and Remove Inputs.



AGR_FormulaHeaderAttributesConfigurator_InputDefinitions

Allows you to customize inputs on the “Details” tab in Formula CFO object. Their order, visibility etc.

Details abut usage can be found in How to Modify, Add and Remove Inputs.

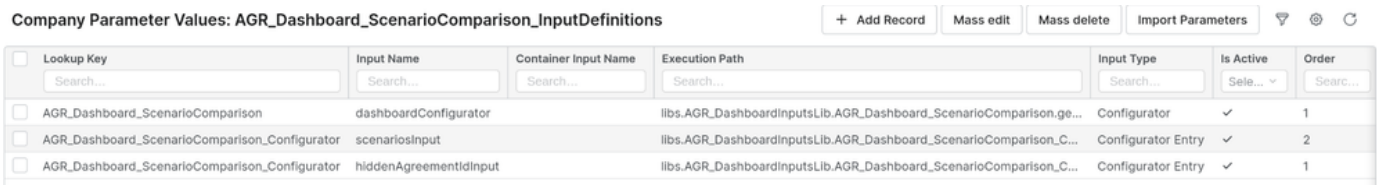


AGR_Dashboard_ScenarioComparison_InputDefinitions

Allows you to customize inputs on the “Scenario Comparison” tab in Agreement screen.

⚠️ Do not turn off individual inputs without checking the impact. Some of them are required for the Accelerator to work.

Details abut usage can be found in How to Modify, Add and Remove Inputs.



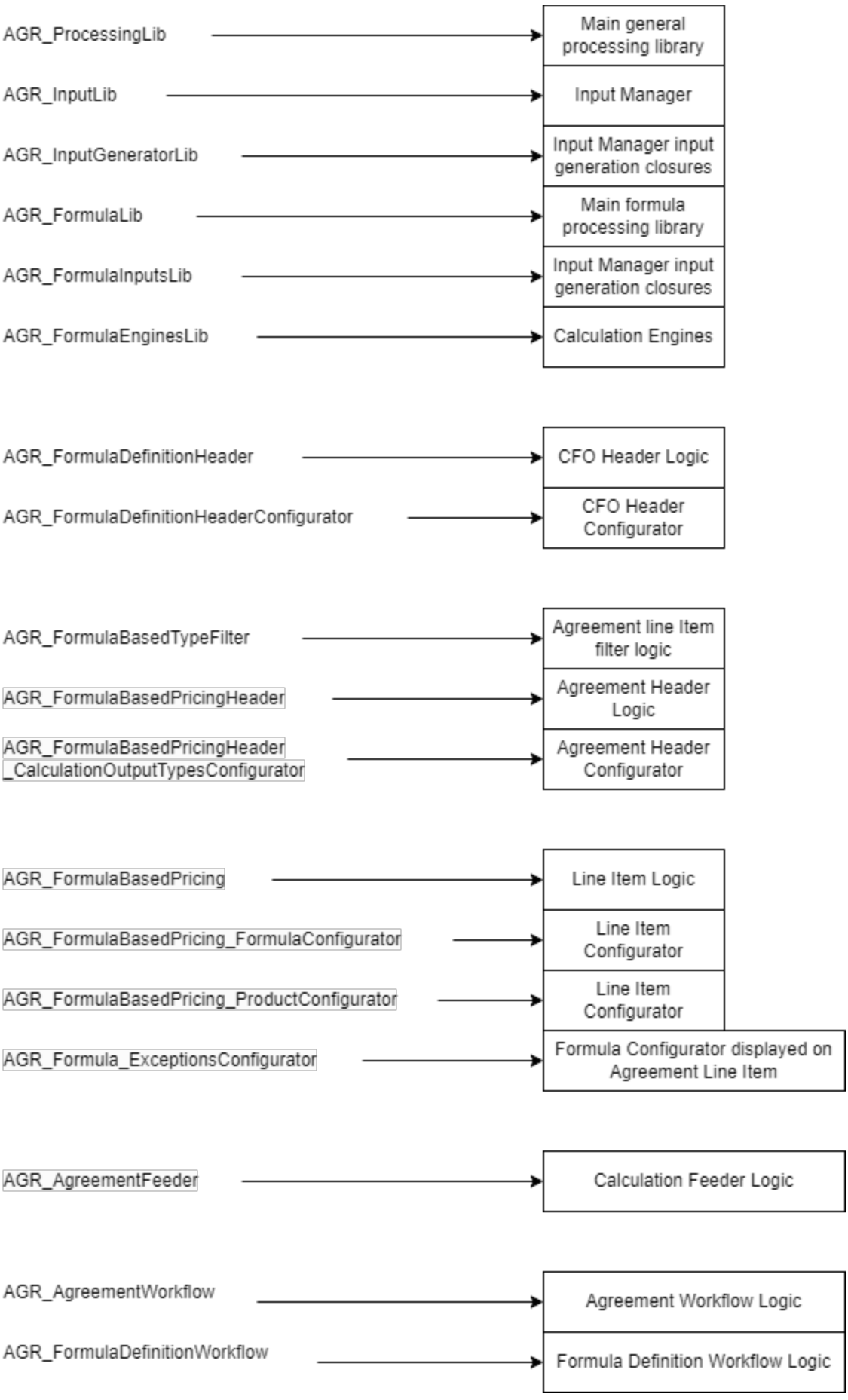
AGR_WarningConfig

Allows you to configure severity of various warnings and errors that the accelerator can throw.

Described in more detail in https://pricefx.atlassian.net/wiki/spaces/ACCDEV/pages/5224924835/Error+Management+Agreements#Configuration

Architecture Diagram (Agreements)

Will be overhauled after 1.0 release


AGR_ProcessingLib
Main general processing library
AGR_InputLib
Input Manager
AGR_InputGeneratorLib
Input Manager input generation closures
AGR_FormulaLib
Main formula processing library
AGR_FormulaInputsLib
Input Manager input generation closures
AGR_FormulaEnginesLib
Calculation Engines
AGR_FormulaDefinitionHeader
CFO Header Logic
AGR_FormulaDefinitionHeaderConfigurator
CFO Header Configurator
AGR_FormulaBasedTypeFilter
Agreement line Item filter logic
AGR_FormulaBasedPricingHeader
Agreement Header Logic
AGR_FormulaBasedPricingHeader _CalculationOutputTypesConfigurator
Agreement Header Configurator
AGR_FormulaBasedPricing
Line Item Logic
AGR_FormulaBasedPricing_FormulaConfigurator
Line Item Configurator
AGR_FormulaBasedPricing_ProductConfigurator
Line Item Configurator
AGR_Formula_ExceptionsConfigurator
Formula Configurator displayed on Agreement Line Item
AGR_AgreementFeeder
Calculation Feeder Logic
AGR_AgreementWorkflow
Agreement Workflow Logic
AGR_FormulaDefinitionWorkflow
Formula Definition Workflow Logic

Calculation Parameters (Agreements)

Calculation can have various Calculation Parameters (those are the parameters of the calculation method defined in Calculation Engine Path in the Company Parameter holding the Formula Type, be it Input-Based or Formula Designer).

There are currently few predefined Calculation Parameters:

- SKU – This is the SKU of the product that is being calculated by the Line Item.
- FORMULA_INPUTS – This is a Map containing all inputs present and filled in the Formula that is being used for calculation.
- FORECAST_DATA - This is a Map containing all Forecast data (produced by Forecast Engine) for configured periods. The Map is grouped by Period and then by Sku.
- FORECAST_PERIODS - This is Map containing all configured Forecast Periods produced by libs. SharedLib.DateUtils. It can be used to retrieve start/end dates and names of each period used.
- HISTORICAL_DATA - This is a Map containing all Historical data (queried from the configured Datamart) for configured periods. The Map is grouped by Period and then by Sku.
- HISTORICAL_PERIODS - This is Map containing all configured Historical Periods produced by libs. SharedLib.DateUtils. It can be used to retrieve start/end dates and names of each period used. This Map contains the same amount of periods as Forecast Periods.
- FORECAST_INPUTS - This Map contains all Forecast inputs that are defined by the used Forecast Engine (if used)

Other Calculation Parameters are input names defined in various places: Header, Scenario (these are the only ones not yet generated via InputManager solution, their names can be found under SCENARIO_INPUTS field in ConstConfig of AGR_ProcessingLib library), Line Item, Formula Type (both on the Formula and Broadcasted).

