Promotion Calculation Engine
Document Version: 1.0 – 2018-04-20

SDK Promotion Calculation Engine

SAP Customer Activity Repository 3.0 FP3

Typographic Conventions

Type Style	Description
Example	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Textual cross-references to other documents.
Example	Emphasized words or expressions.
EXAMPLE	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
Example	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE	Keys on the keyboard, for example, F2 or ENTER.

Document History

Version	Date	Change
1.0	2018-04-20	Initial Version

Content

1 Preface

1.1 Introduction

The Promotion Calculation Engine (PCE) is the calculation core of the Promotion Pricing Service (PPS). In an integrated Omnichannel concept, the PCE ensures homogeneous pricing, the consistent application of promotions, and loyalty activities as well as the accurate calculation of refunds across all channels.

The highly configurable solution is designed to handle a large number of price calculations simultaneously across multiple channels and in real time.

The subsequent chapters provide you with an in-depth look into how the PCE operates. You will get to know the algorithm and the technology behind the PCE and learn how to extend it according to your needs.

> Note that this documentation occasionally uses the term "Pricing Engine". This term denotes the core components of the Promotion Calculation Engine. As the Pricing Engine comprises a part of the PCE, it must not be confused with the PCE itself.

1.2 Definitions

The table below lists and explains the PCE terminology used throughout the document. For the complete set of PCE-related terminology, refer to chapter Glossary.

PCE terminology	OPP/PPS Terminology	Description
Calculation Rule	**Price Rule**	Describes the benefit to be granted to the customer in case that the condition is applied.
Condition	**Promotion Price Derivation Rule**	A condition can be understood as a single action granting a benefit to the customer. It consists of a set of triggers (*eligibilities*), time validity, and a rule describing the benefit. Multiple conditions can belong to a *promotion*.
PCE Context		Holds the complete state for transaction processing and promotion calculation.
Eligibility	**Price Derivation Rule Eligibility**	Precondition which has to be fulfilled in order to apply the condition. An eligibility triggers a *condition*.
Line Item	**Item**	An element of a transaction that contains information on goods or services delivered.

PCE terminology	OPP/PPS Terminology	Description
Normalized Item		Representation of a product, an article, a good, or a service that is bought or returned by a customer. An item is the smallest unit or customer pack that can be ordered independently and that cannot be split further into any smaller units.
Pricing Engine		The core of the Promotion Calculation Engine. The Pricing Engine passes the request from the application, performs the price determination, and returns the result.
Promotion	**OPP Promotion**	Functional mapping of a campaign granting a benefit to the customer. A promotion consists of one or more *conditions*.
Request		Message that contains common information pieces which are needed in order to process the message: A shopping basket with multiple line items and other information being relevant for the calculation of the benefits for the shopping basket. The request is provided via the client API.
Response		Responding message regarding to the *request* sent in. This can be the updated shopping basket including calculated discounts and bonus points as well as information about used promotion master data and applied coupons, or an error information in case that the request could not be processed.
Transaction		The shopping basket with line item data and other information relevant for price calculation.

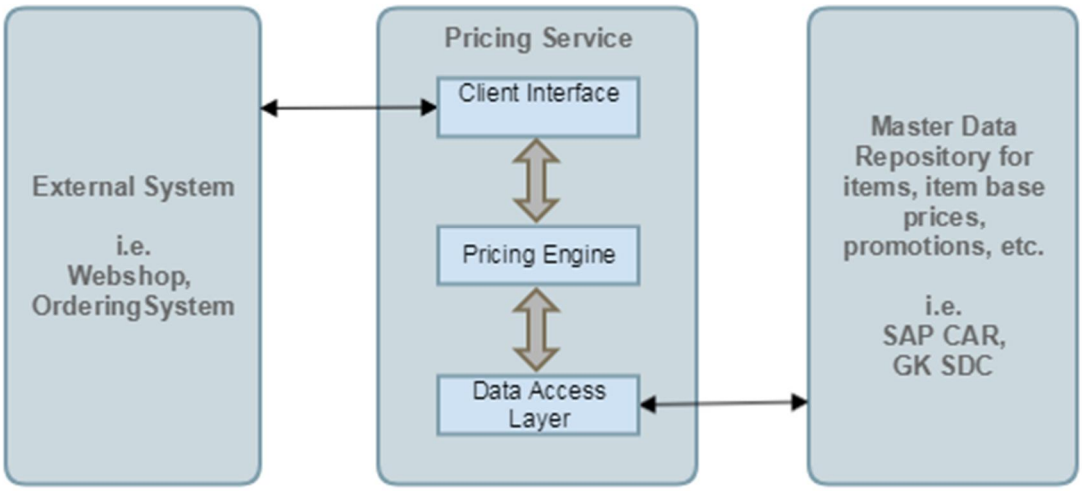
2 Getting Started

The Promotion Calculation Engine (PCE) is a software component designed to calculate promotions over items presented in a transaction. The PCE - besides calculating promotions related to the line items - also processes other parts of the transaction, for example coupons or assigned customer groups among others.

Technically, the PCE receives requests via a client exposed interface API. These requests are expressed in Java objects which store the transaction with all calculation relevant information provided by the client, such as line items that belong to the sales order. The objects contain all necessary information regarding the product. An incoming request is analyzed and mapped to internal PCE model objects. The PCE extracts and analyzes information from the transaction. Afterwards, if applicable, monetary discounts and bonus points are calculated, based on the analyzed data from the transaction and on promotion data.

The result of the PCE calculation is mapped back to an exposed Client API object and is returned to the requesting process, which can be another Java application or an HTTP client.

The following figure demonstrates the PCE process in a larger context:



To process promotions and to apply them to line items, the PCE evaluates promotion definitions. A promotion definition is composed of condition rules (for example, "promotion only applies if you buy for more than a 100€ sum amount") and so-called eligibilities. An eligibility is a trigger for a promotion.

Promotions may be applied on the transaction or on single line items. Position-related promotions are always applied first before transaction-related ones. The order of the applied promotions in case of both position-related and transaction-related discounts is determined by the sequence and resolution values

defined in the promotion master data. In this case of equal sequence and resolution between two promotions, the promotion application must be handled with a promotion collision handler algorithm.

> For further details on the functionality and features of the PCE, refer to the document Functional Guide for the Promotion Calculation Engine.

The API communication within a PPS deployment is stateless and without any GUI. The responsibility for storing information received from the PCE is delegated to the called process.

The transmitted content is based on the ARTS Pricing Service Interface schema.

3 Technologies

Besides Java experience, an extension developer should have some knowledge of the following technologies:

3.1 Spring Framework

Spring is used as a framework for dependency injection and enables the usage of wired components in different combinations.

The PCE uses XML bean context definitions and Spring features to allow extensibility. See the Extensibility chapter for more details.

3.2 ARTS standard

The PCE applies the standard provided by The Association for Retail Technology Standards (ARTS) when defining data exchange message schemes.

The internal data model is based on ARTS standard as well with extensions required for price calculation.

Especially the following ARTS specifications are of interest:

- ARTS Pricing Service Interface Technical Specification v 1.0
- ARTS Pricing Service Domain Model v 1.0.1
- ARTS Retail Transaction Interface Specification v 1.0
- ARTS Retail Transaction Interface Domain Model v 4.0

3.3 Libraries

The following external open source libraries are dependencies of the Promotion Calculation Engine.

All are available via Maven central.

The defined versions of the libraries are the ones that the Promotion Calculation Engine is built and tested against.

A newer version of a library may work if the third party provider did not change their API in an incompatible way.

Furthermore, the listed versions may differ from the dependency versions used in the PPS environment and are specific for the Promotion Calculation Engine.

Group ID	Artifact Id	Version	Scope	Description
org.springframework	spring-beans	3.0.7, 4.1.9 or 4.3.3	compile & runtime	Spring framework
org.springframework	spring-core	3.0.7, 4.1.9 or 4.3.3	compile & runtime	Spring framework
org.springframework	spring-expression	3.0.7, 4.1.9 or 4.3.3	compile & runtime	Spring framework
org.springframework	spring-context	3.0.7, 4.1.9 or 4.3.3	compile & runtime	Spring framework
org.springframework	spring-aop	3.0.7, 4.1.9 or 4.3.3	compile & runtime	Spring framework
org.slf4j	slf4j-api	1.7.21	compile & runtime	Logging API
com.fasterxml.jackson.core	jackson-annotations	2.6.4	compile & runtime	XML & JSON Parsing
com.fasterxml.jackson.core	jackson-databind	2.6.4	compile & runtime	XML & JSON Parsing
com.fasterxml.jackson.core	jackson-core	2.6.4	compile & runtime	XML & JSON Parsing
joda-time	joda-time	2.6	compile & runtime	Datetime utilities

Group ID	Artifact Id	Version	Scope	Description
com.google.guava	guava	12.0	compile & runtime	Guava is a set of core libraries that includes new collection types (such as multimap and multiset), immutable collections, a graph library, functional types, an in-memory cache, and APIs/utilities for concurrency, I/O, hashing, primitives, reflection, string processing, and much more!
net.jperf	jperf	1.0.3	compile & runtime	Performance Monitoring and Statistics for Java Code
commons-collections	commons-collections	3.2.2	compile & runtime	Apache commons collection helper utilities. Extends or augments the Java Collections Framework.
commons-lang	commons-lang	2.6	compile & runtime	Apache commons utilities. Provides extra functionality for classes in java.lang.
com.google.code.findbugs	annotations	3.0.1	compile	Findbugs static code check annotation. Compile time only dependency.

4 How the PCE works

4.1 PCE Request

The Promotion Calculation Engine is called via the Pricing Service API. The requester has to send a PriceCalculate representation. The PriceCalculate object's ARTSHeader element contains request-specific header information, while the PriceCalculateBody contains all required information to perform the price calculation. Only one PriceCalculateBody and one ShoppingBasket is allowed per request.

The representation of a PriceCalculate request in XML format can be seen in the following sample:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PriceCalculate xmlns="http://www.sap.com/IXRetail/namespace/" InternalMajorVersion="1"
InternalMinorVersion="0">
  <ARTSHeader ActionCode="Calculate" MessageType="Request">
    <MessageID>9a89f2edfd1e413ea147e334b9c2ed4b</MessageID>
    <DateTime>2250-01-13T04:48:30.427-05:00</DateTime>
    <BusinessUnit TypeCode="RetailStore">FC01</BusinessUnit>
  </ARTSHeader>
   <PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true">
    <TransactionID>9a89f2edfd1e413ea147e334b9c2ed4b</TransactionID>
    <DateTime>2250-01-13T04:48:30.427-05:00</DateTime>
    <ShoppingBasket>
      <LineItem>
         <SequenceNumber>0</SequenceNumber>
     <MerchandiseHierarchy ID="1">CHA2111</MerchandiseHierarchy>
         <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
            <TaxIncludedInPriceFlag>false</TaxIncludedInPriceFlag>
            <NonPieceGoodFlag>false</NonPieceGoodFlag>

<FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
            <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
            <ItemID>CHA2111001</ItemID>
            <Quantity Units="1" UnitOfMeasureCode="PCE">2</Quantity>
           </Sale>
      </LineItem>
    </ShoppingBasket>
   </PriceCalculateBody>
</PriceCalculate>
```

Code Block 1 Sample 1: Simple request in XML format

4.2 PCE Response

After a successful call, the PCE returns the response in a PriceCalculateResponse object (see com.sap.ppengine.client.dto.PriceCalculateResponse) and informs the client about status using a ResponseCode.

Below you can see the XML representation of the response received for the request presented in sample 1.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PriceCalculateResponse xmlns="http://www.sap.com/IXRetail/namespace/"
InternalMajorVersion="1" InternalMinorVersion="0">
    <ARTSHeader ActionCode="Calculate" MessageType="Response">
        <MessageID>471b53ba-958e-4f8b-804c-d690e1c3f6d7</MessageID>
        <DateTime>2017-04-10T09:29:27.862+02:00</DateTime>
        <Response ResponseCode="OK">
            <RequestID>9a89f2edfd1e413ea147e334b9c2ed4b</RequestID>
            <ResponseTimestamp>2250-01-13T04:48:30.427-05:00</ResponseTimestamp>
        </Response>
        <BusinessUnit TypeCode="RetailStore">FC01</BusinessUnit>
    </ARTSHeader>
    <PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true">
        <TransactionID>9a89f2edfd1e413ea147e334b9c2ed4b</TransactionID>
        <DateTime>2250-01-13T04:48:30.427-05:00</DateTime>
        <ShoppingBasket>
            <LineItem>
                <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
                    <ItemID>CHA2111001</ItemID>
                    <ExtendedAmount>0.0</ExtendedAmount>
                    <ExtendedDiscountAmount>0.0</ExtendedDiscountAmount>
                    <Quantity Units="1.0" UnitOfMeasureCode="PCE">2</Quantity>
                    <TaxIncludedInPriceFlag>false</TaxIncludedInPriceFlag>
                    <NonPieceGoodFlag>false</NonPieceGoodFlag>

<FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
                    <PriceTypeCode>00</PriceTypeCode>
                    <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
                </Sale>
                <SequenceNumber>0</SequenceNumber>
                <MerchandiseHierarchy ID="1">CHA2111</MerchandiseHierarchy>
            </LineItem>
        </ShoppingBasket>
    </PriceCalculateBody>
</PriceCalculateResponse>
```

Code Block 2 Sample 2: Response to request in code sample 1

If a valid request is sent, but with unexpected values or structure, the PCE will inform the client through a Business Error Code in the response header section. You can find detailed descriptions of the Business Error Codes for the concrete error cases in chapter Business Error Codes. The Business Error Code is returned as part of the ARTSHeader element in the response message.

See the following example for details:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PriceCalculateResponse xmlns="http://www.sap.com/IXRetail/namespace/"
InternalMajorVersion="1" InternalMinorVersion="0">
    <ARTSHeader ActionCode="Calculate" MessageType="Response">
        <MessageID>0b15d202-5536-4b1c-bd1e-06f8beee72ba</MessageID>
        <DateTime>2017-04-10T09:38:09.894+02:00</DateTime>
        <Response ResponseCode="Rejected">
            <RequestID>9a89f2edfd1e413ea147e334b9c2ed4b</RequestID>
            <ResponseTimestamp>2017-04-10T09:38:09.893+02:00</ResponseTimestamp>
            <BusinessError Severity="Error">
                <ErrorID>GKR-100500</ErrorID>
                <Description>Value not supported. Element LineItem.SequenceNumber. Sent: -
10. Expected: 0.</Description>
            </BusinessError>
        </Response>
        <BusinessUnit TypeCode="RetailStore">FC01</BusinessUnit>
    </ARTSHeader>
    <PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true">
        <TransactionID>9a89f2edfd1e413ea147e334b9c2ed4b</TransactionID>
        <DateTime>2250-01-13T04:48:30.427-05:00</DateTime>
        <ShoppingBasket>
            <LineItem>
                <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
                    <ItemID>CHA2111001</ItemID>
                    <Quantity Units="1" UnitOfMeasureCode="PCE">2</Quantity>
                    <TaxIncludedInPriceFlag>false</TaxIncludedInPriceFlag>
                    <NonPieceGoodFlag>false</NonPieceGoodFlag>

<FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
                    <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
                </Sale>
                <SequenceNumber>-10</SequenceNumber>
                <MerchandiseHierarchy ID="1">CHA2111</MerchandiseHierarchy>
            </LineItem>
        </ShoppingBasket>
    </PriceCalculateBody>
</PriceCalculateResponse>
```

Code Block 3 Business Error Code GKR-100500 (unexpected value)

4.3 Calculation modes

The PCE currently supports two calculation modes: *Basket* and *LineItem*. The calculation mode controls how the PCE interprets and processes the incoming request, which causes different expected results as well. Calculation modes are supported since interface version 2 (that is, InternalMajorVersion="2"). Some promotions can only be applied in Basket mode, as they might require multiple retail transaction line items present in the transaction to even be eligible. Furthermore, coupons are prohibited in LineItem mode, a

Business Error is returned if one is present in the request. However, if a promotion is allowed and eligible, it will be handled, analyzed, and applied either for Basket or LineItem mode.

4.3.1 Basket mode

The calculation mode *Basket* is the default way to analyze and calculate transactions. All line items are considered and processed at the same time.

- Example 1: If a transaction contains 10 line items with quantity 1, each costs 1€, a transaction coupon of 3€ rebate will be applied and prorated to all items, producing a total price of 7€.
- Example 2: We have a transaction containing 3 line items of "item A" with quantity 1 and price 10€ each. A 10% promotion with an item eligibility (threshold type "QUT" and 3 as threshold quantity) can be applied on all three items.

Basket mode can be explicitly defined in the attribute CalculationMode of the request message's PriceCalculateBody element.

See example below:

```
<PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true"
CalculationMode="Basket">
```

Code Block 4 CalculationMode

4.3.2 LineItem mode

In *Line Item* mode, all line items in the transaction are handled and processed separately. This means that all line items in the request will be split and processed by the PCE independently and the individual results will be summarized in the end. Note that the Client API only supports one PriceCalculateBody and one ShoppingBasket elements in the request message.

- Example 1: If a transaction contains 10 line items with quantity 1, each costs 1€ and a transaction coupon for 3€ discount, a Business Error will be returned, as coupons are prohibited in LineItem mode.
- Example 2: We have a transaction containing 3 line items of "item A" with quantity 1 and price 10€ each. A 10% promotion with an item eligibility (threshold type "QUT" and 3 as threshold quantity) cannot be applied, as the request is split three ways and processed separately. None of the three reaches the defined threshold quantity individually.

Line Item mode can be explicitly defined/requested using the attribute CalculationMode of the request message's PriceCalculateBody element.

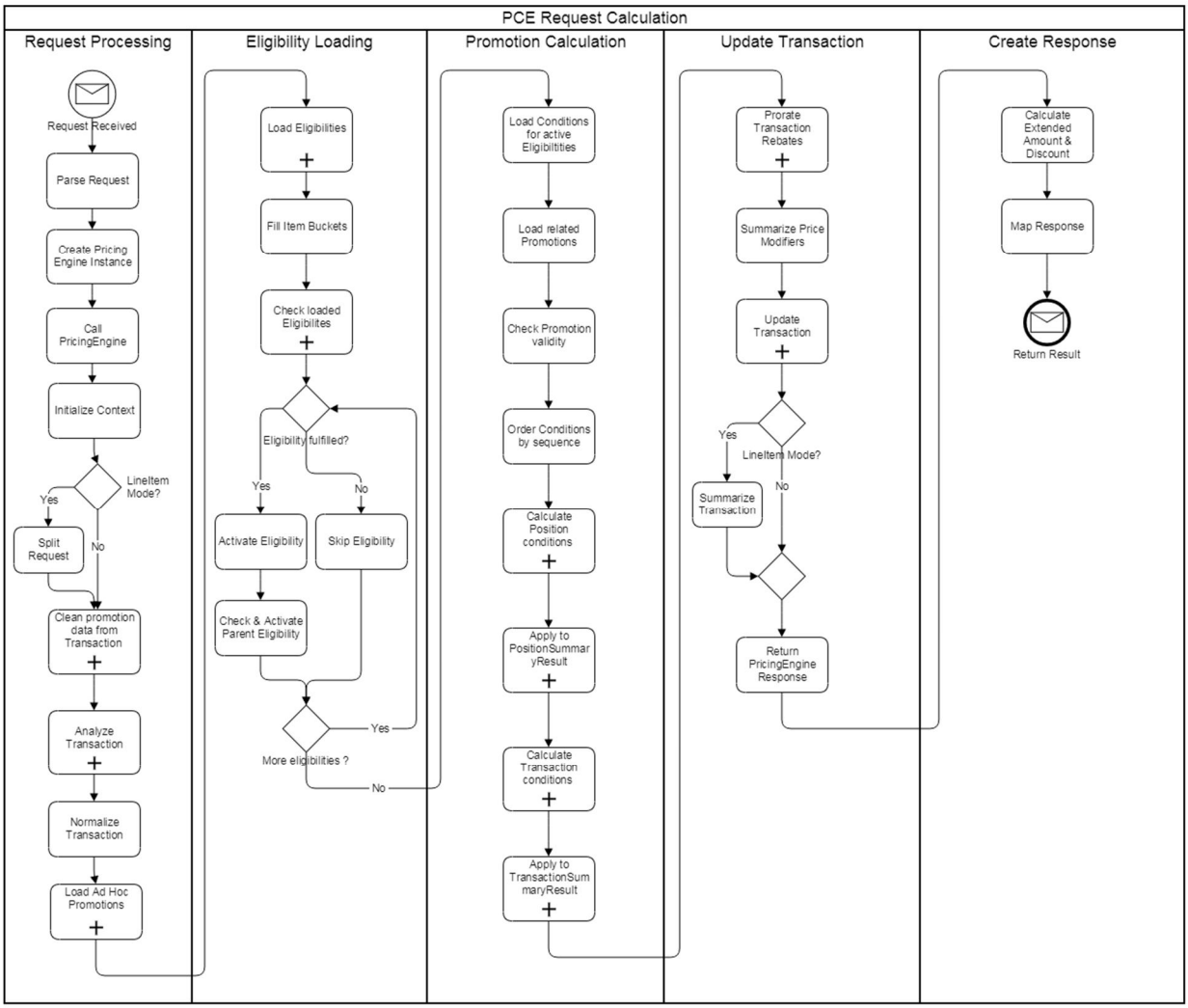
See example below:

```
    <PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true"
CalculationMode="LineItem">
```

Code Block 5 CalculationMode

4.4 PCE Processing

In order to achieve the expected results, the PCE executes a series of well-defined steps. The following figure demonstrates these execution steps:



You find a detailed explanation of these steps in the following sections. More diagrams can also be found in Appendix C.

4.5 Request processing

The process starts when the system receives a request of type PriceCalculate. In the phase of Parse request, the request is validated and mapped to internal Java models with the help of a data mapper object. Afterwards, an instance of Pricing Engine is created. Pricing Engine is called passing through a Pricing Engine request containing configuration data and the mapped request.

HOW IT WORKS

Parse request: The request of type PriceCalculate is parsed, validated, and mapped to internal object models to be processed. Validations are done in this step, for example, check data format integrity, check correlative sequence numbers, check constraints regarding max permitted values, etc. If something is wrong according to specification, a business error is produced and the execution is interrupted.

Map Any fields: The Any fields included in the PriceCalculate object are also processed and mapped to the internal model during the parsing of the request. The Any fields are reserved for custom extensions of the PriceCalculate object to support any possible customer requirements. These custom fields must be mapped to the PCE's internal transaction model to be accessible throughout the entire calculation process and to be available for mapping back to the response PriceCalculateResponse object if necessary. For this reason, each model object of the internal transaction model extends the TxModelDto interface (com.gk_software.pricing_engine.model.transaction.TxModelDto), which defines the List<Object> getAny() method for accessing the extension data provided in the request.

Create Pricing Engine instance: An instance of PricingEngine Interface (PricingEngineImpl) is created with the help of a PricingEngineFactory class.

Call Pricing Engine: Call Pricing Engine main method process passing through a Pricing Engine request that contains configuration data and mapped request. At this step, the called calculation mode is checked. If line item mode is requested, the line item will be split into separate transactions to be processed independently. Otherwise, the default basket mode is used and the transaction with all line items from the request is processed as a whole.

Initialize PCE context: All information during the analysis and calculation is stored into a PCE context. The PCE context is available during the whole calculation process and updated as needed. This step ensures that all initially available information is stored.

Clean promotion data from transaction: Useful for line item mode where the line items are processed independently. Cleans the PCE context and makes sure that no information of a finished process interferes with the process of the next line item.

Analyze transaction: Phase of analysis where every line item is analyzed and classified. The analyzers are split by functionalities and can be easily extended by implementing the interface TransactionAnalyzer. Every analyzer has to store information in the PCE context.

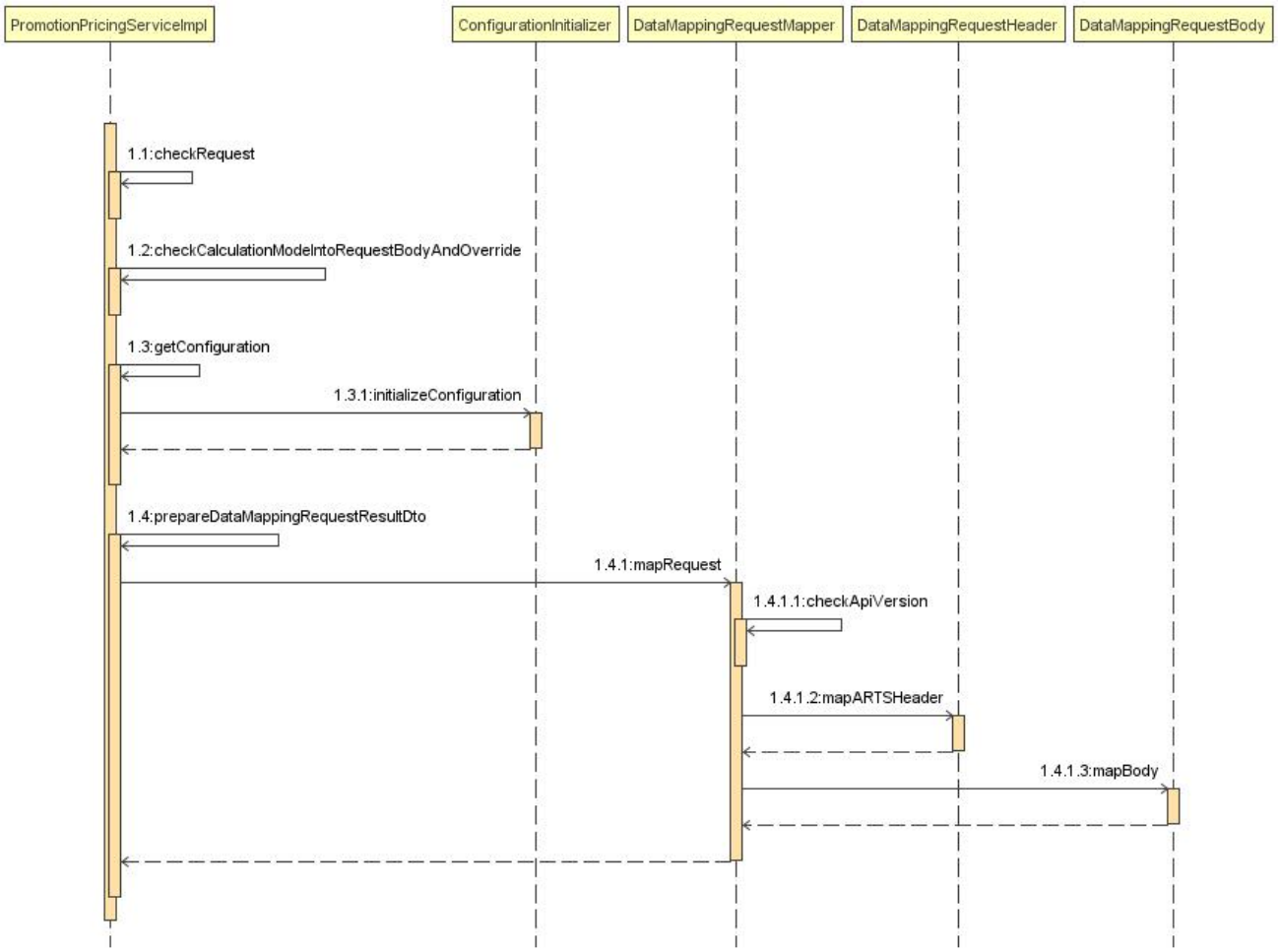
Currently implemented analyzers are:

- *Coupon transaction analyzer*: Responsible for processing and storing coupons from the request.
- *Customer group transaction analyzer*: Responsible for looking for customer group-related items.
- *External promotion transaction analyzer*: Responsible for looking for external promotions like external price modifiers, external line items, loyalty reward, or price derivation rules.
- *Item transaction analyzer*: Responsible for extracting line item basket values, transaction total basket values, finding and processing of related merchandise hierarchy groups.
- *Promotion manual trigger analyzer*: Responsible of looking for manual triggers related to line item position as well as transaction level rebates.

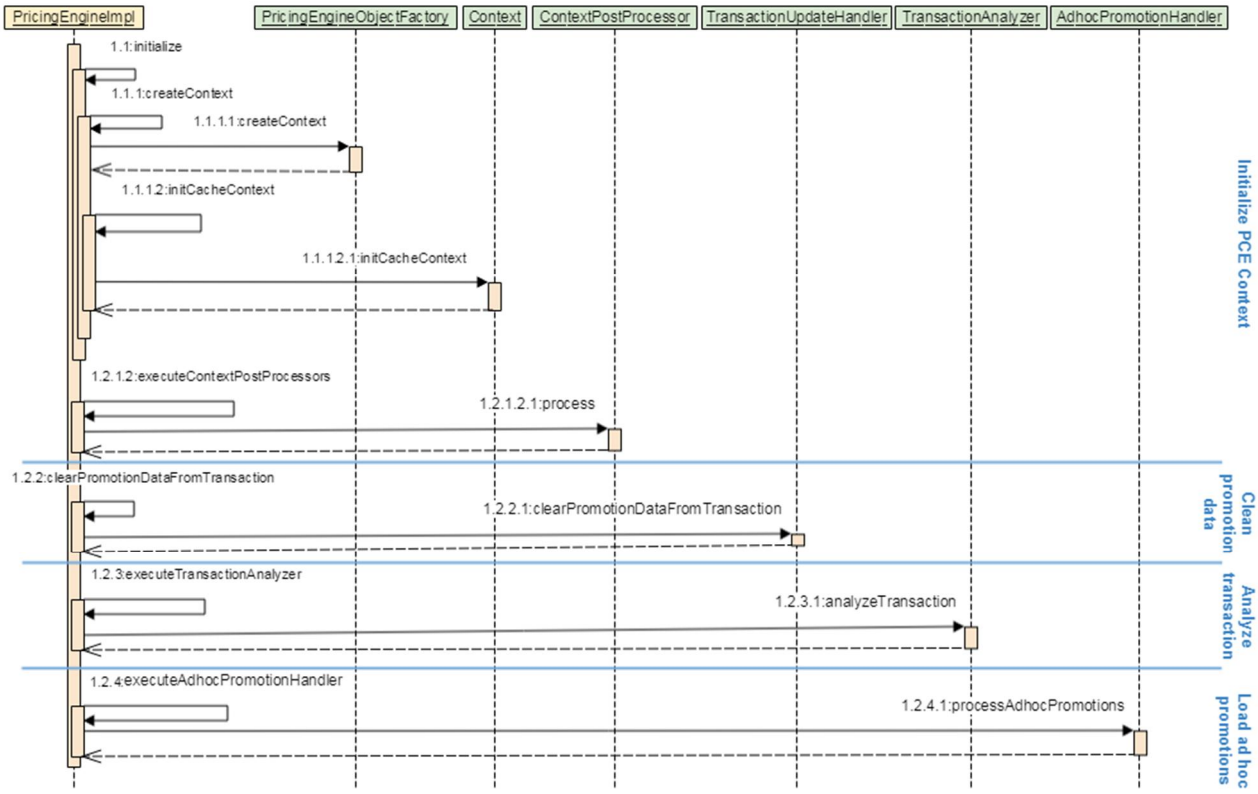
Normalize transaction: Line items are normalized according to their quantity. For instance, if a transaction contains a line item with quantity 10, the result of normalized process will be a transaction with 10 line items with quantity 1. This ensures uniform results of the price calculation, regardless of the structure of the transaction (1 line item with quantity 10 must give exactly the same results as 10 line items with quantity 1). A detailed description of this process and its purpose can be found in chapter Normalized Item Calculation.

Load ad hoc promotions: Ad hoc promotions provided by external systems are loaded in this step in case of available AdhocPromotionLoader plugins. The ad hoc promotion are validated and processed seamlessly with other promotion data retrieved from the data access layer, so it is crucial that they are loaded before eligibility or condition loading and processing. The loading of these promotions and their registration to the PCE context is performed by the AdhocEligibilityHandlerImpl, the core implementation of the AdhocEligibilityHandler interface.

The following diagram shows the sequence of involved classes of the first steps of the request processing: parsing the PriceCalculate request to the internal PCE model. As various elements of the PriceCalculate object contain any fields, the mapping of these is performed as part of the request mapping steps.



The following diagram shows in detail the call sequence of involved classes of the request processing, after the PriceCalculate mapping to the internal PCE model was completed.



4.6 Eligibility loading

Eligibilities are the triggers for applying a promotion. Eligibilities contain all necessary information that enables the PCE to determine whether a promotion is applicable on a certain line item. A promotion must contain at least one eligibility. The eligibilities are loaded from the data access layer.

HOW IT WORKS

Similar to the transaction analyzer, the eligibility data loader is split into well-defined modules, called *fragments*. Each fragment is responsible for loading a specific type of eligibility from the data access layer and for activating and registering it into the PCE context. The fragments are managed by the DefaultEligibilityLoader The PCE implements the following fragments:

- Coupon eligibility loader
- Item eligibility loader
- Merchandise group eligibility loader
- Merchandise set eligibility loader
- Market basket amount eligibility loader

- Customer group eligibility loader

In addition to the DefaultEligibilityLoader, there is a special loader for manual eligibilities: the ManualTriggerEligibilityLoader. This is because ManualEligibilities have to do some advanced context-specific checks and handling.

Besides loading the eligibility type specific promotion master data, the eligibility loading process is responsible for several other steps regarding the preparation of the loaded eligibilities for calculation. Some of these steps are explained below:

- Load eligibilities: Eligibilities are loaded from the persistence layer.
- Fill item buckets: Based on the analyzed data, the item bucket is saved into the PCE context. Bucket is a special data abstraction for items that are handled by the same eligibility. More about buckets can be found in chapter Normalized Item Calculation.
- Check loaded eligibilities and activate eligibility: Checks whether eligibility is valid (active, not expired, amount threshold was achieved,...) and whether items apply to eligibility. If true, the eligibility is activated.
- Check and activate parent eligibility: If an eligibility has a parent eligibility, it will also be checked and activated if the item conditions are fulfilled.

The following diagram explains the eligibility loading process in detail.

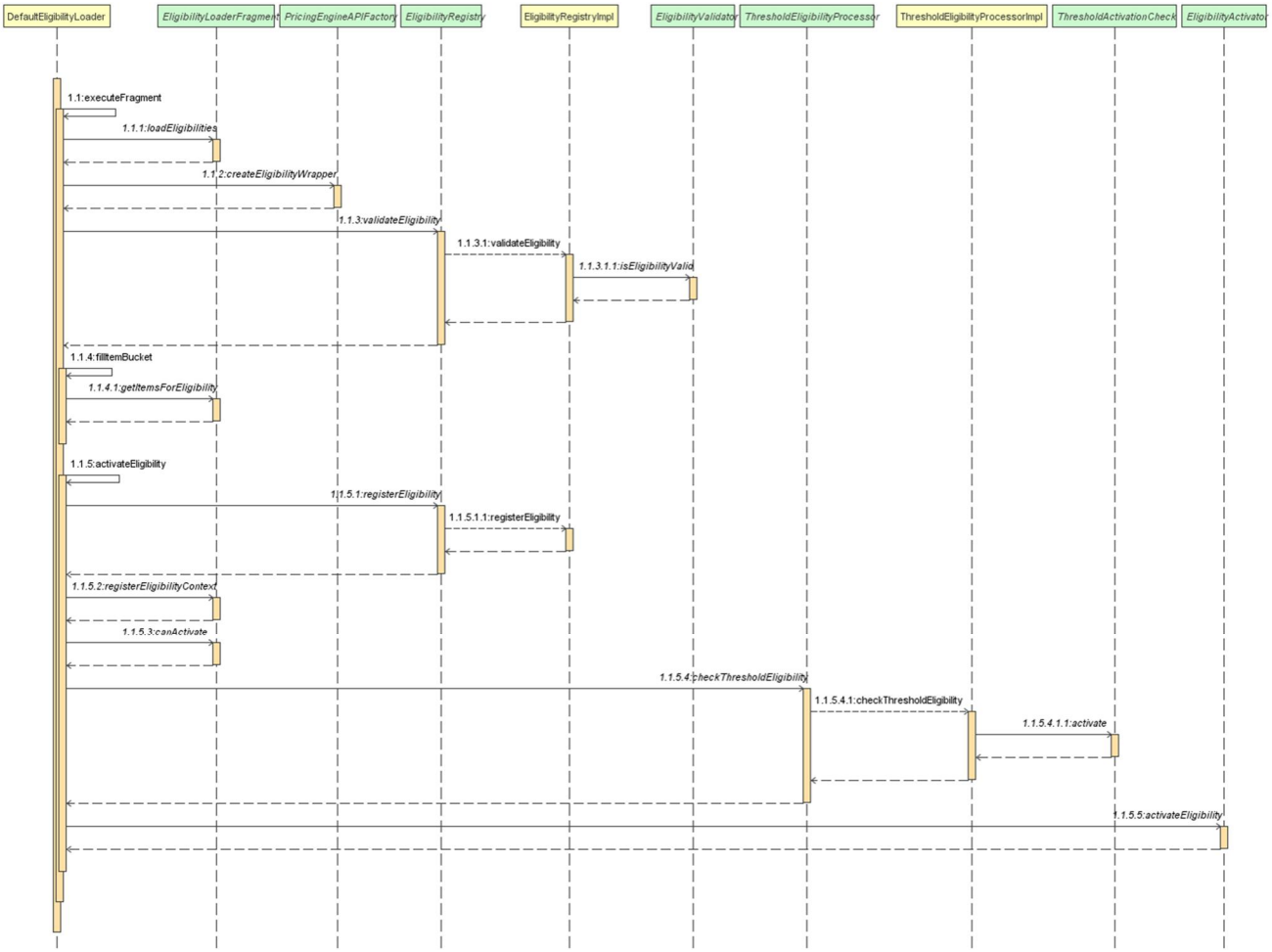


Figure 1 PCE Eligibility Loading

4.7 Promotion calculation

This phase is responsible for calculating the promotions for the normalized items. The calculation and distribution of rebates and bonus points is performed for position- and transaction-related promotions as well. The calculation is based on the analyzed transaction data and on promotion-related master data (promotion conditions, promotion condition rules, etc.) loaded form the data access layer for the activated eligibilities.

HOW IT WORKS

The promotion calculation is responsible for the loading of promotion data related to the activated eligibilities and for calculating the overall benefit.

The steps of the promotion data loading process are explained below.

- Load conditions for activated eligibilities: Promotion conditions and related promotion condition rules are loaded for all of the activated root eligibilities from the data access layer and other sources defined by the registered condition loader fragments.
- Load related promotions: Promotion information is loaded from data access layer and other sources defined by the registered condition loader fragments.
- Check promotion validity: Checks whether the loaded promotions and conditions are valid - effective, not expired, and so on. Promotion eligibilities are already validated.

The following diagram illustrates the condition loading process.

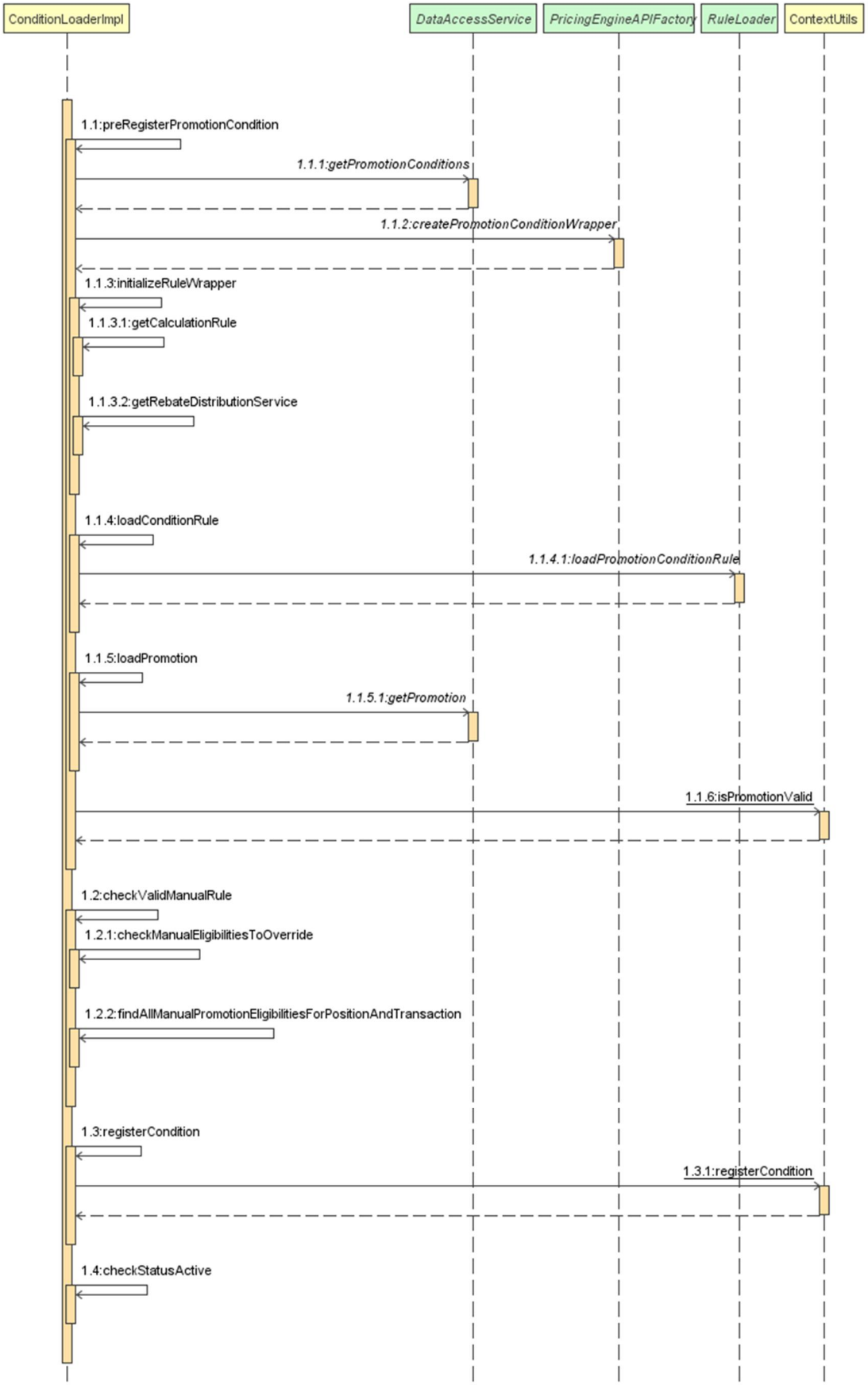


Figure 2 PCE Condition Loading

After the promotion data has been successfully loaded and validated, the calculation is performed. The steps of the calculation process are explained below.

- Order conditions by sequence: Conditions are ordered by sequence number and resolution for processing.
- Calculate position conditions: First position-related conditions are calculated. The calculation is done on normalized items for consistent results regardless of the transaction's structure.
- Apply to position summary results: After the calculation is performed, the resulting information is saved into the position summary result.
- Calculate transaction conditions: Calculate conditions related to transaction.
- Apply transaction summary result: After the calculation on transaction level is finished, the resulting information is saved into the transaction summary result.

The following diagram illustrates the calculation process of conditions.

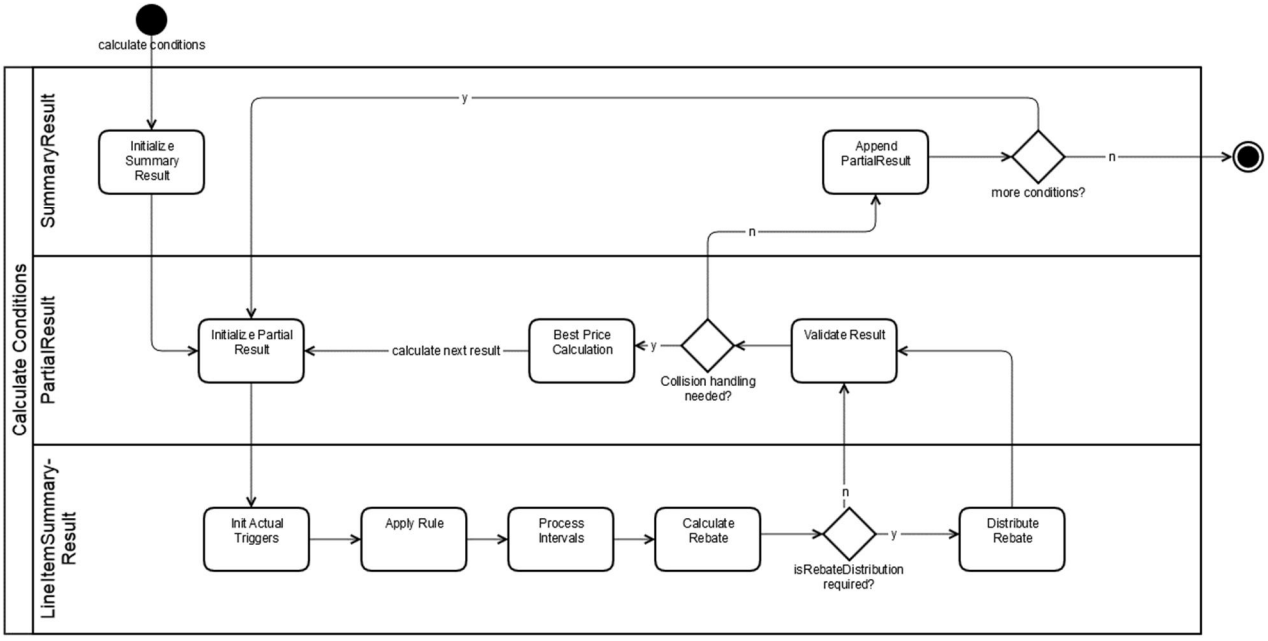


Figure 3 PCE Condition Calculation

4.8 Update transaction

After performing all calculations, the transaction must be updated with the result. The update transaction phase is responsible for creating transaction model objects for the response with the calculated promotions both on item and transaction level and for updating other related information, such as details on used coupons.

The related steps are explained below:

- Summarize price and point modifiers: Retail price modifiers and frequent shopper point modifiers are summarized first. As the calculation is performed on normalized items for consistent results, the modifiers are also created for each individual normalized item. However, the response's structure provided by the PCE to the caller must match the request transaction. For this reason, the modifiers of all normalized items related to the same transaction line item are merged. The line items are then updated with these summarized modifiers.
- Update transaction: Transaction model objects reflecting the calculation result for positions and transaction (for example, sale return line item modifier coupons, sale return line item modifier references, or discount line items) are created and the transaction is updated with them.

The following diagram illustrates the process of updating the transaction with the calculated promotions.

Figure 4 PCE Update the Transaction

4.9 Create response

After the transaction was updated with the calculated results, it is mapped to the PriceCalculateResponse object, which is then sent back to the calling process.

HOW IT WORKS

DataResponseMapper object is used to create PriceCalculateResponse object.

Prepare response mapping context: DataMappingResponseContext is created using DataMappingDefaultObjectFactory class. It is then filled with updated Transaction object, original PriceCalculate request object and Configuration object.

Map response header: ARTSCommonHeaderType object is created. It is filled with newly created random response. The header's messageID element is filled with a pseudo randomly generated UUID, while the DateTime if filled with the current date and time. The response object also contains original request ID and request timestamp. Both are mapped from the request object. The response code is set to OK, if there was no problem during the calculation, or to REJECTED and corresponding error code.

Map response body: PriceCalculateBase object is created. Response body fields are then mapped from the original request, where it is necessary, or from the PCE model (mapped from request and updated during request processing). The shopping basket is mapped completely from the PCE model, using internal mappers:

- DataResponseLineItemMapperService - mapping for line items.
- DataResponseCouponMapperService - mappings for coupons.
- DataResponsePromotionManualTriggerTypeMapperService - mappings for manual triggers.
- DataResponseExternalTriggerMapperService - mappings for external triggers.

Create ARTS response: PriceCalculateResponse object is created. ARTSCommonHeaderType (header) and PriceCalculateBase (body) are linked and InternalMajorVersion and InternalMinorVersion are filled from the request.

Map ANY fields: PCE model objects contains also a list of plain Java objects of any type (java.lang.Object) called *ANY fields*. This is part of the extension concept. They are originally copied from the request but can be changed during the PCE calculation process. During the response mapping step, they are copied back into the response. Example:

```
...
<PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true"
CalculationMode="Basket">
   <any>
```

```
        <Street>Neue Bahnhofstrasse 21</Street>
        <City>Sankt Ingbert</City>
        <PostalCode>66386</PostalCode>
        <Country>Deutschland</Country>
    </any>
...
```

4.9.1 Response Header Mappings:

Response Field	Mapped From	Value
PriceCalculateResponse InternalMajorVersion	Request	
PriceCalculateResponse InternalMinorVersion	Request	

Response Header Field	Mapped From	Value
ARTSHeader ActionCode	Request Header	
ARTSHeader MessageType		Response
ARTSHeader.MessageID		Generated (random)
ARTSHeader.DateTime		Current timestamp
ARTSHeader.BusinessUnit	PCE Model	Transaction.BusinessUnitID
ARTSHeader.Response ResponseCode		OK, REJECTED
ARTSHeader.Response.ResponseTimestamp	Request Header	Request timestamp
ARTSHeader.Requestor	Request Header	
ARTSHeader.WorkstationID	Request Header	
ARTSHeader.RequestedLanguage	Request Header	
ARTSHeader.RequestedMultiLanguage	Request Header	

4.9.2 Response Body Mappings:

Response Body Field	Mapped From	Value
PriceCalculateBody TransactionType	Request Body	
PriceCalculateBody NetPriceFlag	Request Body	
PriceCalculateBody CalculationMode	Request Body	Default: Basket
PriceCalculateBody.TransactionID	PCE Model	Transaction.Key.TransactionID
PriceCalculateBody.DateTime	Request Body	
PriceCalculateBody.ShoppingBasket	PCE Model	
PriceCalculateBody.RegularSalesUnitPriceRoundingRule RoundingMethod	Request Body	
PriceCalculateBody.RegularSalesUnitPriceRoundingRule Multiple	Request Body	

4.10 Normalized Item Calculation

4.10.1 Calculation Overview

The item prices and promotions are calculated from the item data received in a transaction. The transaction is composed of line items (positions), each referring to a specific item from the article master data with different quantities set. The structure of the transaction should have no influence on the result of the calculation.

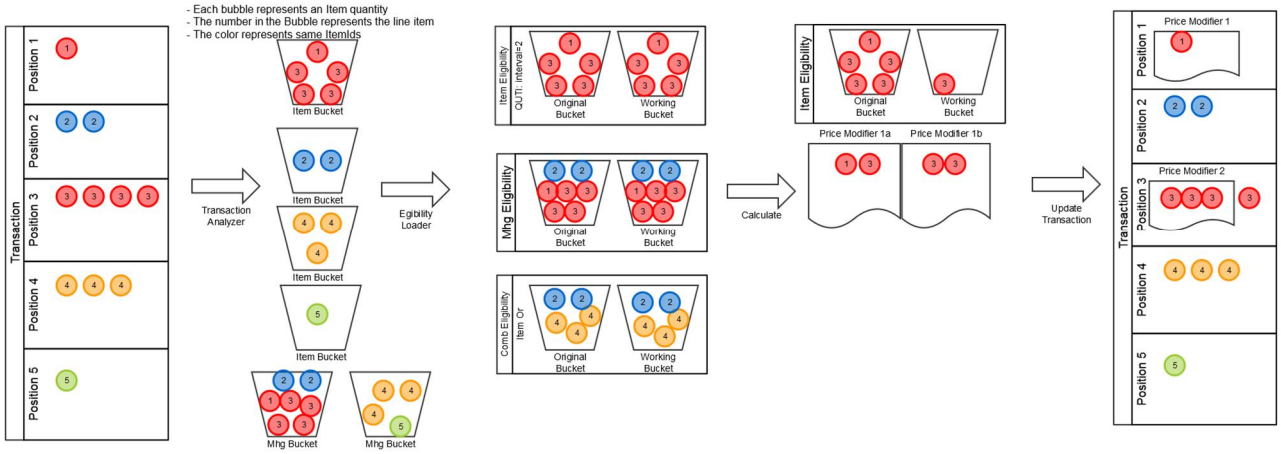
Thus, a position with a quantity of two is semantically the same as two positions of the same item each with a quantity of one. As a real quantity is a combination of unit and quantity, the normalizations lead to items with unit=1 and quantity=1. Any fractional units will lead to an additional NormalizedItem with units<1 (the remaining) and quantity=1.

To reflect this, the PCE uses a normalized item calculation for calculating the promotions. This means each position with quantity not one will be converted to multiple positions with quantity of one for the calculation, a so-called NormalizedItem. When the calculation is finished, the results are then aggregated to the real position structure inside the transaction.

During calculation, these NormalizedItems will be stored in special collections called buckets. A bucket is a collection that allows set operations and includes sorting, filtering, and sum calculation functions. Buckets

are used as the standard collection of NormalizedItems throughout the PCE calculation process. The bucket is created for every activated eligibility containing the eligible NormalizedItems, or as another example the NormalizedItems of the same merchandise hierarchy group or with the same item ID are also stored in buckets in the PCE Context. For more information about buckets please refer to chapter Bucket API.

The following image illustrates the bucket calculation algorithm.



On the left side, there is a sketch of a transaction with five positions (line items) inside. To easily differentiate the items visually, each item (item ID) is represented with a different color (position 1 and 3 contain the same item - in the example represented by red color).

- Position 1 is a red item with a quantity of 1.
- Position 2 is a blue item with a quantity of 2.
- Position 3 is again a red item with a quantity of 4.
- Position 4 is a yellow item with a quantity of 3.
- Position 5 is a green item with a quantity of 1.

During the analysis of the transaction, the positions are normalized to the NormalizedItems (each represented by a bubble). Then they are put into item buckets based on their item type (color). In addition, items are also grouped (put to the bucket) based on transaction merchandise hierarchy groups. In the example we assume that there are two merchandise hierarchy groups defined. One for red and blue items and one for green and yellow items. For each merchandise hierarchy group, an MHG bucket is created which contains all related items.

As each bucket holds only references to the items, the red item 1 in the red item bucket is the same as the red item 1 in the MHG bucket. This allows more complex combinations to use set operations like unions or intersections to define the working set for a calculation.

Each eligibility has a reference to an original bucket which contains all related NormalizedItems and a working bucket that is used during calculation loops and form which elements are removed if the promotion was applied. This is required, for example, for interval calculations.

During eligibility loading step, the buckets are created also for all activated eligibilities. They may be the same as buckets created during transaction analysis (like item or MHG eligibilities), or different (like combination or merchandise set eligibilities). In the example, you see an ItemEligibility for red items, a Merchandise Hierarchy Group Eligibility for red/blue MHG, and an Item Or CombinationEligibility containing blue and yellow items.

During calculation of the promotion, temporary PriceModifiers A-D are created for the red items. Each modifier is related to some specific NormalizedItem. This example assumes a promotion with an interval of two, so the promotion can be applied twice and uses four items.

One item is left in the working bucket as the interval is not fulfilled anymore.

During update of the transaction with the results, the temporary PriceModifiers are aggregated to the final PriceModifiers on the positions. The result is that position 1 has one PriceModifier with an applied quantity of 1 and position 3 has a PriceModifier with an applied quantity of 3 (of 4 item quantities available).

Due to the possibility of performance decrease as a result of the normalization of large number of items the minimumQuantityForSingleNormalizedItemCreation configuration parameter was introduced. The integer value defines the minimum number of line items which must be present in the transaction for disabling the standard item normalization. In case this item count is exceeded a single normalized item will be created for each different item in the transaction with quantity >= 1. For more information regarding the PCE configuration please refer to the Functional Guide.

4.10.2 Discount Calculation and Rounding

Due to the normalized calculation, the discount amounts are rounded on NormalizedItem level as well. This can lead to different values for rounding than it may be expected for a position.

The rounding element consists of a value and a direction.

The rounding value is the absolute difference between the calculated discount before rounding and after rounding. The rounding direction reflects the direction of rounding (UP or DOWN).

Rounding always is done on the discount amount and not on the new prices.

The following chapters show through examples the calculation on NormalizedItem level and the resulting rounding amounts.

4.10.2.1 Position Discount rounding

A position that has a regular sales unit price of 15.95 and a quantity of 10 should get a 10% rebate each.

We assume the rounding decimal places is set to 2 and rounding destination value to 0.01 (round to the nearest cent denomination) and rounding direction should be half up.

The PCE normalizes this to 10 NormalizedItems with a quantity of 1 each.

So a 10% Rebate will calculate a discount amount of 15.95 * 0.1 = 1.595.

This will round to 1.60. The rounding value is 0.005 (1.60-1.595) and the rounding direction is UP for a single NormalizedItem.

In the result, these discounts are summed up for the position (10 times). This leads to a discount of 1.60*10 and a new price of 143.50. The rounding value for the position would then be 0.05.

Note the difference if the engine would calculate on position level completely. Then we could assume a rebate of 15.95*10*0.1=15.95 with a rounding value of 0! But this would lead to different discounts depending on the structure of the transaction. Having 1 position with quantity 10 would have different result than 10 positions with quantity of 1.

Therefore, to ensure consistent rebate calculation the PCE uses the normalized approach.

4.10.2.2 Transaction Rebate Distribution

Now we assume the same transaction with one position that has a regular sales unit price of 15.95 and a quantity of 10.

But this time we have a transaction rebate of 10%.

We assume the rounding decimal places is set to 2 and rounding destination value to 0.01 (round to the nearest cent denomination) and rounding direction should be half up.

The PCE normalizes this to 10 NormalizedItems with a quantity of 1 each.

So a 10% Transaction rebate will calculate a discount amount based on the overall basket amount of 15.95 *10 = 159.50. Getting 10% rebate on this will lead to a discount of 159.50 * 0.1 = 15.95.

This overall discount is then distributed to the single NormalizedItems.

For each item, the related part of the discount will be calculated like 15.95*0.1 = 1.595.

This will again round to 1.60. So the rounding value is 0.005 (1.60-1.595) and the rounding direction is UP for a single Normalized Item.

This is true for the first 9 items. The last item then gets the remaining discount. That means we used up of the overall discount of 15.95 for the first 9 items each a discount of 1.60. So the remaining discount is 15.95-9*1.6=1.55.

As the last item simply gets what is left, no rounding is done as 1.55 already fits to 2 decimal places with destination value of 0.01.

In the result, the discounts are summed up for the position. This leads to a discount of 1.60*9+1.55=15.95 and a new price of 143.55. The rounding value for the position would then be 9*0.005=0.045.

Due to the different ways of calculating position rebates and transaction rebates we may see slightly different results for what seems the same rebate (10%). Keep this in mind when designing your promotions. Always consider the normalized calculation approach will behave as if a transaction contains only positions with quantity 1.

5 PCE Modules

5.1 Overview

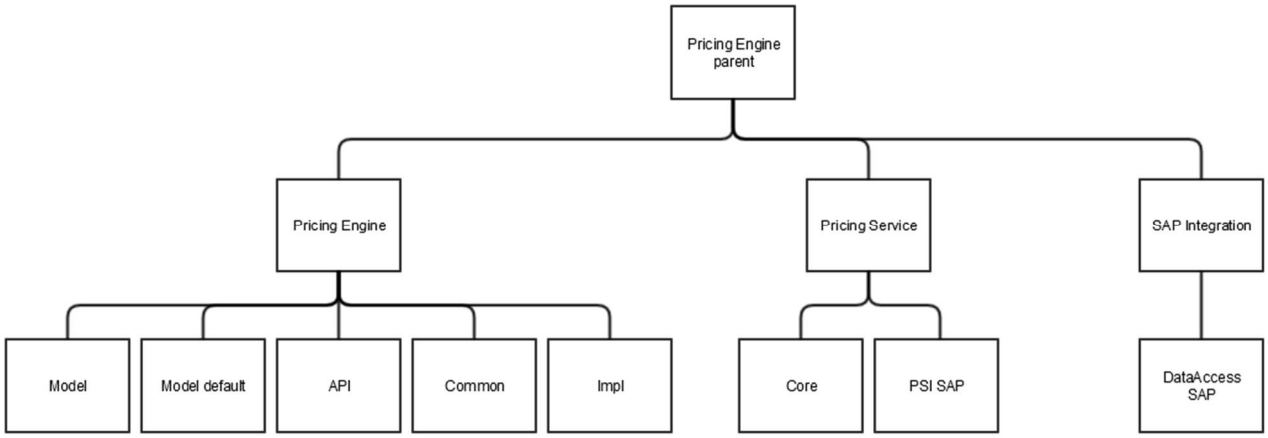
The PCE is composed of several modules. Each module provides a jar file for runtime deployment. The modules are grouped in three main modules that define responsibility of the contained modules.

The three main modules are:

- Pricing Engine: Contains submodules for the concrete promotion calculation. Most of the business logic can be found here.
- Pricing Service: Provides the integration to the service layer via the Client API. Mainly the mapping of the Client API to the internal data model.
- SAP Integration: Contains a single submodule responsible for accessing the master data via the data access layer. It provides mapping of the data access API data model to the internal PCE data model.

Other modules:

- Model: Defines the internal data model of PCE core.
- Model Default: Standard implementation of internal PCE model.
- API: Defines the public API of the PCE core.
- Common: Contains the internal PCE API which is not part of public API.
- Impl: Base implementations for public and internal API, most of the business logic functionality is implemented here.
- Core: Pricing engine service client API and base implementation.
- PSI SAP: Pricing engine service SAP integration.
- Data Access SAP: Mapping of SAP data model into internal PCE model, interface for accessing data layer.



5.2 Spring Context

The PCE core is configured and initialized via Spring XML contexts.

It is assumed that a developer who writes an extension for the PCE knows Spring and its basic concepts.

There are three main XML files needed to set up the complete PCE core context:

- META-INF/com.gk-software.pricing-engine/psi-sap/common.xml: The core engine beans. Defined in pricing-engine-impl module.
- META-INF/com.gk-software.pricing-engine/psi-sap/beans.xml: Service-related beans like request/response mapper. Defined in pricing-engine-psi-sap module.
- META-INF/com.gk-software.pricing-engine/dataaccess-sap/beans.xml: Definitions of data access for SAP data layer. Defined in pricing-engine-dataaccess-sap module.

The following Spring contexts are imported into application context to be able to initialize the complete PCE core:

```
<import resource="classpath:/META-INF/com.gk-software.pricing-engine/psi-sap/common.xml" />
<import resource="classpath:/META-INF/com.gk-software.pricing-engine/psi-sap/beans.xml" />
<import resource="classpath:/META-INF/com.gk-software.pricing-engine/dataaccess-
sap/beans.xml" />
```

5.3 Pricing Engine Model

5.3.1 Overview

The pricing-engine-model module defines the internal data model of the PCE core. The model is defined mostly by interfaces. These interfaces are part of the public API and are kept release-stable for usage. That means extensions based on these APIs should still run on newer version of the PCE core without any migration needed.

> For a more detailed description of the defined stability levels, refer to chapter PCE Extensions.

The standard implementation of this model API is provided in the pricing-engine-model-default module. The classes in that module are not public API. For the GK/Retail OmniPOS integration, a specific implementation of the model API is provided for internal POS usage.

The model has three important parts:

- Typecodes
- Promotion Masterdata model
- Transaction model

In addition, there is a small GK/Retail OmniPOS-specific part for rounding rule definitions and merchandise hierarchy group handling.

5.3.1.1 DataModel Factories

The data model defines factory interfaces for creating concrete model object instance. The implementation used of these interfaces is injected via the Spring application context. By using the Spring bean replacement approach, it is possible to use different implementations in the existing application context.

Factory Interface	Default Bean	Bean Alias	Description
IPromotionFactory	defaultPromotionSOFactory	promotionSOFactory	Factory for promotion master data objects.
ITransactionFactory	defaultTransactionFactory	transactionFactory	Factory for transaction model objects.

5.3.2 Data Model

This chapter provides a high-level overview on all Pricing Engine model interfaces with a brief description.

For details on the model, refer to the Javadoc of the pricing-engine-model project.

5.3.2.1 Typecodes

All typecodes are designed as interfaces with two implementations: One enum implementation for the standard codes and one String-Code implementation for custom or unknown values.

The typecodes all extend the ValueTypeEnum interface that defines specific comparison methods for checking equality against a String or another typecode. Always use these defined "is" methods for comparing ValueTypeEnums and not the equals method of Java.

> Note that while the enum implementations are part of the API, they are just release-stable regarding existing values. The values can be extended without notice to support additional use cases and functionalities.
>
> Consider that additional values may be added in the future, especially when referencing these enums in loops or switch statements. Therefore, it is advised to implement a reasonable default handling in case you must use these enums in loops or switch statements.

5.3.2.1.1 General Typecodes

In the package com.gk_software.pricing_engine.model.typecodes, you find general typecodes that are used inside the calculation logic or defined via configuration.

Specifically, these are:

Interface	Description
DiscountVectorMethodCode	Defines which positions are checked against the discount vector.
RebateShareCalculationMethodCode	Defines the method for proportionally distributing the receipt discount among the different receipt line items (for receipt related monetary discounts, not used for points): Defined by Pricing Engine Configuration.
RoundingMethod	Rounding methods defined in Configuration.
RoundingPercentualPointsMethodCode	Defines the method which is used to calculate percentual points condition values.
TimeValidationMethodCode	Method to determine the validity level of a condition defined in Configuration.

Interface	Description
TransactionAnalyzerType	Transaction analyzer type useful for plugin delimiter identification.
TransactionRebateCalculationMethodCode	Specifies how to determine the calculation base of the receipt related discount.

5.3.2.1.2 Promotion Master Data Codes

The package com.gk_software.pricing_engine.model.typecodes.promotion contains typecodes referenced in the promotion master data model. Besides the interfaces, you find here the enums with the default values. The String-based generic implementations are defined in com.gk_software.pricing_engine.model.typecodes.promotion_default.

Interface	Description
AdjustmentMethodCode	A mnemonic code denoting what kind of adjustment is being made to the retail price of the item.
AmendmentTypeCode	Determines whether the condition can be used only for sales, only for amendments, or for both: - 00/null: for sales as well as for amendments - 01: only for sales - 02: only for amendments.
CalculationBaseType	The parameter CalculationBase is only influencing the calculation base for transaction-related conditions.
ChooseItemMethod	Defines the order in which items are used in promotion calculation, for example, lowest priced first or highest priced first.
CombinationCode	Code denoting the style of combination that is to be applied across the child eligibilities, for example, logical AND, logical OR.
CouponConsumptionCode	Enum for consumption code for coupons.
CouponPrintoutRule	The type code of the printout rule SEPARATE_RECEIPT as "00" PRINT_AT_END as "01".
EligibilityStatusCode	Enum for eligibility status codes values for PromotionConditionEligibilitySO.getStatusCode().
EligibilityType	Eligibility type interface.
MerchandiseSetTypeCode	Defines type of Merchandise set element MRHRC - Merchandise hierarchy group ITM - Item OPR - Set operation.
MixAndMatchCombinationType	Determines how the matching items belonging to the rule can be combined.
PositionType	Type safe, extensible enumeration for RetailTransactionLineItemTypeCodes in a PositionTypePromotionConditionEligibilitySO.
PriceModificationMethod	A code denoting the method of modifying the price that is being applied to the transaction.
PrivilegeType	For mapping privilege type values for manual trigger.

Interface	Description
ReductionMethodMixAndMatch	A code denoting the method of modifying the price that is being applied in case of MixAndMatch promotions.
RoundingMethodCode	Determines how the calculated reduction amount is to be rounded.
RuleControlType	Mapping for PriceDerivationRuleBase-> TransactionControlBreakCode. Determines whether the price derivation rule is to be applied during the registration of the transaction.
RuleType	Defines the type of price derivation rule.
SaleReturnTypeCode	Determines whether the condition can be used only for sales, only for returns, or for both: - 00/null: for sales as well as for returns - 01: only for sales - 02: only for returns.
ThresholdType	A code for the type of threshold which applies to a price derivation rule eligibility.
TriggerValueTypeCode	The type of the TriggerValue for ExternalTriggerValuePromotionConditionEligibilitySO.

5.3.2.1.3 Transaction Model Codes

The package com.gk_software.pricing_engine.model.typecodes.transaction contains typecodes referenced in the transaction data model. Besides the interfaces, you find here the enums with the default values. The String-based generic implementations are defined in com.gk_software.pricing_engine.model.typecodes.transaction_default.

Interface	Description
CalculationMethodCode	A mnemonic code denoting how the price modification calculation was performed.
PriceModificationMethodTx	PriceModification codes for transaction data mapping. Combination of RuleTypeEnum and PriceModificationMethodEnum mapped in RetailTransactionPromotionPriceDerivationRule.getPriceModificationMethodCode( ).
RetailTransactionLineItemTypeCode	Type safe, extensible enumeration for RetailTransactionLineItemTypeCodes.
SaleReturnLineItemActionCode	Type safe, extensible enumeration for SaleReturnLineItemActionCodes.
SalesOrderDeliveryTypeCode	Type safe, extensible enum for mapping the delivery type code of the related sales order SaleReturnLineItemSalesOrder.getSalesOrderDeliveryTypeCode().
UnitOfMeasureCode	Sale return line items quantity input methods.

5.3.2.2 Promotion Master Data

The promotion master data model is used for loading promotion data from the data access layer. It contains the rules and configurations for calculation of promotions and prices.

Package: com.gk_software.pricing_engine.model.md.promotion

Factory Interface: com.gk_software.pricing_engine.model.md.promotion.IPromotionSOFactory

Interface	Description
CombinationPromotionConditionEligibilitySO	Combination of eligibilities.
CouponPromotionConditionEligibilitySO	Eligibility for coupon. If the Promotion.Condition.Eligibility.Coupon.IDList contains more coupon IDs, one eligibility is created for each ID and the eligibilities are connected via a combination eligibility of the type OR.
CustomerGroupPromotionConditionEligibilitySO	Eligibility for customer group.
CustomerPromotionConditionEligibilitySO	Eligibility for customers.
ExternalActionParameterSO	Contains information about parameters for the action to be performed by the client which invoked the Pricing Engine in case that the price derivation rule applies.
ExternalActionPromotionConditionRuleSO	An action is to be performed by the client which invoked the Pricing Engine in case that the price derivation rule applies.
ExternalActionTextSO	Contains information about texts for the action to be performed by the client which invoked the Pricing Engine in case that the price derivation rule applies.
ExternalTriggerValuePromotionConditionEligibilitySO	Eligibility for external trigger.
Get3Pay2PromotionConditionRuleSO	Rule: Get 3 Pay 2. It is not supported with default calculation but may be still used by custom extensions with existing model definition.
ItemPromotionConditionEligibilitySO	Item eligibility.
ManualPromotionConditionEligibilitySO	Manual eligibility promotion condition.
MarketBasketAmountEligibilitySO	Eligibility for market basket amount.
MatchingItemRelationSO	Promotional items.
MatchingItemSO	Matching item.
MatchingMhgRelationSO	Promotional merchandise groups.
MerchandiseSetPromotionConditionEligibilitySO	Defines a merchandise set resulting from the application of set operations to merchandise hierarchy groups and/or items and the further conditions regarding quantities and amounts which govern the application of the associate rule to an item falling under the resulting merchandise set at the time an instance of the item is recorded as a line item in a sale/return.

Interface	Description
MerchandiseSetRelationSO	An association between two merchandise sets that establishes one as the higher and one as the lower level
MerchandiseSetSO	A merchandise set (that is, the reference to the root element of the merchandise set)
MHGPromotionConditionEligibilitySO	Merchandise hierarchy group eligibility. Eligibilities from Promotion.Condition.Eligibility.MerchandiseHierarchyGroupList|MerchandiseHierarchyGroup. Eligibilities from Promotion.Condition.Eligibility.ItemList|Item in case the eligibility contains more than one item. Additionally, if the Promotion.Condition.Eligibility.MerchandiseHierarchyGroup element contains IntersectionList, one MHG eligibility is created for each IntersectionList.ID and the MHG eligibilities are connected via a CombinationEligibility of the type Intersection.
MixAndMatchPromotionConditionRuleSO	Mix and Match.
PositionTypePromotionConditionEligibilitySO	Eligibility for position type.
PromotionConditionEligibilitySO	Promotion condition eligibility.
PromotionConditionRuleSO	Promotion condition rule.
PromotionConditionSO	Condition of the promotion.
PromotionConditionTimeGroupSO	Time restriction.
PromotionMhgFilterSO	This domain object contains the promotion data relevant for the PCE.
PromotionSO	This domain object contains the promotion data relevant for the PCE.
PromotionTextSO	Language-dependent texts to a promotion.
RebatePromotionConditionRuleSO	Simple discount.
ThresholdPromotionConditionEligibility	Threshold promotion condition data for eligibility.
WorkerDiscountGroupPromotionConditionEligibilitySO	Eligibility for workers.

5.3.2.3 Transaction

The transaction data model is the data model for the client data. It is mapped from the incoming request and the results are mapped back.

Each interface defined in the transaction model extends the TxModelIDto interface (com.gk_software.pricing_engine.model.transaction.TxModelIDto), which provides access to the any fields mapped from the request message. This way the custom any fields are available throughout the calculation.

Type	Method	Description
List<Object>	getAny	Extension data provided by request (any objects).

Package: com.gk_software.pricing_engine.model.transaction

Factory Interface: com.gk_software.pricing_engine.model.transaction.ITransactionFactory

Interface	Description
FrequentShopperPointsModifier	A line item modifier that reflects a modification of the frequent shopper points.
LoyaltyRewardLineItem	A RetailTransactionLineItem sub-type that records the crediting of a CustomerLoyaltyAccount with points, or the gifting of an Item or gift certificate to that account because of some Promotion. In the PCE context, at the moment points are used only.
ManualPromotionTrigger	Common interface for manual promotion triggers.
PriceModificationLineItem	A line item component of a RetailTransaction that records the granting of a reduction or increase of price on all Items in the Transaction
RetailPriceModifier	A line item modifier that reflects a modification of the retail selling price. It is provided by PLU through the application of a predefined price derivation rule that depends on parameters provided during the sale transaction. This entity is intended to support pricing schemes which vary depending on the rule applied, Mix and Match etc. The price change reflected in this modifier may be treated as a markdown, a discount, a promotional expense, etc. This entity is also intended to support tiered pricing that is determined by a shopper/customer affiliation with some group like AARP etc. that receives special price breaks.
RetailTransaction	A type of Transaction that records the business conducted between the retail enterprise and another party involving the exchange in ownership and/or accountability for merchandise and/or tender or involving the exchange of tender for services.
RetailTransactionCouponSummary	Information about coupons which were applied for that sales transaction.
RetailTransactionCustomer	Information about the customer the transaction was created for.
RetailTransactionCustomerGroupAssignment	The customer groups which are assigned to the transaction (indirectly by the assigned customers).
RetailTransactionExternalTrigger	Contains the external values which were requested via web services in order to check whether the transaction is eligible for promotions containing eligibilities of type "external trigger value".

Interface	Description
RetailTransactionLineItem	A detail line item of a RetailTransaction that records the business conducted between the retail store and another party involving the exchange in ownership and/or accountability for merchandise and/or tender or involving the exchange of tender for services.
RetailTransactionLineItemAssociation	Associative entity recording relationships between line items within the same RetailTransaction. Examples of usage of this entity include: • Associating an original sales line item to a return line item • Associating an original sales line item to an exchange line item • Associating a sales line item to a fee line item
RetailTransactionModifierCoupon	Information about coupons which were applied for that line item.
RetailTransactionPromotionExternalActionParameter	Contains information about the parameters of the action to be performed by the client which invoked the Pricing Engine in case that the price derivation rule applies.
RetailTransactionPromotionExternalActionText	Contains information about the texts of the action to be performed by the client which invoked the Pricing Engine in case that the price derivation rule applies.
RetailTransactionPromotionPriceDerivationRule	Contains relevant data regarding the price derivation rules which apply to the transaction, respectively, its line items.
RetailTransactionPromotionTrigger	Manual triggers which are created by POS function and used as eligibilities for conditions.
RetailTransactionTotal	A monetary or unit count total for the RetailTransaction. Usually TotalTaxableAmount, TotalTaxCollected, TotalAmountPaid, TotalItemCount, etc.
ReturnLineItem	A sub-type of SaleReturnLineItem recording the reason and disposal of an Item being returned by a customer. If the return is to be recorded against the original retail transaction line item, this should be recorded as a RetailTransactionLineItemAssociation with RetailTransactionLineItemAssociation.AssocationTypeCode of "return".
SaleReturnLineItem	A line item component of a RetailTransaction that records the exchange in ownership of a merchandise item (that is, a sale or return) or the sale or refund related to a service. The sale or refund related to a service captures an item and action taken that reflects an event of interest to the retail business but DOES NOT RESULT in the exchange in ownership of a merchandise item.
SaleReturnLineItemMerchandiseHierarchyGroup	All merchandise hierarchy groups which are directly assigned to the item.
SaleReturnLineItemModifierCoupon	Information about coupons which were applied for that line item.

Interface	Description
SaleReturnLineItemModifierReference	Contains all line items of the current transaction which together caused that a price derivation rule was applied (if the price derivation rule was applied more than one time summarization takes place); is filled only in case that several positions together caused it, is needed for calculating the average rebate in case of item return.
SaleReturnLineItemPromotionTrigger	Manual triggers which are created by POS function and used as eligibilities for conditions.
SaleReturnLineItemSalesOrder	Information about the SAP customer order related to this line item.
Transaction	A record of business activity that involves a financial and/or merchandise unit exchange or the granting of access to conduct business at a specific device, at a specific point in time for a specific employee.
TransactionCategory	Describes which categories transactions fall into.

5.4 Pricing Engine API