For example, you want to use one of the inputs present in Formula, you can use either the Input Name defined in the Formula Type Input Generation Table or use the predefined FORMULA_INPUT, which hold all inputs at once.

All Calculation Parameters should be entered into the Engine Calculation Parameters column of the appropriate Company Parameter. For Input-Based Formula Types it's AGR_InputBasedFormulaTypes Company Parameter, the Formula Designer Formula Types automatically populate Calculation Parameters and create appropriate tables. Calculation Parameters should be separated with commas without whitespaces, for example: SKU,FORMULA_INPUTS,myInput_TableName

Formula Designer (Agreements)

Strategy designer is a tool that allows Visual Configuration of Formula Types, it is tightly connected to the Agreements Accelerator architecture. It is based on Google Blockly.

The main idea is to be able to create simple Formula Types without the need of Groovy code.

Feature	Configures the following part of Agreements
Assign Calculation Parameters as parts of Calculation	Calculation Method Calculation Parameters
Define Calculation Formula using drag and drop feature	Calculation Method of a Formula Type

Assign inputs to various parts of Calculation using the customizable Input Blocks	Input Manager Input Generation Tables
Customizable Result Block	Custom Outputs to Condition Records (requires additional configuration at project start)

What Formula Designer is not:

- Replacement for complex Formula Types (those can be implemented using Input-Based Formulas)
- Visual Configuration for anything else than Formula Types for Agreements, there is no support for Quotes or any other modules.

Before the first Formula

If one is unfamiliar with Blockly it would be wise to take a moment and read the basic documentation about the solution here: Basics of Drag & Drop Visual Configuration

Installation

Formula Designer is embedded into the core release and is available from version 13.0.

Advanced Configuration

The Accelerator comes with some predefined configuration for Formula Designer, but it can be modified if necessary after deployment from Platform Manager.

In order to tweak the Formula Designer configuration navigate to Advanced Property called "formulaDesigner" under Administration Configuration Advanced Configuration Options. There will be a JSON file with properties as follows:

- groovyLibraryName - name of the library that will store the Activated Formula Types Calculation Methods Groovy Code.
- customBlocksGroovyLibraryName - name of the library that will contain definitions of Custom Blocks (functions) [link to article about custom blocks on Strategy Designer]
- customInputBlocksGroovyLibraryName - name of the library that will contain definitions of Input Blocks [link to Input Blocks article]
- deploymentLogicName - name of the logic that will run after the Formula Type has been "Activated", this logic creates all necessary Company Parameter tables and deploys the Calculation Methods
- excludedParameters - unsupported
- additionalInfoFields - defines additional connectors to the result block, this allows introduction of new fields that can be returned from the calculation method, for example to be saved on Condition Records
- additionalParameters - defines additional calculation parameters [link to article what they are] that can be passed to the calculation engines, remember that the calculation parameters need to be assigned to the engine first [link to article how to do so]

User Permissions

The Accelerator comes with predefined permissions that allow view/management of Formulas, ensure that all users that will utilize this feature have appropriate Business Roles assigned - [article about user permissions].

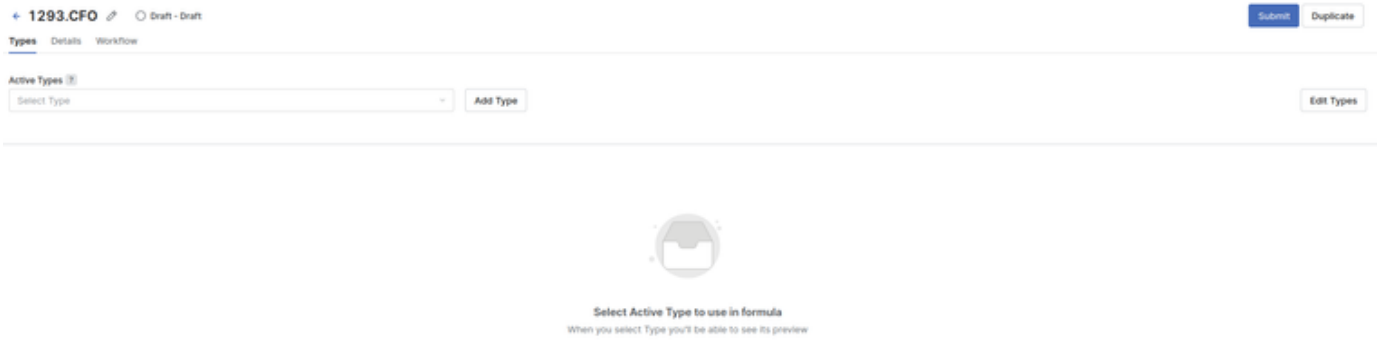
Creation of Formula Type and Formula

Step 1. Create Formula

In order to utilize the Formula Designer, one first needs to create a Formula. Navigate to the Side Menu Agreements & Promotions Formula and click Create New Formula in the top right section and fill in the details.

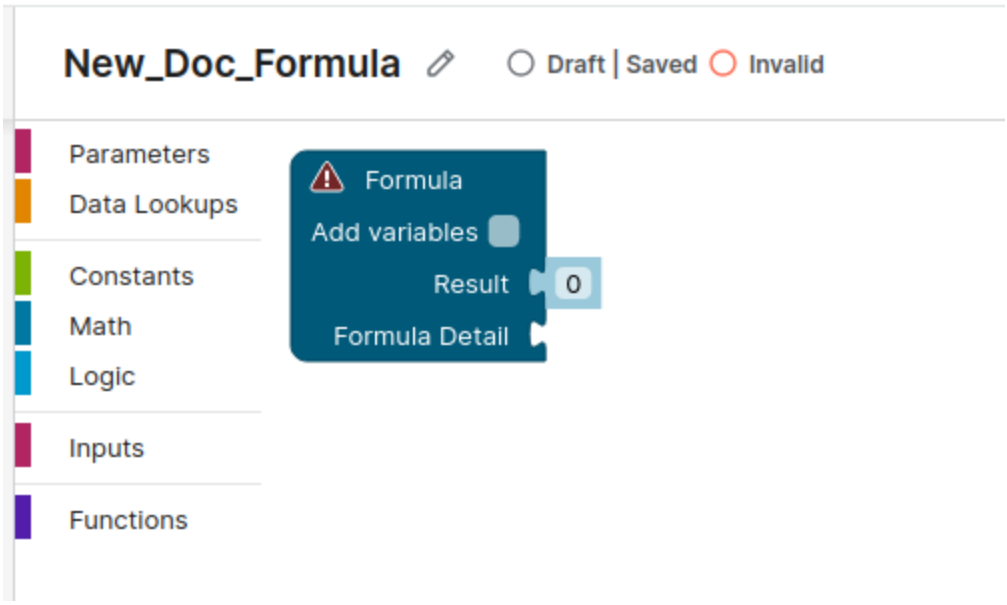
This will create a new Formula Custom Form that will represent the instance of specific use of (yet to be created) Formula Type. The Custom Form configuration can be found in Custom Form Types under Administration, it is named "AGR_Formula".

After creation of a Formula it can be opened, the screen will look like this:



The next step would be to navigate to "Add Type" or "Edit Types" where Formula Type can be created.

The main workspace will look like this:



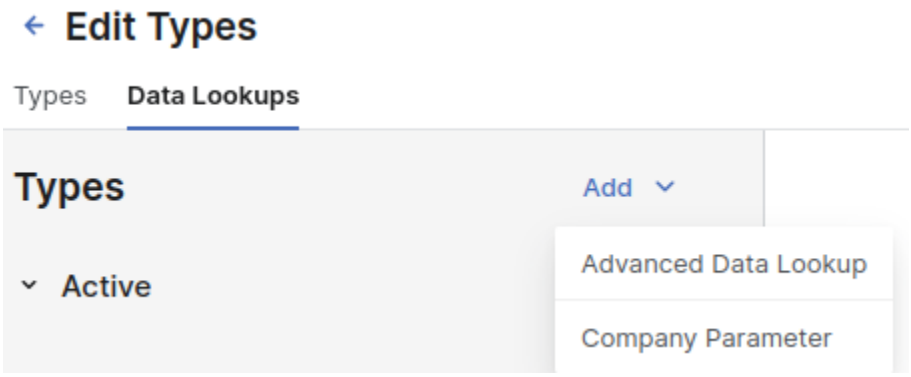
The ribbons define different Block sets, each set provides different functionallity.

- **Parameters** - those blocks allow the use of Calculation Parameters [article] to be used during the calculation
- **Data Lookups** - contains all of your configured Data Lookups.
- **Constants** - contains the following constants
  - **Text block** - allows to enter any static text value
- **Math** - contains blocks for mathematical operations, and all accept numerical inputs.
  - **Number** block
  - **Arithmetic** block
  - **Rounding** block
    (round = half-up, round up, round down)
  - **Constrain** block – Limits the first value by a minimum (low) and maximum (high), both optional.
  - **No price** block – Use this block as the final price of the formula in case you do not want it to return any value.
- **Logic** - contains blocks related to logical operations.
  - **If-Then-Else** block – If the value in the If input is true, then return the value in the Then input, otherwise return the value in the Else input.
  - **Evaluate** block (aka Switch-case block) allows you to evaluate a value plugged into the first input and then specify multiple If conditions to match the value being evaluated. The return value of this block is the return value of the first matched If block.
  - **Logical and/or** block used to group multiple conditions together.
  - **Comparison** block used to compare two values.
  - **Is (not) empty** block returns true if the given input is (not) empty (null or empty text).
  - **Boolean** block returns either true or false.
- **Inputs** - contains blocks that represent Inputs handled by Input Manager, those blocks will be displayed either on the Details tab or on the Agreement
- **Functions** - contains miscellaneous function blocks, that are configured manually in Groovy code.

Step 2. Create Data Lookup (optional)

Data lookups allow you to specify a data set that can be reused across your Formula Types in different ways.

Data lookup can be created by navigating to **Data Lookups tab** and clicking **Add** button above the list of lookups.



There are two types of lookups

- Company Parameter
- Advanced

Company Parameter Lookup

These two simple lookups can be compared to Excel's VLOOKUP function, simply fetching a single row based on a key or a set of keys. They always require all key fields to be specified and always produce just one value. They do not allow filtering, sorting, or aggregating data.

Advanced Data Lookup

The advanced data lookup works in a different way compared to the Company Parameter lookup. It splits the data fetching from the aggregation.

- The data fetching part is configured in the Data Lookups tab. An advanced data lookup does not produce a single value but always fetches a set of rows. You can filter and sort the result as you wish.
- The aggregation part happens in your Formula Type. Here you determine how to transform the rows to produce a single value.

The flexibility, however, comes with a price, and that is that the advanced data lookup is generally slower.

> **Do not use the advanced data lookup if you can achieve the same result with the simple one.** The advanced data lookup is generally slower than the simple one.

The advanced data lookup supports these data sources:

- Company Parameter
- Product Extension
- Product Master
- Customer Extension
- Customer Master
- Competition Data

Step 3. Use Formula Type in Formula

After Formula Type has been created and is valid (indicated by a green cirle with the word "Valid" near the Formula Name) it can be renamed, by clicking the Pencil icon and providing new name and activated by pressing the "Activate" button.

Only Active Formula Types can be used in Formula.

Once Active, the created Formula Type will show up in the drop down (if it doesn't then refresh the page) and can be selected. Its Blockly representation will show up in the Preview section below.

Step 4. Fill in the Details Tab

On the "Details" tab user is presented with any Inputs that were a part of the Formula Type. The values filled in here will be passed to the Calculation into appropriate places. If any Input has been marked as "Broadcasted" then it will show up on the Agreement instead (after the Formula has been approved)

Additionally in the Attributes section a Text Description of the Formula can be added to easily define its purpose.

Step 5. Submit and Approve the Formula

Once everything is covered the Formula CFO can be renamed and sent through approval workflow (if defined - using Approval Workflow Package [link to article]).

Once approved the Formula will become available on the Agreement.

Creation of Custom Functions

Custom Functions are created in the same way Strategy Designer does it: Create Custom Blocks in Groovy (Strategy Designer). Keep in mind that the Accelerator comes together with a library designed to be used for Custom Functions called "AGR_FormulaDesigner_CustomBlocks".

Creation of Custom Input Blocks

Create Groovy Library that will store the Custom Blocks. Accelerator comes with a library named "AGR_FormulaDesigner_CustomInputs" that contains some premade inputs.

For a new library simply follow these steps:

1. Create a Groovy Library and name it' (write down this name for later).
2. In the library, create the first element and name it Meta. Set its Display Mode to Everywhere. More on the Meta element later.
3. Create other elements (at least one) which will contain the functions your custom inputs will call.

The Meta element **must return an array of objects**, each object representing a block you want to create in Strategy Designer.

Each block has a label and can contain parameters (block inputs where other blocks can be plugged in). The block will then call a specified custom Groovy function located in another element of this library, passing the values of the user-provided parameters to it.

The block object has the following structure:

Property	Type	Required?	Description
`element`	String	Required	Location of the Groovy function which the block calls.
`function`	String	Required	Name of the function in the element specified by the `element` property, which the block calls.
`params`	List		List of parameter objects specifying the inputs of the block.

		Optional	Each parameter object has the following properties: • `name` (String) – Unique name of the parameter. • `type` (String) – Type of the parameter, either *number*, *string*, or *option*. • `default` – Default value supplied if the user does not provide the input; its type must correspond with the `type` property. • `options` – Specified only if the `type` is *option*. It is **either a list of values, or a map** where the key is the option's value and value is the option's label.
`label`	String	Required	Label rendered on the block. It can contain parameter placeholders like `{1}`, `{2}`, etc. If `params` are specified, each parameter will be rendered at the position of the corresponding placeholder, so the first parameter will be rendered at the `{1}` placeholder, the second at the `{2}` placeholder, and so on. If no placeholders are specified, the parameters will be all added at the end of the block.
`color`	Integer or String	Optional	Color of the block. It can be either an integer value (0 to 360) representing HSV hue value, or a #rrggbb String representing a web color. If no color is specified, the block will have the default color of the Functions toolbox category.
`tooltip`	String	Optional	Tooltip of the block which is displayed when the user hovers the pointer over the block. If empty, no tooltip will be displayed.
`meta`	Map	Optional	Meta block contains any additional information that user may want to pass to the Deployment logic. Whole Meta element is passed and can be read in unchanged form.
`inlineInputs`	boolean	Optional	Defines whether the label placeholders should be displayed inline with the label text or not.
`returns`	String, either **number** or **string**	Required	Use "string", not utilized for Input Blocks.

Example

As an example let's create a new BigDecimal input that will provide some Adder Value.