5.4.1 Overview

The pricing-engine-api module defines the public API of the PCE core. The API defined in this module is release-stable from PCE 2.9.4 onward. That means extensions based on these APIs should still run on newer version of the PCE core without any migration needed. The PCE defines two levels of API stability: Consumer stability and Extension stability.

> For more detailed description of the defined stability levels, refer to chapter PCE Extensions.

5.4.2 API Overview

This chapter gives an overview on the common API use cases for caller type APIs. Details on the extension points API can be found in the chapter PCE Extensions.

5.4.2.1 General API

5.4.2.1.1 How to create API object instances

Several API objects are instantiated at runtime during processing of a calculation request. As the API is defined as interfaces, there is a factory pattern implemented to create concrete instances of these objects.

HOW IT WORKS

The factory to create API object instances is defined via the Interface com.gk_software.pricing_engine.api.PricingEngineAPIFactory.

An implementation of this factory is initialized via the Spring context. The bean is available to be injected in extension implementations via the following Spring coordinates.

You should use the bean alias to access factory via Spring.

Bean Alias	Default bean name	Implemented interfaces
pricingEngineObjectFactory	defaultPricingEngineObjectFactory	com.gk_software.pricing_engine.api.PricingEngineAPIFactory com.gk_software.pricing_engine.PricingEngineObjectFactory

The factory allows to create the following object instances:

Object to create	Method	Description
Bucket<NormalizedItem>	createNormalizedItemBucket	Creates new normalized item bucket.
EligibilityLoaderContext	createEligibilityLoaderContext	Creates new eligibility loader context object.
EligibilityWrapper<T>	createEligibilityWrapper	Creates new eligibility wrapper object from a PromotionConditionEligibilitySO.
BucketFilter<NormalizedItem>	createFullyPricedNormalizedItemFilter	Creates new fully priced filter for NormalizedItem buckets.
BucketFilter<LineItemSummaryResult>	createFullyPricedLineItemSummaryResultFilter	Creates new fully priced filter for LineItemSummaryResult buckets.
LineItemSummaryResult	createLineItemSummaryResult	Creates new line item summary result object.
Bucket<LineItemSummaryResult>	createLineItemSummaryResultBucket	Creates new line item summary result bucket.
NormalizedItem	createNormalizedItem	Creates new normalized item.
PriceCalculationContext	createPriceCalculationContext	Creates new price calculation context.

Object to create	Method	Description
PriceCalculationResult	createPriceCalculationResult	Creates new price calculation result object.
PromotionConditionWrapper	createPromotionConditionWrapper	Creates new promotion condition wrapper object.
RebateShareResult	createRebateShareResult	Creates new rebate share result.
RuleCalculationContext	createRuleCalculationContext	Creates new rule calculation context.

5.4.2.2 Bucket API

5.4.2.2.1 What are normalized items

A transaction can contain multiple line items. But a line item does not have to represent a single item in the basket. Each line item has a quantity and unit attribute that together define the actual quantity represented. This means a single line item in the transaction or request with multiple quantities is equivalent to a request with multiple items with single quantities. For example, let us assume the following LineItem from a request with a quantity of 3:

```
<LineItem>
  <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
      <ItemID>00000000000000885</ItemID>
      <RegularSalesUnitPrice>1.6</RegularSalesUnitPrice>
      <Quantity Units="1.0" UnitOfMeasureCode="PCE">3</Quantity>
      <TaxIncludedInPriceFlag>false</TaxIncludedInPriceFlag>
      <NonPieceGoodFlag>false</NonPieceGoodFlag>
      <FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
      <PriceTypeCode>00</PriceTypeCode>
      <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
  </Sale>
  <SequenceNumber>0</SequenceNumber>
</LineItem>
```

This is logically the same as a request with three single line items with a quantity of 1 each:

```
<LineItem>
  <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
      <ItemID>00000000000000885</ItemID>
      <RegularSalesUnitPrice>1.6</RegularSalesUnitPrice>
      <Quantity Units="1.0" UnitOfMeasureCode="PCE">1</Quantity>
```

```
      <TaxIncludedInPriceFlag>true</TaxIncludedInPriceFlag>
      <NonPieceGoodFlag>false</NonPieceGoodFlag>
      <FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
      <PriceTypeCode>00</PriceTypeCode>
      <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
  </Sale>
  <SequenceNumber>0</SequenceNumber>
</LineItem>
<LineItem>
  <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
      <ItemID>0000000000000000885</ItemID>
      <RegularSalesUnitPrice>1.6</RegularSalesUnitPrice>
      <Quantity Units="1.0" UnitOfMeasureCode="PCE">1</Quantity>
      <TaxIncludedInPriceFlag>true</TaxIncludedInPriceFlag>
      <NonPieceGoodFlag>false</NonPieceGoodFlag>
      <FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
      <PriceTypeCode>00</PriceTypeCode>
      <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
  </Sale>
  <SequenceNumber>1</SequenceNumber>
</LineItem>
<LineItem>
  <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
      <ItemID>0000000000000000885</ItemID>
      <RegularSalesUnitPrice>1.6</RegularSalesUnitPrice>
      <Quantity Units="1.0" UnitOfMeasureCode="PCE">1</Quantity>
      <TaxIncludedInPriceFlag>true</TaxIncludedInPriceFlag>
      <NonPieceGoodFlag>false</NonPieceGoodFlag>
      <FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
      <PriceTypeCode>00</PriceTypeCode>
      <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
  </Sale>
  <SequenceNumber>3</SequenceNumber>
</LineItem>
```

To make sure the promotion calculation behaves the same, independent of how the items are structured in the transaction, the PCE maps LineItems from the request to an internal item type called NormalizedItem. A NormalizedItem extends the SaleReturnLineItem from the transaction data model. Each NormalizedItem is used to represent a single quantity of an item. So in the examples above, both transaction items would be represented as three NormalizedItem objects for the calculation of the promotions.

HOW IT WORKS

When the items of a transaction are analyzed by the PCE core implementation of the ItemTransactionAnalyzer, during analysis the SaleReturnLineItems in the transaction are transformed to NormalizedItems. These are then stored in the TransactionAnalyzerContext attributes positionPromotionRelevantItems and transactionPromotionRelevantItems. During normalization process, the actual quantities (units times quantity) of the LineItems are analyzed and normalized to one or a fraction of one if the actual quantity is not an integer value.

The NormalizedItem contains calculation-relevant data and a reference via the key to the original LineItem.

THERE IS MORE

The normalization process of a LineItem to NormalizedItems is handled by the internal utility method normalizeLineItem in com.gk_software.pricing_engine.common.utils.BucketUtils.

The NormalizedItems are internally stored in buckets.

5.4.2.2.2 What is a bucket

A Bucket is a PCE-specific implementation of Java Collection Queue API. It supports all common Java Collection and Queue operations and in addition adds additional functionality:

- Buckets can only contain unique elements similar to a Set. This check assumes all elements follow the Java standard equals and hashCode contract.
- Buckets are sortable by adding a comparator.
- Buckets allow applying a Filter.
- Support collection operations for union, difference, and intersection.
- Additional sum calculation on contained elements is possible.

HOW IT WORKS

The Bucket API is defined by com.gk_software.pricing_engine.api.bucket.Bucket Interface.

Type	Method	Description
boolean	addBucket(Bucket<T> otherBucket)	Adds another bucket to this bucket, overtaking calculated values.
Bucket<T>	addFluent(T item)	Same as add but for use in fluent call.
void	addItemValues(T item)	Adds item values to totalized values.
void	applyComparator(Comparator<T> comparator)	Adds new comparator to the bucket.
Bucket<T>	applyFilter(BucketFilter<T> filter)	Applies a filter to the bucket.
List<T>	asList()	Gets bucket items as list.
Comparator<T>	comparator()	Gets the current comparator.
T	getItemByLineItemKey(SaleReturnLineItem.Key lineItemKey)	Returns the item from the bucket identified by the saleReturnLineItem key passed.
List<T>	getItemsByItemKey(RetailTransactionLineItem.Key lineItemKey)	Gets item by RetailTransaction Line Item Key.

Type	Method	Description
BigDecimal	getTriggerAmount()	Gets the sum of all actual amounts inside this bucket.
BigDecimal	getTriggerQuantity()	Gets the sum of all actual quantities inside this bucket.
BigDecimal	getUsedTriggerAmt()	Gets the sum of trigger amounts used in a rebate calculation or ZERO if not applicable.
BigDecimal	getUsedTriggerQut()	Gets the sum of trigger quantities used in a rebate calculation or ZERO if not applicable.
boolean	offer(T o, boolean sort)	Adds the given item to bucket. Allows to control is bucket is immediately resorted.
void	recalculate()	Recalculates all summed up values.
boolean	remove(Object o, boolean sort)	Removes the given item from bucket. Allows to control is bucket is immediately resorted.
void	sort()	Sorts the bucket.
void	subtractItemValues(T item)	Subtracts values for this item from the summed up values.

In addition, a Bucket supports the standard Java API Queue and Collection methods.

Modifier and Type	Method
boolean	add(T e)
boolean	addAll(Collection<? extends T> c)
void	clear()
boolean	contains(Object o)
boolean	containsAll(Collection<?> c)
T	element()
boolean	isEmpty()
Iterator<T>	iterator()
boolean	offer(T e)
T	peek()
T	poll()
T	remove()

Modifier and Type	Method
boolean	remove(Object o)
boolean	removeAll(Collection<?> c)
boolean	retainAll(Collection<?> c)
int	size()
Object[]	toArray()
<T> T[]	toArray(T[] a)

THERE IS MORE

The PCE relies on two different implementations of Buckets: One implementation is based on NormalizedItems as Bucket elements (com.gk_software.pricing_engine.common.impl.bucket.NormalizedItemBucket).
This is mainly used for the normalization process, in the Context and for eligibility loading and evaluation.

Another implementation is based on LineItemSummaryResults as elements. These LineItemSummaryResults are containers for calculation-specific values (see section Result API for details). The LineItemSummaryResultBucket is used in the actual promotion calculation.

You can create both types of Buckets via the PricingEngineAPIFactory.

5.4.2.3 Context API

The PCE context is responsible for storing and exposing all data extracted and analyzed in the phases of a PCE process. The PCE Context helps to carry out this information between processes and the process that uses it to read and update content. The PCE Context is defined in the interface PCE Context (com.gk_software.pricing_engine.api.context.Context). The context is passed as a parameter on all phases of the PCE process.

More specifically, the context stores the following information:

- Transaction analyzer context,
- Eligibility loader context,
- Promotion condition context,
- Plugin registry context,
- Cache context,
- Request context,
- Configuration,

- Transaction and retail transaction line items,
- Position and transaction relevant line items,
- Business unit group,
- Position and transaction summary results,
- Partial results,
- Manual promotions triggers.

HOW IT WORKS

The context is composed by several elements, each responsible for storing a well-defined set of data.

Interface	Description
CacheContext	Local master data cache.
Context	The PCE Context stores all needed data for the processing of the Pricing Engine.
ContextPostProcessor	Any functionality to be processed before the price calculation is executed can be added as ContextPostProcessor plugin implementation.
EligibilityLoaderContext	Stores information for loaded eligibilities.
PromotionConditionContext	Context data on loaded promotions and conditions.
RequestContext<R>	Used to store information regarding the original request message received by the Pricing Engine.
TransactionAnalyzerContext	Analyzed data from transaction preprocessing.
PluginRegistryContext	Contains all registered plugins.

5.4.2.3.1 Reading configuration settings

It is possible to pass a configuration map containing key/value pairs with the request message to the PCE. The configuration parameters are mapped to an internal Configuration object which is then stored in the context making it accessible in all phases of the PCE processing. The PCE Configuration is defined in the interface Configuration (com.gk_software.pricing_engine.api.configuration.Configuration).

HOW IT WORKS

The configured values passed with the PCE request are parsed by the ConfigurationInitializer to the internal Configuration instance (com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer). In case a key value pair was not provided with the request for a given configuration parameter, the default value defined

in the ConfigurationImpl, the PCE core implementation of the Configuration, will be used in the calculation (com.gk_software.pricing_engine.common.impl.configuration.ConfigurationImpl).

For a list of the configuration parameters supported by the Pricing Engine, refer to chapter Configuration.

5.4.2.3.2 Getting transaction information

The TransactionAnalyzerContext stores all data available from analyzing and preprocessing the transaction.

The PCE TransactionAnalyzerContext is defined in the interface TransactionAnalyzerContext (com.gk_software.pricing_engine.api.context.TransactionAnalyzerContext).

HOW IT WORKS

The analysis is carried out by the PCE core plugin implementations of the TransactionAnalyzer interface (com.gk_software.pricing_engine.api.analyzer.TransactionAnalyzer). All analyzer plugin implementations are responsible for extracting and processing a standard transaction part, such as coupons, external promotions or triggers, and are executed in sequence.

The TransactionAnalyzerContext stores the following data:

Data extracted from transaction	PCE core implementation used for analysis (in com.gk_software.pricing_engine.core.analyzer)
Item information for all items	ItemTransactionAnalyzer
The associated merchandise hierarchy group IDs for each merchandise hierarchy group present in the transaction	ItemTransactionAnalyzer
Coupon numbers of coupons present in the transaction	CouponTransactionAnalyzer
Customer group IDs present in the transaction and customer ID	CustomerGroupTransactionAnalyzer
Transaction elements resulting from an external promotion such as price and points modifiers, price derivation rules, price modification, and loyalty reward line items	ExternalPromotionsTransactionAnalyzer
Normalized item buckets filled according to different filtering criteria such as item IDs, merchandise hierarchy groups, position or transaction promotion relevant items	ItemTransactionAnalyzer
Manual triggers	PromotionManualTriggerTransactionAnalyzer

Data extracted from transaction	PCE core implementation used for analysis (in com.gk_software.pricing_engine.core.analyzer)
General transaction related information • Does it include sale and return line items • Transaction basket amount • Discountable items on position and transaction level	ItemTransactionAnalyzer

5.4.2.3.3 Getting specific item buckets from context

The item buckets stored in the TransactionAnalyzerContext are used throughout the PCE process to access the normalized items eligible for the specific calculation use case. The following buckets are available from the context:

- Item buckets stored in a map by item key specific to the given collection. The item key is constructed from the item ID and the unit of measure code.
- Merchandise hierarchy group (mhg) buckets stored in a map by mhg key specific to the given collection. The mhg key is constructed from the mhg ID and the mhg qualifier.
- Merchandise set buckets stored in a map by merchandise set id specific to the given collection.
- Position-related promotion-relevant item bucket.
- Transaction-related promotion-relevant item bucket.

HOW IT WORKS

The buckets are filled separately for each line item present in the transaction when the transaction analysis is performed by the PCE core implementation ItemTransactionAnalyzer after the item normalization was completed.

The transaction-related promotion-relevant item bucket is filled with all normalized items available from the transaction. Any transaction-related promotion calculation process will use this set of items as base for the calculation. The position-related promotion-relevant bucket is filled with all normalized items which meet the criteria set by the internal utility method isPositionPromotionRelevant in com.gk_software.pricing_engine.common.utils.LineItemUtils. This bucket contains all normalized items which can be used in position-related promotion calculation. Any further analysis is done for position discount-related normalized items only.

The analysis continues with processing all items available from the normalization. An ItemInfo (com.gk_software.pricing_engine.common.impl.dto.ItemInfo) object is created and saved to the TransactionAnalyzerContext, as well as the item key and the normalized item to the item bucket map

As a last step, the merchandise hierarchy groups present in the request are processed. For each merchandise hierarchy group, a findAssociatedMhgByChildId call is made to the data access layer retrieving the associated parent merchandise hierarchy groups. The mhg ID and all active, non-expired parent mhg IDs are then saved to the context. The normalized items are added to the mhg buckets under each mhg key.

THERE IS MORE

One normalized item might belong to several buckets. For example, the line item in the request fragment below would belong to the item bucket in the item bucket map stored under the key "000000000000000885__PCE" as well as the merchandise hierarchy group bucket stored in the mhg bucket map under the key "031901__MAIN". In case that "031901__MAIN" had parent merchandise hierarchy groups in the master data structure, the normalized item would be included in the buckets for all the parent mhgs as well.