Step 1. Create new Meta entry

In the Meta element of AGR_FormulaDesigner_CustomBlocks, let's append the existing list with a new entry:

```
[
        "element"     : "InputGenerationFunctions", <--- this defines
the element where we'll put our function that will generate the input
        "function"    : "generateAdderValueInput", <--- this defines
the name of the function that will generate the input
        "label"       : "Adder Input Is Broadcasted {1}", <--- this
will be displayed on the Block in the Blockly workspace, the
Broadcasted flag is necessary so we leave it there
        "color"       : "#287F62", <--- this can be set to anything
valid
```

```
        "inlineInputs": true, <--- short input so this looks better if
set to true
        "meta"        : [ <--- here we'll pass some additional
information to the deployment logic so InputManager will know how to
handle our new Input
                "name"    : "adderInput", <--- name of the input, if
name is not passed then the Deployment logic will use "id" which is
much less readable when debugging the code
                "element" : "InputGenerationFunctions", <--- we pass
the same element as in the "element" tag
                "function": "generateAdderValueInput", <--- we pass the
same element as in the "function" tag
                "order"   : 0, <--- here we pass the order used by
InputManager, if the project know that certain inputs should be
displayed in certain order (always) then this can me modified here
                "defaultLabel": "Adder Input" <--- if the label of the
input is not passed then this will be used instead (that is our case,
because label is not present, may require change in getInputLabel to
notify the code that the label is missing by `return meta.params?.label
?: meta.defaultLabel` )
        ],
        "params"      : [
                ["name": "isBroadcast", "type": "option", "options":
isBroadcastValues]
        ],
        "returns"     : "string"
],
```

Step 2. Create Generation Function

Navigate to the Element pointed by the Meta and create a function with the same name as the function property in Meta. In our case this will be something like this:

```
ConfiguratorEntry generateAdderValueInput(Map inputs, Map meta) {
    String inputLabel = getInputLabel(meta)

    ContextParameter inputContextParameter = api.inputBuilderFactory().
createUserEntry(meta.inputId)
            .setLabel(inputLabel)
            .buildContextParameter()

    return libs.AGR_InputLib.CommonInputGenerationUtils.
buildConfiguratorEntry(inputContextParameter)
}
```

This function will generate a new "Adder Value" input block that will be possible for selection under Inputs tab. **Meta map is crucial** here, because it contains all meta from the previous step but also all parameters (with user filled values) and inputId, in case name is not provided. If everything is filled correctly then

Deployment logic will handle the rest of the process. Notice that the function **returns ConfiguratorEntry**, this has to be the case for all input blocks.

Formula Activation logic

Activation or Deployment, those two phrases can be used interchangeably when talking about what happens after "Activate" is pressed on the Formula Type. The deployment is set under deploymentLogicName in the Advanced Properties of the Formula Designer (set to AGR_FormulaDesinger_Deployment by default). This logic is run after Activate is pressed and in Accelerators case it does a number of things.

This logic has access to a special variable called "input", which contains information about the Formula Type, such as:

- input.formulaLibraryName - name of the library from Advanced Configuration
- input.formulaName - name of the Formula Type from the Formula Designer
- input.formulaParameters - names of the used parameter blocks
- input.inputs - list of used input blocks

Populate Inputs

If there are any inputs used by the Formula Type, then their respective Input Generation tables need to be created. This process follows the same principle as regular inputs defined here [article to How to add inputs].

1. First the Input Generation table is created by following the pattern defined by: libs. AGR_ProcessingLib.ConstConfig.FORMULA_DESIGNER_CONFIG.INPUT_GENERATION. PRICE_PARAMETER.NAME_PATTERN, which by default is AGR_Your*FormulaName*_InputDefinitions
2. Next the Input Generation table is filled based on the meta of the Input block
   a. Important step is that meta block is saved as one of the attributes of the table, which means that meta can be accessed by the generation function of the input, which can be handy.
   b. Meta is also extended by two additional entries: inputId and params (which are user filled values)

Populate Formula Types

After inputs have been generated the last step is to create the entry in FormulaTypes table pointed by libs.AGR_FormulaLib.ConstConfig.FORMULA_TYPES_PARAMETER_CONFIG, this step fills data such as formula name, calculation method path or parameters (taken from the used blocks)

How To Articles (Agreements)

- How to Create New Input-Based Formula
- How to Add New Input-Based Formula Type with Calculation Method and Inputs
- How to Add New Outputs to Line Item
- How to Modify, Add and Remove Inputs
- How to Modify What Is Saved in Condition Records
- How to Configure a new Forecast Engine

How to Create New Input-Based Formula

To create a new Formula Definition to be used in Agreement Line Item calculations:

1. Go to **Agreements & Promotions** > **Input-Based Formula**.
2. Click the **Create New Input-Based Formula** button on the top right, provide a label (to identify formula) and other fields if necessary.
3. After navigating into the Formula you get the option to select the Formula Type. Those are defined in the **AGR_InputBasedFormulaTypes** Company Parameter and described in detail How to Add New Formula Type with Calculation Method and Inputs.
4. After selecting the Formula Type the appropriate set of inputs (if defined in the Formula Type) will be shown. You can fill them with necessary data and then submit the Formula Definition. Before the Formula Definition can be used on an Agreement Line Item it needs to be approved.

How to Add New Input-Based Formula Type with Calculation Method and Inputs

To add a new Input-Based Formula Type with a calculation method and inputs, follow these steps:

- Step 1 - Define Input-Based Formula Type
- Step 2 - Define Inputs
- Step 3 - Define Calculation method
  - Exemple configuration
- Tips & Tricks

Step 1 - Define Input-Based Formula Type

Navigate to the AGR_InputBasedFormulaTypes Company Parameter and fill in the columns:

- **Formula Type Name** - Provide a name of the Formula Type that will be displayed in the dropdown in Input Based Formula. It will also serve as an identifier for the Formula Type, therefore it must be unique.
- **Calculation Engine Path** - Provide a path to a calculation method in the format `libs.<name of lib>.<name of element>.<name of method>`. Keep in mind that there is no parentheses at the end of the path.
- **Engine Calculation Parameters** - Provide any Calculation Parameters that you may need, which parameters are available and what they provide can be found here [link to Calculation Parameters] article
- (Optional) **Input Generation Table Name** - If the Formula Type requires some user inputs to be generated by InputManager, provide a name of another Company Parameter table that will store their definitions. Usually it's a good practice to name the InputGeneration table using the name of the formula type with "InputDefinitions" suffix for ease of identification. For example if Formula Type is named "AGR_MyFormulaType" then the InputGeneration table name would be "AGR_MyFormulaType_InputDefinitions".

Step 2 - Define Inputs

Inputs are managed by the InputManager solution available in AGR_InputLib library and described in How to Modify, Add and Remove Inputs.

All inputs used by FormulaTypes must be defined as "Configurator Entry" (if they are to be displayed in Formula Definition) or as Broadcast (if they are to be displayed in Agreement Line Item).

Keep in mind that the Input Name provided in the InputGeneration table can be used as a calculation parameter that is passed to the Calculation method defined in Step 3.

For example if the Input is named "header_myHeaderValue" it can be passed into Calculation Parameters SKU,FORMULA_INPUTS,header_myHeaderValue, which in Calculation method would look like:

```
Map calculate(String sku,
              Map formulaInputs,
              String header_myHeaderValue) {
      <Calculation Body Here>
}
```

Step 3 - Define Calculation method

Calculations are handled by the EngineCalculator solution that is present in AGR_FormulaLib library.

The calculation method has to follow a certain signature. First the name of the method has to match the name (and place) that the **Calculation Engine Path** in AGR_InputBasedFormulaTypes Company Parameter points to. Second it has to have the **Engine Calculation Parameters** defined in the same order as in AGR_InputBasedFormulaTypes Company Parameter and of an appropriate type.

More on types and available Calculation Parameters can be found here [link to Calculation Parameters] article.

EngineCalculator requires a certain structure to be returned from your calculation method; the structure can be looked up in the `createEngineResult` method in EngineCalculator and it is as follows:

```
[result        : result,
 message        : message,
 messageType    : messageType,
 additionalInfo : additionalInfo]
```

where:

- `result` key stores the result value (final price) of your Calculation Engine. This value is stored in Condition Records in the Engine Result column.
- `message` key can store any message in String format that you wish to return from the engine, it can be a calculation status or something else. EngineCalculator automatically fills this field if it is not overridden.
- `messageType` can store the type of the message provided in the `message` key. EngineCalculator automatically fills this field if it is not overridden. The available values are stored in `libs.AGR_FormulaLib.EngineCalculator.MESSAGE_TYPES`.
- `additionalInfo` can store anything else that you wish to return from the calculation. Our CostPlus utilizes this to return a Map with information such as calculatedResult, formulaDetail, baseAdderType, baseAdderSource all of which are stored in their appropriate columns in Condition Records.

Keep in mind that you may need to adjust the Condition Records structure to match your calculation and its results.

Exemple configuration

- AGR_InputBasedFormulaTypes
  - Formula Type Name - AGR_DemoFormula
  - Calculation Engine Path - libs.AGR_InputBasedFormulaEnginesLib.AGR_DemoFormula.calculate
  - Engine Calculation Parameters - FORMULA_INPUTS,SKU,header_myHeaderValue
  - Input Generation Table Name - AGR_DemoFormula_InputGeneration

- AGR_DemoForecast_InputGeneration
  - Lookup Key - AGR_DemoFormula
  - Input Name - demo_AdderInput
  - Execution Path - libs.AGR_InputGeneratorLib.AGR_DemoFormula.getAdderInput
  - Order - 0
  - Input Type - Configurator Entry
  - Is Active - true

The calculation method in pseudo code could look something like this:

```
Map calculate(Map formulaInputs,
              String sku,
              BigDecimal header_myHeaderValue) {
    BigDecimal adderValue = getAdderValue(forecastInputs)
    BigDecimal productPrice = getProductPrice(sku)

    return [result        : productPrice + adderValue +
header_myHeaderValue,
            additionalInfo: [productPrice: productPrice,
                             adderValue  : adderValue]]
}
```

The input generation method could look something like this:

```
ConfiguratorEntry getAdderInput(Map inputs) {
    ContextParameter adderParameter = api.inputBuilderFactory()
            .createUserEntry("demo_AdderInput") //Remember to use the
same name as in Input Generation Table Config
            .setLabel("Adder Input")
            .setRequired(true)
            .buildContextParameter()

    return libs.AGR_InputLib.CommonInputGenerationUtils.
buildConfiguratorEntry(adderParameter)
}
```

The following example should generate a Formula that will take some Product Price and add some static Adder value to it as well as the value from Header.

Tips & Tricks

- If you throw an exception in the Calculation Method it will be caught and displayed nicely in the Calculation Outputs Result Matrix.

How to Add New Outputs to Line Item

The Agreement Accelerator follows the usual rules when it comes to output generation - that is that each output requires its own element to be present.

To create a new output, create a new element and put it after the VolumeChangeOutput element and before OccuredWarningsOutput in the AGR_FormulaBasedPricing line item logic.

How to Modify, Add and Remove Inputs

Most inputs present in the package are handled by InputManager available in *AGR_InputLib* library.

InputManager is a solution that enables dynamic management of inputs, reordering, containers, turning on and off (if not required by underlying code) and other functions.

In order to add an input there's several steps that have to be done:

Step 1. Define the Logic that will use the InputManager

InputManager can be used in various places: Header, Line Item logics, Formula Types or other places. The most important part defining whether the InputManager can be used is if the inputs that will be defined are supported by the solution, so as long as the input you want to define is one of the ones listed below, then it should be fine:

- Void - usually used when there's a behaviour that InputManager needs to do, without generating any input. This can be used for example to hide system Header inputs.
- Basic Input - used for inputs generated using `.getInput()` call from InputBuilderFactory.
- Configurator - used for generation of Configurators using `api.inlineConfigurator` or `api.configurator`.
- ConfiguratorEntry - used for generation of Configurator Entries inside Configurator logics generated using `api.createConfiguratorEntry`.
- Broadcast - special type of Input used only for FormulaType InputDefinitions, makes it so the input will show on Agreement Line Item instead of the Inputs tab of a Formula. This input is a Configurator Entry.
- Container - special type of Input used to group other inputs into sections, for example rows or collapsible sections. The Container has to contain other Configurator Entries and be generated using for example using `createRowLayout`.

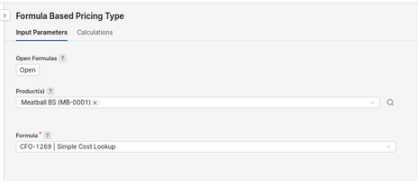
The logics that use InputManager currently are:

AGR_FormulaBasedPricingHeader

(Header)

AGR_FormulaBasedPricing

(Line Item)

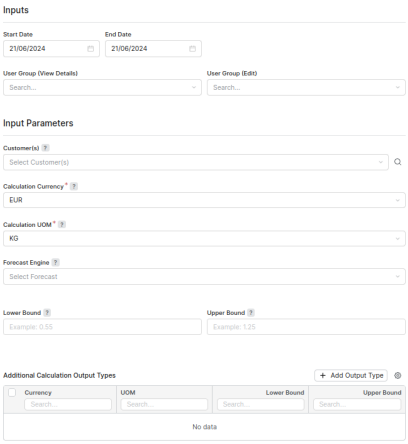


Inputs displayed below Formula depend on defined Broadcastable inputs

AGR_FormulaBasedPricing_FormulaConfigurator

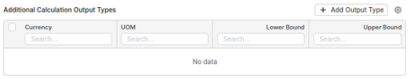
(Line Item)

This Configurator handles the display of Formula Type specific inputs.



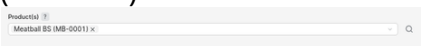
AGR_FormulaBasedPricingHeader_AdditionalCalculationOutputTypesConfigurator

(Header)



AGR_FormulaBasedPricing_ProductConfigurator

(Line Item)



AGR_FormulaBasedPricingHeader_ForecastConfigurator

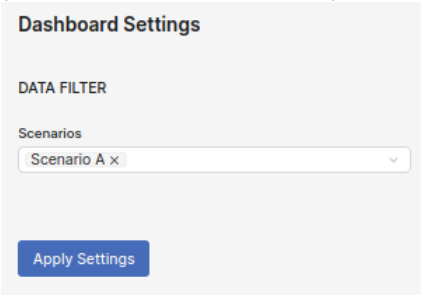
(Header + Scenario)

AGR_FormulaBasedPricingHeader_BoundsConfigurator

(Header)

AGR_Dashboard_ScenarioComparison

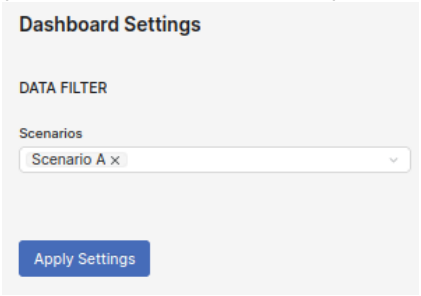
(Scenario Comparison Tab)



Handles the display of AGR_Dashboard_ScenarioComparison_Configurator

AGR_Dashboard_ScenarioComparison_Configurator

(Scenario Comparison Tab)



More on each logic can be found in [link to Logic Docs]

Step 2. Find appropriate Company Parameter to fill the Input Generation configuration

There are several predefined Company Parameters that hold the configuration of Input Manager inputs. Those Company Parameters are divided based on the logics that generate the inputs, therefore which one should be modified is strictly connected to the choice from Step 1.

The predefined Input Generation tables are as follows:

- AGR_FormulaBasedPricingHeader_InputDefinitions - defines the inputs generated on the Header of Agreement
- AGR_FormulaBasedPricing_InputDefinitions - defines the inputs generated on the Line Item (additionally Broadcast inputs are generated there as well, but they are not a part of this table)

- AGR_Dashboard_ScenarioComparison_InputDefinitions - defines the inputs shown on the Scenario Comparison Dashboard [link to Scenario Comparison Dashboard] article
- AGR_Formula_CostPlus_InputDefinitions - defines the inputs present on the Input-Based Formula when AGR_Formula_CostPlus is selected as the used Formula Type
- AGR_Forecast_CostPlus_InputDefinitions - defines the inputs present on the Header when AGR_Forecast_CostPlus is selected as a Forecast Engine
- AGR_Forecast_Average_InputDefinitions - defines the inputs present on the Header when AGR_Forecast_Average is selected as a Forecast Engine
- AGR_Formula_ExceptionsConfigurator_InputDefinitions - defines the inputs present in the Product Exceptions table that is part of the Cost Plus Formula Type and Cost Plus Forecast Engine
- AGR_FormulaHeaderAttributesConfigurator_InputDefinitions - defines the inputs present on Attributes section of a Formula Designer Formula tab called "Inputs"

Inputs are defined using Company Parameter tables with defined and set structure, which is as follows:

- Table Type: JSON
- Value Type: JSON2
- Columns:
  - key1 - Lookup Key
    - This is a key defining the "set" of inputs that will be used during generation.
  - key2 - Input Name
    - Name of the input. This name can also be used as a calculation parameter for the Calculation Engine. This name must match the name that the input is created with in the code.
  - attributeExtension___executionPath - Execution Path
    - Generation Path for the input. Provided in format `libs.<name of lib>.<name of element>.<name of method>`. The generation path return value must match the Input Type of the input, defined below.
  - attributeExtension___order - Order
    - Integer indicating the order of this input compared to others. Inputs are ordered by descending order.
  - attributeExtension___inputType - Input Type
    - Type of the input generated by the Execution Path, this is used to handle different contexts, for example Configurators or Broadcasted inputs.
      - Void - the Execution Path returns void, useful for example for hiding system inputs or other cases where the value is not needed.
      - Basic Input - ExecutionPath returns the input generated using `.getInput()` call from InputBuilderFactory.
      - Configurator - ExecutionPath returns the Configurator input Map using `api.inlineConfigurator` or `api.configurator`.
      - ConfiguratorEntry - ExecutionPath returns the ConfiguratorEntry generated using `api.createConfiguratorEntry`.
      - Broadcast - ExecutionPath returns the ConfiguratorEntry generated using `api.createConfiguratorEntry`.
      - Container - ExecutionPath returns the ConfiguratorEntry generated using `api.createConfiguratorEntry` that contains other Configurator Entries
  - attributeExtension___isActive - Is Active
    - Boolean flag indicating whether the input is shown or not. Please be careful when turning off inputs that may be expected to be present by the underlying code.
  - attributeExtension___containerInputName - Container Input Name

- Defines the name of a Container Input that will hold currently defined Configurator Entry. This mechanism is used to group inputs into rows or collapsible sections.

Step 3. Define the Input Generation method

The Input Generation method has to follow a certain signature. First the name of the method has to match the name (and place) that the ExecutionPath in Company Parameter defined in Step 2. points to. Second it has to have a parameter called `inputs` of type Map - this allows access to other inputs generated previously (for example to create dependencies between Configurator Entries).

Example Input Generation Method for a Decimal Entry could look like this:

```
BigDecimal getDecimalEntryInput(Map inputs) {
    return api.inputBuilderFactory()
            .createUserEntry("decimalEntry")
            .setLabel("Decimal Entry Input")
            .setRequired(true)
            .getInput()
}
```

The Execution Path to it could be for example `libs.MyInputs.MyConfiguratorEntries.getDecimalEntryInput`

Step 4. Generate the Inputs

After the inputs have been defined, they can be generated in bulk by calling an appropriate method on an instance of the InputManager from Step 1.:

- `getInputs` - Generates all inputs defined as Void, Basic Input, Configurator. Used for example for line item logics, headers, dashboards etc.
- `getConfiguratorEntries` - Generates all inputs defined as ConfiguratorEntry or Container. Used for Configurator logics.
- `getBroadcastConfiguratorEntries` - Generates all inputs defined as Broadcast. Used for Broadcast inputs. This mechanism is already handled by the Accelerator Configurator build for this purpose: *AGR_FormulaBasedPricing_FormulaConfigurator*.

This call returns the values of the inputs. In predefined logics there usually is an element called InputGeneration, which calls one of such methods. It also is used to access any values returned from those inputs in the logic.

For example to access the value of the input generated in Step 3. one would call `out.InputGeneration.getAt("decimalEntry")` to retrieve the value entered by the user.

Remember that all inputs generated are automatically appended as Calculation Parameters [link to article] that can be used when defining Input-Based Formulas [link to article].

Step 5. Validate proper generation of inputs

Last step is to navigate to the place where logic from Step 1. is run and validate whether the inputs have been properly generated.

(Optional) Add your own Input Manager logic

If you want to create a new logic that will use Input Manager, then InputManager needs to be initialized in it by calling: `libs.AGR_InputLib.InputManager.getInstance(<CompanyParameterName>, <LookupName>)`

where:

- `CompanyParameterName` is a name of the input generation table defined in Step 2.
- `LookupName` is the value of key1 column of that table that defines the set of inputs used by this logic - one Company Parameter can store multiple "sets", this solution allows to store inputs that belong to similar "context" in one Company Parameter - for example Dashboard logic that calls a Configurator and Configurator Entries of that Configurator logic.

How to Modify What Is Saved in Condition Records

The current implementation has a certain Condition Records structure to fit the Cost Plus use case implemented in the Accelerator. The structure can be freely changed to meet the project requirements.

To modify what is saved in Condition Records:

- Step 1 – Modify Condition Records Set
- Step 2 – Modify ConditionRecords Element

Step 1 – Modify Condition Records Set

Go to **Administration > Configuration > Condition Record Sets (under Master Data)** > **AGR_DefaultCondition_Record_Set** and modify any columns/attributes to fit your calculation outputs.

You can also create new Condition Record set instead, but remember to change the CONDITION_RECORD_SET_NAME value to the name of the new set in the library AGR_ProcessingLib element ConstConfig. The new set has to have the same keys as the AGR_DefaultCondition_Record_Set.

Step 2 – Modify ConditionRecords Element

Go to the AGR_FormulaBasedPricing logic and open the ConditionRecords element.

There are several variables that hold important information, those are:

- calculationParameters – Stores all possible parameters to use by Calculation Engine CalculationParameters.
- processedCalculationOutputs – Stores all calculation outputs in Map format. The Map's main key is a combination of Currency and UOM used for that run of calculation (based on selected CalculationOutputTypes) and the content is :

```
[outputCurrency     : outputCurrency,
 outputUom          : outputUom,
 exchangeRate       : exchangeRate,
 conversionFactor   : conversionFactor,
 calculationResults : [:]]
```

where:

- `outputCurrency` – Currency of the run as defined by CalculationOutputTypes.
- `outputUom` – Unit of measure of the run as defined by CalculationOutputTypes.

- `exchangeRate` - Exchange rate used to convert from calculation currency to the current outputCurrency.
- `conversionFactor` - Conversion factor used to convert from calculation unit of measure to the current outputUom.
- `calculationResults` - Extended outputs of the EngineCalculator in Map format, grouped by SKU. This contains the map returned by the Calculation method.
  The format of the Maps stored under calculation Results looks as follows

```
[(sku): [engineResult     : engineResult,
         boundStatus      : boundStatus,
         previousResult   : previousResult,
         calculationStatus: calculationStatus,
         convertedResult  : convertedResult]]
```

  - `engineResult` - this is the Map that is returned from the Calculation method
  - `boundStatus` - if Lower/Upper Bounds are applied, this property contains whether Upper or Lower bound has been applied. Has value only during Recalculation.
  - `previousResult` - if Lower/Upper Bounds are applied, this property contains the result from the Calculation method of the Previous Period.
  - `calculationStatus` - defines whether the calculation finished with or without error. Values stored in CALCULATION_STATUS Map in AGR_ProcessingLib/ConstConfig.
  - `convertedResult` - contains the result from the Calculation method that has been converted using UoM and Currency conversions.

To change the structure, you must modify the Map variable called conditionRecordStructure, with the changes the reflect new/modified attributes from Step 1 - that is assign proper values to proper attribute keys.

How to Configure a new Forecast Engine

The setup of Forecast engine is almost identical to the Input Based Formulas which are defined here: [link to "How to Add New Formula Type with Calculation Method and Inputs"]

There are few steps required to configure the Forecast Engine:

1. **Company Parameter Configuration** - where the user defines the name of the forecast and it's calculation configuration
2. **Inputs** - where the user creates a InputGeneration table, fills it with data and connects it to the Forecast Company Parameter
3. **Groovy logic** - where the user writes the calculation method code and any additional support methods, such as input generation code.

The setup is done fully in UI and Groovy, there is **no Formula Designer support** for Forecast Engines.

Company Parameter Configuration

The configuration starts with Company Parameter named "**AGR_ForecastTypes**", this table contains several columns that need to be filled in:

- Forecast Type Name - defines the name that will be used for this Forecast Engine, it will for example be displayed on the Agreement Header input that allows Forecast Engine selection.