```
<LineItem>
  <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
      <ItemID>00000000000000885</ItemID>
      <RegularSalesUnitPrice>1.6</RegularSalesUnitPrice>
      <Quantity Units="1.0" UnitOfMeasureCode="PCE">1</Quantity>
      <TaxIncludedInPriceFlag>true</TaxIncludedInPriceFlag>
      <NonPieceGoodFlag>false</NonPieceGoodFlag>
      <FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
      <PriceTypeCode>00</PriceTypeCode>
      <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
  </Sale>
  <MerchandiseHierarchy ID="MAIN">031901</MerchandiseHierarchy>
  <SequenceNumber>0</SequenceNumber>
</LineItem>
```

Thus, the buckets all contain references to the same normalized items, making sure that any modifications to them during the PCE processes are considered in subsequent calculation steps as well.

5.4.2.3.4 Check loaded and activated eligibilities

The EligibilityLoaderContext stores all data available from loading eligibilities from the data access layer.

The PCE EligibilityLoaderContext is defined in the interface EligibilityLoaderContext (com.gk_software.pricing_engine.api.context.EligibilityLoaderContext).

The EligibilityLoaderContext stores eligibility collections for all supported PromotionConditionEligibilitySO implementations that are filled with the data retrieved by the eligibility loaders. The collections contain wrapper objects which include the eligibility and further information regarding the loaded eligibility. All eligibilities received from the data access layer are stored, regardless of their status.

HOW IT WORKS

The eligibility loading and storing in the EligibilityLoaderContext is carried out by the PCE core implementations of the EligibilityLoader interface (com.gk_software.pricing_engine.api.eligibility.EligibilityLoader). All eligibility loader plugin implementations are executed in sequence.

The eligibilities are stored in the EligibilityLoaderContext in the following collections and can be accessed through the getter methods provided by the interface.

Type of collection	Name of collection	Description
List<EligibilityWrapper<CouponPromotionConditionEligibilitySO>>	couponEligibilityList	List of loaded coupon eligibilities.
List<EligibilityWrapper<CustomerGroupPromotionConditionEligibilitySO>>	customerGroupEligibilityList	List of loaded customer group eligibilities.
Map<Long, EligibilityWrapper<ItemPromotionConditionEligibilitySO>>	itemEligibilityMapByItemEligibilityID	Map of loaded item eligibilities stored by eligibility ID.
List<EligibilityWrapper<ManualPromotionConditionEligibilitySO>>	manualPromotionEligibilityListForPositions	List of loaded position related manual trigger eligibilities.
List<EligibilityWrapper<ManualPromotionConditionEligibilitySO>>	manualPromotionEligibilityListForTransaction	List of loaded transaction related manual trigger eligibilities.
List<EligibilityWrapper<MarketBasketAmountEligibilitySO>>	marketBasketAmountEligibilityList	List of loaded market basket amount eligibilities.

Type of collection	Name of collection	Description
Map<String, List<EligibilityWrapper<MerchandiseSetPromotionConditionEligibilitySO>>>	merchandiseSetEligibilitiesByID	Map of loaded merchandise set eligibilities stored by merchandise set ID.
Map<Long, EligibilityWrapper<MHGPromotionConditionEligibilitySO>>	mhgEligibilityMapByMhgEligibilityID	Map of loaded merchandise hierarchy group eligibilities stored by eligibility ID.

How to find activated conditions

The PromotionConditionContext stores all data available from loading promotion conditions, rules, and promotions from the data access layer based on the activated root eligibilities. Besides that, some further information regarding the loaded promotion data is also available from the PromotionConditionContext throughout the promotion calculation.

The PCE PromotionConditionContext is defined in the interface PromotionConditionContext (com.gk_software.pricing_engine.api.context.PromotionConditionContext).

HOW IT WORKS

The following information regarding the loaded promotion data is stored in the PromotionConditionContext in collections and can be accessed through the methods provided by the interface.

Type of collection	Name of collection	Description
Set<Long>	activatedRootEligibilities	List of activated root eligibilities. Further promotion master data is loaded based on this.
SortedMap<Long, SortedMap<Long, Map<String, PromotionConditionWrapper>>>	lineItemConditionTreeMap	Position-related promotion condition maps stored by sequence and resolution.
SortedMap<Long, SortedMap<Long, Map<String, PromotionConditionWrapper>>>	lineItemConditionAtSubtotalTreeMap	Position-related promotion condition at subtotal maps stored by sequence and resolution.
SortedMap<Long, SortedMap<Long, Map<String, PromotionConditionWrapper>>>	transactionConditionTreeMap	Transaction-related promotion condition maps stored by sequence and resolution.

Type of collection	Name of collection	Description
SortedMap<Long, SortedMap<Long, Map<String, PromotionConditionWrapper>>>	transactionConditionAtSubtotalTreeMap	Transaction-related promotion condition at subtotal maps stored by sequence and resolution.
Map<Long, ManualPromotionTrigger>	appliedManualPromotionForRule	Applied manual promotion triggers provided in the transaction stored by rule ID.
Map<Long, List<ManualPromotionTrigger>>	manualPromotionTriggerMapByEligibilityID	Manual promotion triggers provided in the transaction stored by eligibility ID.
Map<Long, List<String>>	matchingItemIDMapByMerchandiseSetID	Mix and match matching items IDs stored by merchandise set IDs.
Map<Long, EligibilityWrapper<?>>	promotionConditionEligibilityMapByID	Eligibilities stored by internal eligibility ID.
Map<Long, List<EligibilityWrapper<?>>>	promotionConditionEligibilityMapByParentID	Eligibilities stored by their parent eligibility ID.
Map<Long, PromotionConditionWrapper>	promotionConditionMapByEligibilityID	Loaded promotion conditions stored by eligibility ID.
Map<Long, PromotionConditionRuleSO>	promotionConditionRuleMapByID	Loaded promotion condition rules stored by internal rule ID.
Map<Long, PromotionSO>	promotionMapByPromotionID	Loaded Promotions stored by promotion ID.
Map<Long, PromotionSO>	adhocPromotionsByPromotionID	Loaded ad hoc promotions stored by promotion ID.
Map<Long, PromotionConditionRuleSO>	virtualDiscounts	Rules providing virtual discounts stored by their internal rule ID.

5.4.2.4 Eligibility API

5.4.2.4.1 Getting calculation specific eligibility information

The eligibilities are loaded from the data access layer for the transaction and are processed by the PCE core plugin implementations of the EligibilityLoader interface (com.gk_software.pricing_engine.api.eligibility.EligibilityLoader). Besides retrieving the master data, the EligibilityLoader is responsible for creating an eligibility-specific bucket to work on during calculation filled with all normalized items affected by the eligibility, for registering the eligibility in the context, for checking and loading of parent eligibilities, and for validating and activating eligibilities. All information resulting from

the eligibility processing is stored in a wrapper object which is stored by the EligibilityLoaderContext as well, making it available throughout the entire PCE process.

The eligibility wrapper is defined in the generic interface EligibilityWrapper (com.gk_software.pricing_engine.api.eligibility.EligibilityWrapper<T extends PromotionConditionEligibilitySO>).

HOW IT WORKS

After the eligibilities have been loaded from the data access layer, the next step of the eligibility loading process is to create a new EligibilityWrapper object for each loaded eligibility. In all further steps of the eligibility loading and the calculation, the wrapper object is used exposing all information available from the eligibility loading process.

The EligibilityWrapper<T extends PromotionConditionEligibilitySO> stores the following information:

Name	Type	Description
Eligibility	T extends PromotionConditionEligibilitySO	The eligibility loaded from the data access layer.
Eligibility status	com.gk_software.pricing_engine.api.eligibility.EligibilityStatus	The status of the eligibility. The status is determined by the eligibility loader process on eligibility activation.
Child count	int	The total count of child eligibilities of a combination eligibility. Relevant in case of combination eligibilities.
Active child count	int	The count of already activated child eligibilities of a combination eligibility. Relevant in case of combination eligibilities.
Combination code	com.gk_software.pricing_engine.model.typecodes.promotion.CombinationCode	The combination code of the eligibility. Relevant in case of combination eligibilities.
Current coupon count	int	The number of used coupons.
Associated retail transaction line item keys	Set<com.gk_software.pricing_engine.model.transaction.RetailTransactionLineItem.Key>	Set of retail transaction line item keys of items related to this eligibility.
Original item bucket	Bucket<NormalizedItem>	Bucket containing all normalized items related to the eligibility.

Name	Type	Description
Working bucket	Bucket<NormalizedItem>	Bucket used during PCE calculation. The working bucket is initialized with the complete set of normalized items related to the eligibility. During calculation, all filtering is done on this bucket, so the original set of normalized items are still available in the original item bucket.
Is manual trigger transaction level	boolean	True if the eligibility is a ManualPromotionConditionEligibilitySO and the manual trigger is transaction-related, false otherwise.
Is ad hoc eligibility	boolean	True if the eligibility was loaded as part of an ad hoc promotion, false otherwise.

5.4.2.5 Result API

5.4.2.5.1 Checking overall calculation result

The overall calculation result is maintained throughout the PCE calculation in the so-called SummaryResult. It contains the total achieved benefit from all applied conditions. The SummaryResult holds the LineItemSummaryResult (com.gk_software.pricing_engine.api.result.LineItemSummaryResult) for each normalized item in a bucket with the individual calculation result values and other related information such as used coupons in the calculation. The SummaryResult also stores similar buckets with the calculation result for each applied sequence, as these may be required as a base of the calculation in some use cases.

The SummaryResult is defined in the interface SummaryResult (com.gk_software.pricing_engine.api.result.SummaryResult).

HOW IT WORKS

The SummaryResult stores the following information regarding the overall calculation result:

Name	Type	Description
Condition applied list	Map<String, Integer>	The number of how many times the given condition was applied stored by promotion condition ID.
LineItemSummaryResult bucket	Bucket<LineItemSummaryResult>	The LineItemSummaryResult bucket holding the overall calculation result.

Name	Type	Description
Sequence line item summaries	Map<Long, Bucket<LineItemSummaryResult>>	The LineItemSummaryResult buckets holding the calculation result for each applied condition sequence stored by sequence.
Customer group factor	BigDecimal	The customer group factor, which is a multiplier for earned points based on the customer group.

The PCE API contains two additional interfaces extending the SummaryResult interface: PositionSummaryResult and TransactionSummaryResult. These API objects are instantiated and initialized as the first step of the PCE condition calculation process, and saved to the Context. Note that the corresponding SummaryResults are only created and saved to the context if position or transaction calculation is deemed required based on the loaded eligibilities. In case of multiple calculated sequences, the summary result must be updated with the result of the single calculation and the updated SummaryResult's LineItemSummaryResultBucket must be saved to the sequence map.

The PositionSummaryResult is defined in the interface PositionSummaryResult (com.gk_software.pricing_engine.api.result.PositionSummaryResult). It extends the Summary result with the following additional information:

- The calculation PartialResults from the current sequence processing. The list is cleared on starting the next sequence calculation.

The TransactionSummaryResult is defined in the interface TransactionSummaryResult (com.gk_software.pricing_engine.api.result.TransactionSummaryResult). It extends the Summary result with the following additional information:

- The sum amount of awarded frequent shopper points.
- The transaction basket amount.
- The actual basket amount to be considered in the calculation.
- The sum awarded rebate amount.
- The updated coupon info to be included in the response.
- Loyalty reward line items created for the transaction.
- Price modification line items created for the transaction.
- The calculation base amount to be used by further conditions stored by the promotion sequence

Working with the result of a single calculation

The result for each condition applied by the PCE is calculated separately from one another. For each individual calculation, a PartialResult is created which then holds the calculation result. After all condition results were calculated for the given promotion sequence and resolution, the SummaryResult is updated with the PartialResults.

The PartialResult is defined in the generic interface PartialResult (com.gk_software.pricing_engine.api.result.PartialResult<S extends SummaryResult>).

HOW IT WORKS

The PartialResult is created and initialized for each condition as a first step of the calculation for both transaction- and position-related promotions. The PartialResult takes over the LineItemSummaryResult bucket from the SummaryResult as the base normalized item bucket to perform the calculation and filters it if necessary. In case of position-related conditions, it is possible that the calculation base is required to be different from the latest overall result, such as the original price for each item, or the result after applying a given promotion sequence. In that case, the LineItemSummaryResultBucket is retrieved from the SummaryResult's sequence line item summaries map.

Throughout the processing of the given condition, the calculation result and any additional information regarding it is stored in the PartialResult. Finally, the SummaryResult is updated with the calculated valid PartialResults. In case there are multiple conditions with the same sequence and resolution and the calculation results in multiple valid PartialResults, the best price calculation is performed before updating the SummaryResult, ensuring that the highest benefit is applied.

The PartialResult interface provides the following information:

- The SummaryResult used as base for the calculation
- The ConditionWrapper of the condition to be applied
- The promotion condition rule
- The ChooseItemMethod to be used in the calculation
- The LineItemSummaryResult bucket to perform the calculation on
- Calculation related flags:
    - isResultValid indicating whether the PartialResult is valid and can be applied
    - isBonusPointsCondition indicating if the condition is bonus points related or not
    - isCashMonetaryDiscountFlag indicating if the calculation result is cache monetary discount or not
- Counters for applying the condition
    - Number of eligibilities applied
    - Number of completed intervals
    - Number of coupons available
- Lists of eligibilities related to the condition
    - Simple item eligibilities and interval item eligibilities
    - Simple mhg eligibilities and interval mhg eligibilities

- Simple merchandise set eligibilities and interval merchandise set eligibilities
- Simple combination eligibilities and interval combination eligibilities
- Coupon eligibilities
- Market basket amount eligibilities

- Rounding methods, rounding denomination, and rounding scale for the following calculations: rebate calculation, rebate share calculation, points amount calculation, awarded points calculation
- Error reason in case of invalid partial result

Understand single item result

While the SummaryResult represents the overall calculation result and the PartialResult holds the calculated result for a single condition, the LineItemSummaryResult stores the calculation result for a single NormalizedItem. Just like NormalizedItems, the LineItemSummaryResults are stored in bucket throughout the calculation.

The LineItemSummaryResult is defined in the interface LineItemSummaryResult (com.gk_software.pricing_engine.api.result.LineItemSummaryResult).

HOW IT WORKS

The LineItemSummaryResult is created on initialization of the SummaryResult. A LineItemSummaryResult is instantiated for every NormalizedItem stored in the TransactionAnalyzerContext's position promotion-relevant items bucket in case of position-related calculation, while in case of transaction-related calculation the transaction promotion relevant items bucket is used.

The LineItemSummaryResult stores the following information regarding the normalized item it represents:

- The NormalizedItem itself
- The RetailTransactionLineItem key for the item
- Flag indication if the position is valid for receiving bonus points or not
- Flags indicating if the position is a valid trigger position and if discounts can be applied on it or not
- The trigger amount and trigger quantity that are available for applying further promotions

The LineItemSummary furthermore stores the following information on the applied condition and trigger amounts and quantities available for it.

Name	Type	Description
New price	BigDecimal	The new price defined in the promotion mater data if there is one
Used count	int	Number of times the item was used in the calculation

Name	Type	Description
Applied count	int	The number of conditions applied on the item
Required count	int	Number of times the item is required to be used in the calculation. Relevant in case of Mix and Match promotion rules.
External promotion count	int	The number of external promotions applied on this item
Awarded points sum	BigDecimal	The total value of awarded points for this item
Trigger quantity	com.gk_software.pricing_engine.api.result.TriggerValues	The quantities of this item available for the calculation. In the TriggerValues class, this value is maintained for rebates and points calculation separately, as well as values considering all previously applied discounts and values not affected by those.
Trigger amount	com.gk_software.pricing_engine.api.result.TriggerValues	The amounts of this item available for the calculation. In the TriggerValues class, this value is maintained for rebates and points calculation separately, as well as values considering all previously applied discounts and values not affected by those.
Lost trigger quantity	com.gk_software.pricing_engine.api.result.TriggerValues	The quantities lost for this calculation that may not be used. In the TriggerValues class, this value is maintained for rebates and points calculation separately, as well as values considering all previously applied discounts and values not affected by those.
Lost trigger amount	com.gk_software.pricing_engine.api.result.TriggerValues	The amounts lost for this calculation that may not be used. In the TriggerValues class, this value is maintained for rebates and points calculation separately, as well as values considering all previously applied discounts and values not affected by those.
Condition values	com.gk_software.pricing_engine.api.result.LineItemSummaryValues	Stores calculation results for the single line item calculation for the overall calculation.
Interval values	com.gk_software.pricing_engine.api.result.LineItemSummaryValues	Stores calculation results for the single line item calculation for interval calculation.
Sequence values	com.gk_software.pricing_engine.api.result.SequenceValues	Calculation values related to the given sequence and resolution
Applied Mix and Match items	List<MatchingItemSO>	The matching items used for applying the condition on this item

Name	Type	Description
Matching item condition values	Map<MatchingItemSO, LineItemSummaryValues>	The partial calculation results for each matching item used for applying the condition on this item. This information is used to create individual retail price modifiers in the response for each matching item used.
Custom result value	<C extends CustomResultValues> C	Custom calculation result values may be added as extension

THERE IS MORE

The LineItemSummaryResult also stores all price and points modifiers as well as updated coupons created or updated as a result of the applied promotions.

5.4.2.6 Rules API

5.4.2.6.1 Getting calculation-specific condition information

The promotion conditions and the related promotion condition rules are loaded from the data access layer based on the activated root eligibilities and processed by the PCE core plugin implementation of the ConditionLoader interface (com.gk_software.pricing_engine.api.engine.ConditionLoader). Besides retrieving the master data, the ConditionLoader is responsible for the preprocessing of the loaded promotion data for the PCE calculation process, such as validation whether all required promotion data was available for the given condition, validation of the condition, or saving the loaded data to the context. All results of the condition loading and processing are stored in the context and a designated wrapper object, similar to the EligibilityWrapper.

The wrapper class for the condition is defined in the interface PromotionConditionWrapper (com.gk_software.pricing_engine.api.rules.PromotionConditionWrapper) and it stores the loaded condition and additional information.

HOW IT WORKS

The ConditionLoader loads the related promotion conditions and rules for each activated root eligibility found in the context. A PromotionConditionWrapper is instantiated for each and it is initialized with the promotion data loaded. The PromotionConditionWrappers are stored in the PromotionConditionContext.

The PromotionConditionWrapper stores the following information:

Name	Type	Description
Promotion condition	com.gk_software.pricing_engine.model.md.promotion.PromotionConditionSO	The promotion condition.
Sequence	Long	Calculated sequence value to be used in the calculation. It adds up of the condition's sequence and the related manual triggers sequence added, if there is one.
Condition status	int	The status of the condition determined on condition loading.
Is overridden flag	boolean	Indicates whether - in case of a manual rule - the values should be taken from the request transaction or not.
Manual trigger addend	Long	The manual trigger sequence added or null if the condition does not relate to a manual eligibility.
Calculation rule processor	com.gk_software.pricing_engine.api.rules.CalculationRuleProcessor	The processor responsible for executing rule-specific calculations.
Rebate distribution processor	com.gk_software.pricing_engine.api.rules.RebateDistributionProcessor	The rule-specific processor responsible for rebate distribution between a set of items.

5.5 Pricing Engine Common

The pricing-engine-common module contains the internal PCE API. This is not part of the public API, therefore it is not guaranteed to be release-stable and classes and interfaces may change without notice.

That being said, it is not expected to often change and will be kept stable as much as possible. The use of this API is possible for extensions, but you should expect some migration effort on new versions of the PCE.

The module contains the common implementation for some API interfaces and basic test framework and utility classes.

Important internal API interfaces and classes are:

- Bucket and NormalizedItem core implementations.
- Common implementations for context and configuration related interfaces.
- PricingEngine and PricingEngineRequest for internal embedding.
- PricingEngineObjectFactory for PCE calculation related object creation.

- DataAccessService, the internal data access interface.
- PromotionCalculator, the main entry point to promotion calculation.
- CalculatorService interface for calculating, distributing, and validating promotions.
- ConflictHandler interface for resolving possible conflicts between promotion conditions.
- TransactionUpdateHandler interface for updating transaction with calculated data.
- Classes for business error handling.
- Comparator and utility classes used by the calculation.

HOW IT WORKS

There are several internal APIs provided by the common module.

Package: com.gk_software.pricing_engine

Interface	Description
PricingEngine	Main entry point to promotion calculation.
PricingEngineFactory	Factory for creating and initializing PricingEngine instances and PricingEngineRequest and response objects.
PricingEngineObjectFactory	This object contains factory methods for calculation-related PricingEngine objects.
PricingEngineRequest	Internal Pricing Engine request object.
PricingEngineResponse	Internal Pricing engine response object.
ResolutionComparator	Resolution comparator for collision handling.

Package: com.gk_software.pricing_engine.common.engine

Interface	Description
ConflictHandler	Resolves possible conflicts between promotion conditions in case multiple valid partial results are available after calculation.
PromotionCalculator	Promotion calculator.

Package: com.gk_software.pricing_engine.common.data_access

Interface	Description
DataAccessService	Data access service interface for integration of master data loading.

Package: com.gk_software.pricing_engine.common.calculator_service

Interface	Description
CalculatorService<S extends SummaryResult>	Service for calculating, distributing, and validating promotions.
TransactionCalculationService	Service to perform transaction related calculations.

Package: com.gk_software.pricing_engine.common.eligibility

Interface	Description
AdhocEligibilityProcessor	Processes eligibilities loaded with ad hoc promotions.
EligibilityService	Eligibility related operations service, responsible for providing additional eligibility specific information for promotion calculation of a single partial result.
ThresholdEligibilityProcessor	Process to handle all registered ThresholdActivationCheck instances.

Package: com.gk_software.pricing_engine.common.exceptions

Interface	Description
BusinessErrorCode	Interface to define business errors by specific codes

Package: com.gk_software.pricing_engine.common.result

Interface	Description
PartialResultCalculator	Partial result calculation related service.
ResultCustomizerService	Helper service to handle ResultDtoCustomizer plugins.
SummaryResultHandler<S extends SummaryResult>	Supports updating summary results with single calculation partial results.

Package: com.gk_software.pricing_engine.common.transaction