- Calculation Engine Path - defines the path to the library that will contain the calculation method of the Forecast Engine. The Accelerator comes with a library, which can be used for this purpose, it's called "**AGR_ForecastEnginesLib**". The path should point to the main calculation method that will return the valid result structure (more on that in the Groovy Logic section). For example: libs.AGR_ForecastEnginesLib.MyTestingForecast.calculate, notice there is no parentheses at the end of the path.
- Engine Calculation Parameters - defines calculation parameters that the calculation method will use. All parameters that can be used and how they should be used are defined in [link to article about Calculation Parameters]
- Input Generation Table Name - defines the name of the InputManager inputs definition table.

Inputs

All inputs used by the Forecast Engine should be defined as Configurator Entries (**The "Input Type" column in the InputGeneration table should be set as "Configurator Entry"**). More on Inputs can be found in [link to How to Modify, Add and Remove Inputs].

Groovy logic

For Groovy logic the only necessity is the definition of calculation method that matches the path defined in the AGR_ForecastTypes Company Parameter.

The calculation method must take the parameters defined in the order pointed by Engine Calculation Parameters column.

There are additional parameters that can be returned by the Forecast Engine under the "additionalInfo" key. Those are "productCost" and "productVolume", if they will be present in the returned data set, then the respective "Change" output will be calculated. For example, if productVolume is present then the Volume Change output will be calculated.

Example calculation method return structure:

```
[result       : result,
additionalInfo: [productCost  : productCost,
                 productVolume: productVolume]]
```

Exemple configuration

- AGR_ForecastTypes
  - Forecast Type Name - AGR_DemoForecast
  - Calculation Engine Path - libs.AGR_ForecastEnginesLib.AGR_DemoForecast.calculate
  - Engine Calculation Parameters - FORECAST_INPUTS,SKU
  - Input Generation Table Name - AGR_DemoForecast_InputGeneration
- AGR_DemoForecast_InputGeneration
  - Lookup Key - AGR_DemoForecast
  - Input Name - demo_AdderInput
  - Execution Path - libs.AGR_ForecastInputsLib.AGR_DemoForecast.getAdderInput
  - Order - 0
  - Input Type - Configurator Entry
  - Is Active - true

The calculation method in pseudo code could look something like this:

```
Map calculate(Map forecastInputs,
              String sku) {
    BigDecimal adderValue = getAdderValue(forecastInputs)
    BigDecimal productPrice = getProductPrice(sku)

    return [result        : productPrice + adderValue,
            additionalInfo: [productPrice: productPrice,
                             adderValue  : adderValue]]
}
```

The input generation method could look something like this:

```
ConfiguratorEntry getAdderInput(Map inputs) {
    ContextParameter adderParameter = api.inputBuilderFactory()
            .createUserEntry("demo_AdderInput") //Remember to use the
same name as in Input Generation Table Config
            .setLabel("Adder Input")
            .setRequired(true)
            .buildContextParameter()

    return libs.AGR_InputLib.CommonInputGenerationUtils.
buildConfiguratorEntry(adderParameter)
}
```

The following example should generate a Forecast that will take some Product Price and add some static Adder value to it.

Error Management (Agreements)

Agreement Accelerator comes with build-n Warning Manager, which takes care of proper calculation flow. In case an error occurs, Warning Manager intercepts it and handles it in a warning specific way. All caught warnings (if configured to do so) are then displayed in a matrix.

Technical Information

Warning Manager overrides standard behavior in case any exception is thrown. Instead of showing an alert to the user with technical message, it captures the exception and selects the corresponding message from configuration (see Configuration section).

To achieve this, Warning Manager has a method called *tryToExecuteElement,* which takes code we want to run as a closure parameter and then runs it in a *try/catch* block. Sample method call:

```
return out.WarningManager.tryToExecuteElement("FormulaData") {
    Map generatedInputs = out.InputsGeneration

    String formulaCfoUniqueName = libs.AGR_ProcessingLib.InputUtils.
```

```
getFormulaCfo(generatedInputs)

    if (!formulaCfoUniqueName) {
        api.throwException("MISSING_FORMULA_DATA")
    }

    String formulaObjectTypeName = libs.AGR_FormulaLib.
FormulaProcessingUtils.getFormulasObjectTypeName(formulaCfoUniqueName)
    String formulaTypeName = libs.AGR_FormulaLib.FormulaProcessingUtils.
getFormulaTypeName(formulaCfoUniqueName, formulaObjectTypeName)
    Map formulaType = libs.AGR_FormulaLib.FormulaProcessingUtils.
findFormulaType(formulaTypeName, formulaObjectTypeName)
    String engineCalculatorParameterName = libs.AGR_FormulaLib.
FormulaProcessingUtils.getEngineParameterName(formulaObjectTypeName)
    Map formulaInputValues = libs.AGR_FormulaLib.FormulaProcessingUtils.
getFormulaInputValues(formulaCfoUniqueName, formulaType,
formulaObjectTypeName)

    return [formula                     : formulaType,
            formulaTypeName             : formulaTypeName,
            engineCalculatorParameterName:
engineCalculatorParameterName,
            formulaInputValues          : formulaInputValues]
}
```

tryToExecuteElement method code:

```
tryToExecuteElement        : { String elementName, def fallbackOnError
= null, Closure elementCode ->
    try {
        return elementCode()
    } catch (any) {
        manager.handleThrownException(elementName, any)
    }

    return fallbackOnError
},
```

As you can see, the exception thrown in line 7 (first code snippet), will be caught in Warning Manager's internal *try/catch* block.

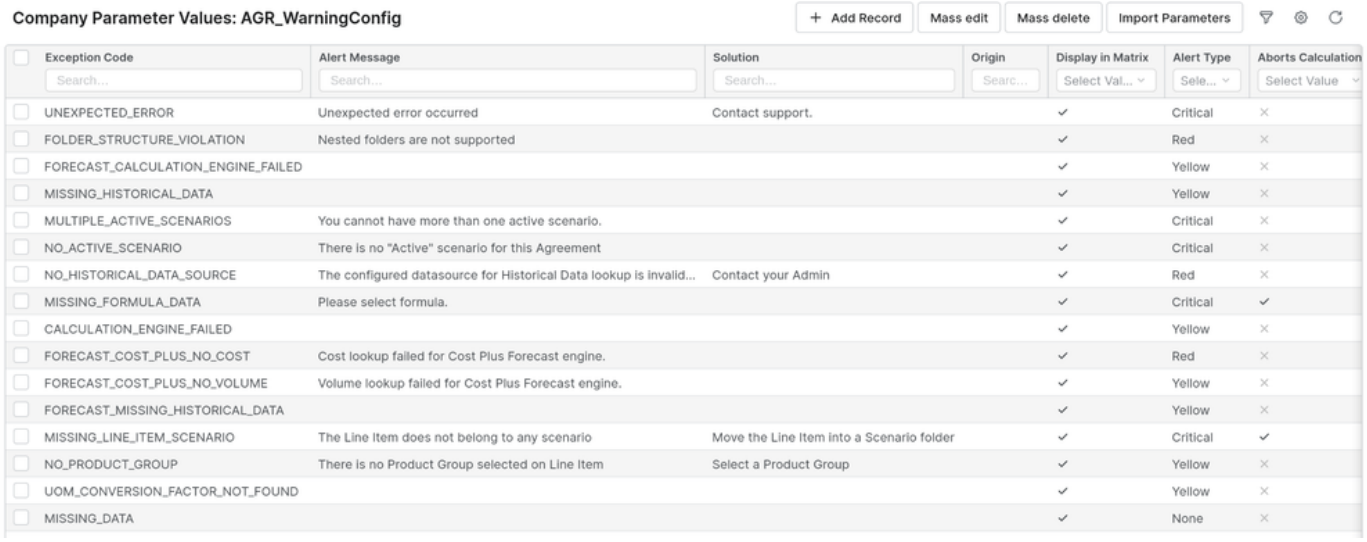
To determine error-specific behavior, we use uniform exception message format: **EXCEPTION::ALERT_MESSAGE.** EXCEPTION_CODE serves as a lookup key in Warning Manager configuration, ::ALERT_MESSAGE is optional, mostly used when you want to include a variable value in ALERT_MESSAGE string.

Alert Types

Warning Manager uses standard Unity alerts to notify the user. You can read about them in detail here.

Configuration

To set up Warning Manager, it is enough to configure one price parameter called AGR_WarningManager.



Exception Code	Alert Message	Solution	Origin	Display in Matrix	Alert Type	Aborts Calculation
UNEXPECTED_ERROR	Unexpected error occurred	Contact support.		✓	Critical	×
FOLDER_STRUCTURE_VIOLATION	Nested folders are not supported			✓	Red	×
FORECAST_CALCULATION_ENGINE_FAILED				✓	Yellow	×
MISSING_HISTORICAL_DATA				✓	Yellow	×
MULTIPLE_ACTIVE_SCENARIOS	You cannot have more than one active scenario.			✓	Critical	×
NO_ACTIVE_SCENARIO	There is no "Active" scenario for this Agreement			✓	Critical	×
NO_HISTORICAL_DATA_SOURCE	The configured datasource for Historical Data lookup is invalid...	Contact your Admin		✓	Red	×
MISSING_FORMULA_DATA	Please select formula.			✓	Critical	✓
CALCULATION_ENGINE_FAILED				✓	Yellow	×
FORECAST_COST_PLUS_NO_COST	Cost lookup failed for Cost Plus Forecast engine.			✓	Red	×
FORECAST_COST_PLUS_NO_VOLUME	Volume lookup failed for Cost Plus Forecast engine.			✓	Yellow	×
FORECAST_MISSING_HISTORICAL_DATA				✓	Yellow	×
MISSING_LINE_ITEM_SCENARIO	The Line Item does not belong to any scenario	Move the Line Item into a Scenario folder		✓	Critical	✓
NO_PRODUCT_GROUP	There is no Product Group selected on Line Item	Select a Product Group		✓	Yellow	×
UOM_CONVERSION_FACTOR_NOT_FOUND				✓	Yellow	×
MISSING_DATA				✓	None	×

Except for the warning message, you can also define the type of alert you want to show, whether it should be visible in the summary matrix, and if it should abort all further calculation or not. There is also a possibility to add solution suggestions and origin of the warning, e.g. database issue, configuration issue etc. You can find thorough description of columns' purpose.

Column name	Value	Required (Yes /No)
Exception code	text, defines exception code used when throwing an exception	Yes
Alert Message	text, message shown in alert displayed to the user	No
Solution	text, possible solution of the problem	No
Origin	text, type of the problem's origin	No
Display in Matrix	true/false	Yes
Alert Typed	Critical/Red/Yellow/Warning/None	Yes
Aborts Calculation	true/false	Yes

Release Notes (Agreements)

- Agreements Accelerator 1.0.0
- Agreements Accelerator 0.1.0

Agreements Accelerator 1.0.0

This document summarizes major improvements and fixes introduced in the Agreements Accelerator release version.

Version	1.0.0
Release Date	Jun 24, 2024

New Features and Improvements

Feature	ID
In the agreement header, the label "Calculation Output Types" has changed to "Additional Calculation Output Types".	PFPCS-8050
In the agreement header, the labels "Default Currency" and "Default UOM" have changed to "Calculation Currency" and "Calculation UOM".	PFPCS-8234
The new Calculation Outputs Result Matrix feature, accessible in the Calculations & Details section, provides a comprehensive view of all line item results in the active scenario, enabling users to evaluate price record generation formulas and view prices in different currencies and units of measure using conversion rates.	PFPCS-7540
To allow users to manage their formula library directly within an agreement, a new "Manage Formulas" input was added above Formula Selection, visible only to users with the "AGR_SalesAgent" or "AGR_FormulaManager" role.	PFPCS-8211
To assess the impact of different contract scenarios, you can now compare historical vs. forecasted impacts on revenue, margin, and volume.	PFPCS-7774
There is a new article in the documentation on how to define your own forecast engine.	PFPCS-8194
Contextual help and placeholder values have been introduced to provide additional information and examples of input values, facilitating easier and more relevant agreement input.	PFPCS-8236
Forecast Calculation Output Types Matrix has been added.	PFPCS-8266
To enable users to compare the impact of different line item selections, a Statistical Forecast Engine has been introduced, allowing for formula comparisons against statistical forecasts.	PFPCS-8210
Forecast Engine has been moved to the header input and to scenarios.	PFPCS-8201
To limit calculation load and detect potential issues, it is possible now to set a maximum number of recalculations per price record.	PFPCS-8192
Data are now generated to Condition Records (instead of Price Records).	PFPCS-8237

To enhance input handling, input blocks have been added to the Formula Designer.	PFPCS-8204
Forecasts have been made optional.	PFPCS-8199
To help users distinguish calculation outputs, color coding has been added to the message column in the output matrix: green for info, orange for warnings, and red for errors.	PFPCS-8111
Error handling using Warning Manager has been added for the header level.	PFPCS-8336
Warnings caught during line item calculation are added now to Warning Matrix on the header level.	PFPCS-8337
To improve error and warning tracking, WarningManager support has been integrated into each element.	PFPCS-8143
It is possible to specify an input (e.g. collapsible, row layout) as a container element and mark other inputs as its contents - using a new column "Container Name" in the PP.	PFPCS-8023
To simplify Formula Designer deployment, new configuration logic for setting up AGR_FormulaTypeDefinition and input table names has been added to the Formula Library.	PFPCS-8329
To streamline input management, bounds inputs have been removed from the Formula Library and moved to the Agreement, where they are now displayed and calculated in the Header Input.	PFPCS-8205
There is new contextual help on the formula detail sections.	PFPCS-8465
It is possible now to select different scenarios for comparison.	PFPCS-8240
To help users select the most impactful scenario on their contract KPIs, a Scenario Comparison Dashboard has been introduced for comparing different scenarios.	PFPCS-7570
It is possible to select a scenario directly from the comparison dashboard.	PFPCS-7541
There is now a contract header summary to help you evaluate which formula to use for condition record generation.	PFPCS-7542
Formula Designer (built via standalone Custom Form Type) has been added, allowing you to define your formula type, input definition, attributes, and send it for approval.	PFPCS-8323
A new warning is now issued if you leave the product group empty, preventing you from generating empty SKU lists and getting no results.	PFPCS-7892
The accelerator now contains two new business roles, Sales Agent and Formula Manager, allowing you to restrict view and edit permissions within the accelerator.	PFPCS-7086

Formula Library has been renamed to Input-Based Formula Library and this custom form type is now hidden by default.	PFPCS-8345
Scenario Comparison Metrics have been renamed to "Forecasted Margin", "Forecasted Revenue" and "Forecasted Volume".	PFPCS-8467

Agreements Accelerator 0.1.0

This document summarizes major improvements and fixes introduced in the Agreements Accelerator release version.

Version	0.1.0 (pre-release, available upon request)
Release Date	Jan 29, 2024

Feature List

The following functionalities have been delivered with the initial release of the Agreements Accelerator.

Feature	ID
Agreement Header: Filtering logic for showing relevant condition types for selected header type	PFPCS-7277
Agreement Header: Input Parameters	PFPCS-6897
Agreement Line Item: Add new Product Configurator	PFPCS-7735
Agreement Line Item: Outputs	PFPCS-7150
Agreement Recalculation: Feeder logic for picking contracts to be recalculated	PFPCS-7276
Agreement Recalculation: Set up scheduled recalculation	PFPCS-7508
Formula Library: Implement AWP for Formula Definitions	PFPCS-7734
Formula Library: Introduce "Cost +" Formula Elements & Definitions	PFPCS-7085
Formula Library: Introduce the Formula Library	PFPCS-7083
Input Manager: Introduce extendable inputs to Contract objects	PFPCS-7124
Price Record: Record Structure	

	PFPCS-7588
Scenario: Folder definition rename	PFPCS-7630
Scenario: Inputs	PFPCS-7247

Archive of Documentation (Agreements)

File	Modified
Accelerate_Agreements_Package-0.1.0.pdf	28 minutes ago by Ondřej Tesař

Drag and drop to upload or browse for files