Interface	Description
TransactionDomainObjectService	Service for creating and handling transaction DO entities.
TransactionMapper	Map internal calculation results to transaction data objects...
TransactionUpdateHandler	Transaction update handler, responsible for creating transaction model objects based on the calculation result and updating the transaction for the response.

Package: com.gk_software.pricing_engine.common.price_modification

Interface	Description
PriceModificationEngine	Price or points discount value calculation service.

5.6 Pricing Engine Core Implementation

5.6.1 Overview

The pricing-engine-impl module contains the bulk of the implemented logic of the PCE core. The module contains all classes below the package com.gk_software.pricing_engine.core and the main Spring context definitions.

5.6.2 Spring Context

This module contains the majority of the Spring bean definitions of the PCE core.

There is one main XML that provides access to the defined beans: META-INF/com.gk-software.pricing-engine/psi-sap/common.xml

Internally, the bean definitions are structured in several smaller context definition files:

- META-INF/com.gk-software.pricing-engine/core/default-beans.xml: All standard beans are defined in this context definition file.
- META-INF/com.gk-software.pricing-engine/core/beans.xml: Helper context definition file to integrate default beans and import default plugins.
- META-INF/com.gk-software.pricing-engine/core/conditions/default-plugin-pce.xml: All condition loading-related default plugins are defined in this context definition file.
- META-INF/com.gk-software.pricing-engine/core/eligibilities/default-plugin-pce.xml: All eligibility processing-related default plugins are defined in this context definition file.
- META-INF/com.gk-software.pricing-engine/core/filters/default-plugin-pce.xml: All default item filter plugins are defined in this context definition file.
- META-INF/com.gk-software.pricing-engine/core/price_modifications/default-plugin-pce.xml: All price modification related default plugins are defined in this context definition file.
- META-INF/com.gk-software.pricing-engine/core/rules/default-plugin-pce.xml: All rule processing related default plugins are defined in this context definition file.

- META-INF/com.gk-software.pricing-engine/core/transaction_analyzers/default-plugin-pce.xml: All default request transaction analyzer plugins are defined in this context definition file.
- META-INF/com.gk-software.pricing-engine/core/validators/default-plugin-pce.xml: All validation related default plugins are defined in this context definition file.

5.6.2.1 Default beans

Inside the bean context, the standard beans providing the processing framework of the PCE core are defined.

Bean Name	Alias	Bean Class	Description
defaultPriceModificationLookup	priceModificationLookup	com.gk_software.pricing_engine.core.price_modification.PriceModificationLookupImpl	Handler for lookup of the priceModification related to a PromotionConditionRuleSO checking for PriceModificationMethod or getting the PriceModification for a given ReductionMethodMixAndMatch.
defaultPriceModificationEngine	priceModificationEngine	com.gk_software.pricing_engine.core.price_modification.PriceModificationEngineImpl	Manages calculation logic regarding specific price modification during rule processing.
defaultPricingEngineObjectFactory	pricingEngineObjectFactory	com.gk_software.pricing_engine.common.impl.PricingEngineDefaultObjectFactory	Default implementation of the PricingEngineAPIFactory and PricingEngineObjectFactory interfaces.
defaultResultCustomizerService	resultCustomizerService	com.gk_software.pricing_engine.common.impl.result.ResultCustomizerServiceImp	Helper service to handle ResultDtoCustomizer plugins.
defaultPromotionSOFactory	promotionSOFactory	com.gk_software.pricing_engine.model_default.PromotionSODefaultFactory	The default model factory for promotion master data DOs (data objects).
defaultTransactionFactory	transactionFactory	com.gk_software.pricing_engine.model_default.TransactionDefaultFactory	The default model factory for promotion transaction DOs (data objects).

Bean Name	Alias	Bean Class	Description
defaultPricingEngineFactory	pricingEngineFactory	com.gk_software.pricing_engine.core.engine.PricingEngineDefaultFactory	Factory for creation of PricingEngine and PricingEngineRequest instances.
defaultContextProviderFactory	contextProviderFactory	com.gk_software.pricing_engine.core.engine.SimpleContextProviderFactory	The ContextProvider manages the context in a PricingEngine instance. This factory is called during initialization of a PricingEngine instance to get the corresponding ContextProvider.
defaultConfigurationInitializer	configurationInitializer	com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer	Parser for converting a configuration property map to the internal Configuration instance.
defaultTransactionUpdateHandler	transactionUpdateHandler	com.gk_software.pricing_engine.core.transaction.TransactionUpdateHandlerImpl	Central entry point for manipulating the transaction DO. Handles update of transaction with calculated data and cleaning data from request.
defaultTransactionMapper	transactionMapper	com.gk_software.pricing_engine.core.transaction.TransactionMapperImpl	Responsible for mapping internal data objects to the transaction DO objects.
defaultTransactionSplitter	transactionSplitter	com.gk_software.pricing_engine.common.utils.TransactionSplitter	Utility component for Pricing Engine Calculation Mode to split a request in many request with one line item each.
defaultAdhocEligibilityProcessor	adhocEligibilityProcessor	com.gk_software.pricing_engine.core.eligibility.AdhocEligibilityProcessorImpl	Processes eligibilities loaded with ad hoc promotions.
defaultConditionLoader	conditionLoader	com.gk_software.pricing_engine.core.engine.ConditionLoaderImpl	Loading and validation of Conditions and Promotions.
defaultAdhocPromotionHandler	adhocPromotionHandler	com.gk_software.pricing_engine.core.engine.AdhocPromotionHandlerImpl	Loads and processes ad hoc promotions and stores the result in the context.

Bean Name	Alias	Bean Class	Description
defaultPromotionCalculator	promotionCalculator	com.gk_software.pricing_engine.core.engine.PromotionCalculatorImpl	Central entry point to promotion calculation of loaded Conditions.
defaultTransactionCalculator	transactionCalculator	com.gk_software.pricing_engine.common.impl.result.calculator.TransactionCalculator	Utility for different calculation steps during condition processing.
defaultConflictHandler	conflictHandler	com.gk_software.pricing_engine.core.engine.DefaultConflictHandler	Handler for collisions and best price calculation.
defaultPartialResultCalculator	partialResultCalculator	com.gk_software.pricing_engine.core.engine.PartialResultCalculatorImpl	Calculates the result of a single condition.
defaultPositionSummaryResultHandler	positionSummaryResultHandler	com.gk_software.pricing_engine.core.result.PositionSummaryResultHandler	Applies a single position-related partial result to the overall position summary result.
defaultTransactionSummaryResultHandler	transactionSummaryResultHandler	com.gk_software.pricing_engine.core.result.TransactionSummaryResultHandler	Applies a single transaction-related partial result to the overall transaction summary result.
defaultCalculatorPositionPartialService	calculatorPositionPartialService	com.gk_software.pricing_engine.core.calculator_service.CalculatorPositionPartialService	Calculates the result of a single position-related condition. Used by PartialResultCalculator.
defaultCalculatorTransactionPartialService	calculatorTransactionPartialService	com.gk_software.pricing_engine.core.calculator_service.CalculatorTransactionPartialService	Calculates the result of a single transaction-related condition. Used by PartialResultCalculator.
defaultEligibilityActivator	eligibilityActivator	com.gk_software.pricing_engine.core.eligibility.EligibilityActivatorImpl	The EligibilityActivator handles the activation of a valid eligibility. It checks the parent eligibilities and tries to activate them accordingly. It sets the EligibiltiyWrapper status to ACTIVE and updates the context There is one EligibilityActivator in the application context.

Bean Name	Alias	Bean Class	Description
defaultEligibilityRegistry	eligibilityRegistry	com.gk_software.pricing_engine.core.eligibility.EligibilityRegistryImpl	The eligibility registry is responsible to validate and register the eligibilities to the context. It will not do additional activation checks.
defaultThresholdTypeEligibilityProcessor	thresholdTypeEligibilityProcessor	com.gk_software.pricing_engine.core.eligibility.ThresholdEligibilityProcessorImpl	Process to handle all registered ThresholdActivationCheck instances. It receives the right instance for a given ThresholdType and checks ThresholdPromotionConditionEligibility for activation.
defaultCalculationBaseService	calculationBaseService	com.gk_software.pricing_engine.core.calculation.services.CalculationBaseService	Service to encapsulate logic regarding to calculation base

5.6.2.2 Plugin Lists

Besides standard Spring beans, the Spring context defines the lists of collected plugins. These are provided via Spring context as beans of Java List type. The content of the list is automatically collected from all beans in the application context that implement the corresponding plugin interface.

Bean Name	Alias	Plugin Interface
allPCEPlugins	-	com.gk_software.pricing_engine.api.PCEPlugin
defaultEligibilityLoaderList	eligibilityLoaderList	com.gk_software.pricing_engine.api.eligibility.EligibilityLoader
defaultEligibilityLoaderFragments	eligibilityLoaderFragments	com.gk_software.pricing_engine.api.eligibility.EligibilityLoaderFragment
defaultEligibilityRuleHandlerFragments	eligibilityRuleHandlerFragments	com.gk_software.pricing_engine.api.rules.EligibilityRuleHandlerFragment
defaultEligibilityValidators	eligibilityValidators	com.gk_software.pricing_engine.api.eligibility.EligibilityValidator
defaultThresholdTypeActivators	thresholdTypeActivators	com.gk_software.pricing_engine.api.eligibility.ThresholdActivationCheck

Bean Name	Alias	Plugin Interface
defaultEligibilityParsers	eligibilityParsers	com.gk_software.pricing_engine.api.eligibility.EligibilityResultParser
defaultRebatableItemFilterPlugins	rebatableItemFilterPlugins	com.gk_software.pricing_engine.api.filter.plugin.RebatableItemFilterPlugin
defaultPriceModifications	priceModifications	com.gk_software.pricing_engine.api.price_modification.PriceModification
defaultRuleLoaderList	ruleLoaderList	com.gk_software.pricing_engine.api.rules.RuleLoader
defaultEligibilityRuleHandler	eligibilityRuleHandlers	com.gk_software.pricing_engine.api.rules.EligibilityRuleHandler
defaultTransactionAnalyzerList	transactionAnalyzerList	com.gk_software.pricing_engine.api.analyzer.TransactionAnalyzer
defaultContextPostProcessorList	contextPostProcessorList	com.gk_software.pricing_engine.api.context.ContextPostProcessor
defaultRebateDistributionProcessors	rebateDistributionProcessors	com.gk_software.pricing_engine.api.rules.RebateDistributionProcessor
defaultConditionValidatorList	conditionValidatorList	com.gk_software.pricing_engine.api.calculation.ConditionValidator
defaultMixAndMatchProrationCustomizers	mixAndMatchProrationCustomizers	com.gk_software.pricing_engine.api.rules.MixAndMatchProrationCustomizer
defaultResultValidatorList	resultValidatorList	com.gk_software.pricing_engine.api.result.ResultValidator
defaultCalculationRuleProcessors	calculationRuleProcessors	com.gk_software.pricing_engine.api.rules.CalculationRuleProcessor
defaultExclusiveCheckHandler	exclusiveCheckHandler	com.gk_software.pricing_engine.api.calculation.ExclusiveCheckHandler
defaultRebateValidatorList	rebateValidatorList	com.gk_software.pricing_engine.api.result.RebateValidator
defaultBestPriceCalculators	bestPriceCalculators	com.gk_software.pricing_engine.api.calculation.BestPriceCalculator
defaultRetailPriceModifierCustomizer	retailPriceModifierCustomizer	com.gk_software.pricing_engine.api.result.customizer.RetailPriceModifierCustomizer
defaultFrequentShopperPointsModifierCustomizer	frequentShopperPointsModifierCustomizer	com.gk_software.pricing_engine.api.result.customizer.FrequentShopperPointsModifierCustomizer
defaultLoyaltyRewardLineItemCustomizer	loyaltyRewardLineItemCustomizer	com.gk_software.pricing_engine.api.result.customizer.LoyaltyRewardLineItemCustomizer
defaultPriceModificationLineItemCustomizer	priceModificationLineItemCustomizer	com.gk_software.pricing_engine.api.result.customizer.PriceModificationLineItemCustomizer

Bean Name	Alias	Plugin Interface
defaultRetailTransactionPromotionPriceDerivationRuleCustomizer	retailTransactionPromotionPriceDerivationRuleCustomizer	com.gk_software.pricing_engine.api.result.customizer.RetailTransactionPromotionPriceDerivationRuleCustomizer
defaultAdhocPromotionLoaders	adhocPromotionLoaders	com.gk_software.pricing_engine.api.engine.AdhocPromotionLoader
defaultConditionLoaderFragments	conditionLoaderFragments	com.gk_software.pricing_engine.api.engine.ConditionLoaderFragment

5.6.2.3 Core Plugins

The following core plugin implementations are registered to the application context and available through the plugins listed above.

Bean Name	Alias	Plugin List Alias	Bean Class
defaultConditionLoaderFragment	conditionLoaderFragment	conditionLoaderFragments	com.gk_software.pricing_engine.core.engine.DefaultConditionLoaderFragment
defaultAdhocConditionLoaderFragment	adhocConditionLoaderFragment	conditionLoaderFragments	com.gk_software.pricing_engine.core.engine.AdhocConditionLoaderFragment
defaultEligibilityLoader	defaultEligibilityLoader	eligibilityLoaderList	com.gk_software.pricing_engine.core.eligibility.DefaultEligibilityLoader
defaultManualTriggerEligibilityLoader	manualTriggerEligibilityLoader	eligibilityLoaderList	com.gk_software.pricing_engine.core.eligibility.ManualTriggerEligibilityLoader
defaultCouponEligibilityLoader	couponEligibilityLoader	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.CouponEligibilityLoader
defaultItemEligibilityLoader	itemEligibilityLoader	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.ItemEligibilityLoader
defaultMerchandiseGroupEligibilityLoader	merchandiseGroupEligibilityLoader	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.MerchandiseGroupEligibilityLoader
defaultMerchandiseSetEligibilityLoader	merchandiseSetEligibilityLoader	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.MerchandiseSetEligibilityLoader
defaultMarketBasketAmountEligibilityLoader	marketBasketAmountEligibilityLoader	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.MarketBasketAmountEligibilityLoader
defaultCustomerGroupEligibilityLoader	customerGroupEligibilityLoader	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.CustomerGroupEligibilityLoader
defaultAdhocEligibilityLoaderFragment	adhocEligibilityLoaderFragment	eligibilityLoaderFragments	com.gk_software.pricing_engine.core.eligibility.fragments.AdhocEligibilityLoader

Bean Name	Alias	Plugin List Alias	Bean Class
defaultThresholdActivatorAMQU	thresholdActivatorAMQU	thresholdTypeActivators	com.gk_software.pricing_engine.core.eligibility.threshold.ThresholdActivatonCheckAMQU
defaultThresholdActivatorAMT	thresholdActivatorAMT	thresholdTypeActivators	com.gk_software.pricing_engine.core.eligibility.threshold.ThresholdActivatonCheckAMT
defaultThresholdActivatorQUT	thresholdActivatorQUT	thresholdTypeActivators	com.gk_software.pricing_engine.core.eligibility.threshold.ThresholdActivationCheckQUT
defaultThresholdActivatorCOMB	thresholdActivatorCOMB	thresholdTypeActivators	com.gk_software.pricing_engine.core.eligibility.threshold.ThresholdActivationCheckCOMB
defaultCouponEligibilityParser	couponEligibilityParser	eligibilityParsers	com.gk_software.pricing_engine.core.eligibility.parser.CouponEligibilityParser
defaultTotalEligibilityParser	totalEligibilityParser	eligibilityParsers	com.gk_software.pricing_engine.core.eligibility.parser.TotalEligibilityParser
defaultItemEligibilityParser	itemEligibilityParser	eligibilityParsers	com.gk_software.pricing_engine.core.eligibility.parser.ItemEligibilityParser
defaultMhgEligibilityParser	mhgEligibilityParser	eligibilityParsers	com.gk_software.pricing_engine.core.eligibility.parser.MhgEligibilityParser
defaultCombinationEligibilityParser	combinationEligibilityParser	eligibilityParsers	com.gk_software.pricing_engine.core.eligibility.parser.CombinationEligibilityParser
defaultMerchandiseSetEligibilityParser	merchandiseSetEligibilityParser	eligibilityParsers	com.gk_software.pricing_engine.core.eligibility.parser.MerchandiseSetEligibilityParser
defaultItemEligibilityRuleHandlerFragment	itemEligibilityRuleHandlerFragment	eligibilityRuleHandlerFragments	com.gk_software.pricing_engine.core.rules.handler.fragments.ItemEligibilityRuleHandlerFragment
defaultMerchandiseSetEligibilityRuleHandlerFragment	merchandiseSetEligibilityRuleHandlerFragment	eligibilityRuleHandlerFragments	com.gk_software.pricing_engine.core.rules.handler.fragments.MerchandiseSetEligibilityRuleHandlerFragment
defaultMhgEligibilityRuleHandlerFragment	mhgEligibilityRuleHandlerFragment	eligibilityRuleHandlerFragments	com.gk_software.pricing_engine.core.rules.handler.fragments.MhgEligibilityRuleHandlerFragment
defaultCombinationEligibilityRuleHandlerFragment	combinationEligibilityRuleHandlerFragment	eligibilityRuleHandlerFragments	com.gk_software.pricing_engine.core.rules.handler.fragments.CombinationEligibilityRuleHandlerFragment
defaultSaleReturnTypeItemFilter	defaultSaleReturnTypeItemFilter	rebatableItemFilterPlugins	com.gk_software.pricing_engine.common.impl.bucket.filter.SaleReturnTypeItemFilter
absoluteDiscountPriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.AbsoluteDiscountPriceModification
newPricePriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.NewPricePriceModification

Bean Name	Alias	Plugin List Alias	Bean Class
newTotalPricePriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.NewTotalPricePriceModification
newSetPricePriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.NewSetPricePriceModification
totalAbsoluteDiscountPriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.TotalAbsoluteDiscountPriceModification
totalPercentageDiscountPriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.TotalPercentageDiscountPriceModification
totalPercentageDiscountRoundedTwicePriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.TotalPercentageDiscountRoundedTwicePriceModification
percentageDiscountPriceModification	-	priceModifications	com.gk_software.pricing_engine.core.price_modification.PercentageDiscountPriceModification
defaultEligibilityRuleHandler	defaultEligibilityRuleHandler	eligibilityRuleHandlers	com.gk_software.pricing_engine.core.rules.handler.DefaultEligibilityRuleHandler
defaultStandardRuleLoader	standardRuleLoader	ruleLoaderList	com.gk_software.pricing_engine.core.rules.loader.StandardPriceRuleLoader
defaultExternalActionRuleLoader	externalActionRuleLoader	ruleLoaderList	com.gk_software.pricing_engine.core.rules.loader.ExternalActionRuleLoader
defaultMixAndMatchRuleLoader	mixAndMatchRuleLoader	ruleLoaderList	com.gk_software.pricing_engine.core.rules.loader.MixAndMatchRuleLoader
defaultRebateRuleLoader	rebateRuleLoader	ruleLoaderList	com.gk_software.pricing_engine.core.rules.loader.RebateRuleLoader
defaultExternalActionCalculationRule	externalActionCalculationRule	calculationRuleProcessors	com.gk_software.pricing_engine.core.rules.ExternalActionCalculationRule
defaultManualPriceCalculationRule	manualPriceCalculationRule	calculationRuleProcessors	com.gk_software.pricing_engine.core.rules.ManualPriceCalculationRule
defaultMixAndMatchCalculationRule	mixAndMatchCalculationRule	calculationRuleProcessors	com.gk_software.pricing_engine.core.rules.MixAndMatchCalculationRule
defaultNoDiscountCalculationRule	noDiscountCalculationRule	calculationRuleProcessors	com.gk_software.pricing_engine.core.rules.NoDiscountCalculationRule
defaultSimpleDiscountCalculationRule	simpleDiscountCalculationRule	calculationRuleProcessors	com.gk_software.pricing_engine.core.rules.SimpleDiscountCalculationRule
defaultExternalActionRebateDistributionService	externalActionRebateDistributionService	rebateDistributionProcessors	com.gk_software.pricing_engine.core.rules.proration.ExternalActionRebateDistributionService
defaultManualPriceRebateDistributionService	manualPriceRebateDistributionService	rebateDistributionProcessors	com.gk_software.pricing_engine.core.rules.proration.ManualPriceRebateDistributionService

Bean Name	Alias	Plugin List Alias	Bean Class
defaultMixAndMatchRebateDistributionService	mixAndMatchRebateDistributionService	rebateDistributionProcessors	com.gk_software.pricing_engine.core.rules.proration.MixAndMatchRebateDistributionService
defaultNoDiscountRebateDistributionService	noDiscountRebateDistributionService	rebateDistributionProcessors	com.gk_software.pricing_engine.core.rules.proration.NoDiscountRebateDistributionService
defaultSimpleDiscountRebateDistributionService	simpleDiscountRebateDistributionService	rebateDistributionProcessors	com.gk_software.pricing_engine.core.rules.proration.SimpleDiscountRebateDistributionService
defaultBestPriceCalculator	bestPriceCalculator	bestPriceCalculators	com.gk_software.pricing_engine.core.calculation.DefaultBestPriceCalculator
defaultItemTransactionAnalyzer	itemTransactionAnalyzer	transactionAnalyzerList	com.gk_software.pricing_engine.core.analyzer.ItemTransactionAnalyzer
defaultExternalPromotionsTransactionAnalyzer	externalPromotionsTransactionAnalyzer	transactionAnalyzerList	com.gk_software.pricing_engine.core.analyzer.ExternalPromotionsTransactionAnalyzer
defaultCouponTransactionAnalyzer	couponTransactionAnalyzer	transactionAnalyzerList	com.gk_software.pricing_engine.core.analyzer.CouponTransactionAnalyzer
defaultCustomerGroupTransactionAnalyzer	customerGroupTransactionAnalyzer	transactionAnalyzerList	com.gk_software.pricing_engine.core.analyzer.CustomerGroupTransactionAnalyzer
defaultPromotionManualTriggerAnalyzer	promotionManualTriggerAnalyzer	transactionAnalyzerList	com.gk_software.pricing_engine.core.analyzer.PromotionManualTriggerTransactionAnalyzer
defaultResultCouponCountValidator	resultCouponCountValidator	resultValidatorList	com.gk_software.pricing_engine.core.result.validator.ResultCouponCountValidator
defaultResultActualPurchaseBaseValidator	resultActualPurchaseBaseValidator	resultValidatorList	com.gk_software.pricing_engine.core.result.validator.ResultActualPurchaseBaseValidator
defaultPositionResultInfluencedPositionValidator	positionResultInfluencedPositionValidator	resultValidatorList	com.gk_software.pricing_engine.core.result.validator.PositionResultInfluencedPositionValidator
defaultTransactionResultInfluencedPositionValidator	transactionResultInfluencedPositionValidator	resultValidatorList	com.gk_software.pricing_engine.core.result.validator.TransactionResultInfluencedPositionValidator
defaultPositionResultConditionAmountValidator	positionResultConditionAmountValidator	resultValidatorList	com.gk_software.pricing_engine.core.result.validator.PositionResultConditionAmountValidator
defaultTransactionResultConditionAmountValidator	transactionResultConditionAmountValidator	resultValidatorList	com.gk_software.pricing_engine.core.result.validator.TransactionResultConditionAmountValidator
defaultConditionSequenceAllowedValidator	conditionSequenceAllowedValidator	conditionValidatorList	com.gk_software.pricing_engine.core.calculation.validator.ConditionSequenceAllowedValidator
defaultExclusivePromotionCombinationValidator	exclusivePromotionCombinationValidator	conditionValidatorList	com.gk_software.pricing_engine.common.validator.ExclusivePromotionCombinationValidator
defaultSaleReturnTypeConditionValidator	saleReturnTypeConditionValidator	conditionValidatorList	com.gk_software.pricing_engine.core.calculation.validator.SaleReturnTypeConditionValidator

Bean Name	Alias	Plugin List Alias	Bean Class
defaultEligibilityExpirationValidator	eligibilityExpirationValidator	eligibilityValidators	com.gk_software.pricing_engine.core.eligibility.validator.EligibilityExpirationValidator
defaultEligibilityStatusValidator	eligibilityStatusValidator	eligibilityValidators	com.gk_software.pricing_engine.core.eligibility.validator.EligibilityStatusValidator
defaultMarketBasketAmountEligibilityValidator	marketBasketAmountEligibilityValidator	eligibilityValidators	com.gk_software.pricing_engine.core.eligibility.validator.MarketBasketAmountEligibilityValidator
defaultAdhocEligibilityFulfillmentValidator	adhocEligibilityFulfillmentValidator	eligibilityValidators	com.gk_software.pricing_engine.core.eligibility.validator.AdhocEligibilityFulfillmentValidator
defaultExclusiveCheckHandlerValidator	exclusiveCheckHandlerValidator	exclusiveCheckHandler	com.gk_software.pricing_engine.common.validator.DefaultExclusiveCheckHandler

5.7 Pricing Engine Promotion Service Impl

The modules provide the Implementation of PromotionPricingService interface of the Client API. It provides the main integration point in the PPS environment.

The module handles all mapping from Client API to PCE transaction model. During this mapping, the Client API compatibility checks and validations are done. In the psi-sap module, the Spring bean context for the service integration is defined.

The module provides three important parts:

- The class PromotionPricingServiceImpl is the implementation of com.sap.ppengine.client.service.PromotionPricingService and the main entry point for SAP integration.
- DataMappingRequestMapper: The classes in package com.gk_software.pricing_engine.psi.sapmapping.request are responsible for mapping and validating the client API request to the PCE model.
- DataMappingResponseMapper: The classes in package com.gk_software.pricing_engine.psi.sapmapping.response are responsible for mapping PCE model to client API response.

HOW IT WORKS

The general call sequence for Client API request processing follows these steps:

1. The PromotionPricingServiceImpl.calculate method is called with request and configuration map information.
2. The configuration is parsed by the ConfigurationInitializer.
3. The DataMappingRequestMapper is called to map the Client API request to the PCE transaction model. During this mapping, the API compatibility and validity is checked.
4. The DataMappingRequestMapper delegates the calls to several child classes that are responsible for mapping specific elements of the request.
5. The parsed transaction is passed to the internal PricingService that creates a PricingEngine instance and processes the calculation request.
6. The returned transaction is then mapped by the DataResponseMapper back to a Client API response.
7. Again, the DataResponseMapper delegates the mapping to child classes for specific elements.
8. The mapped response is returned from the calculate method.

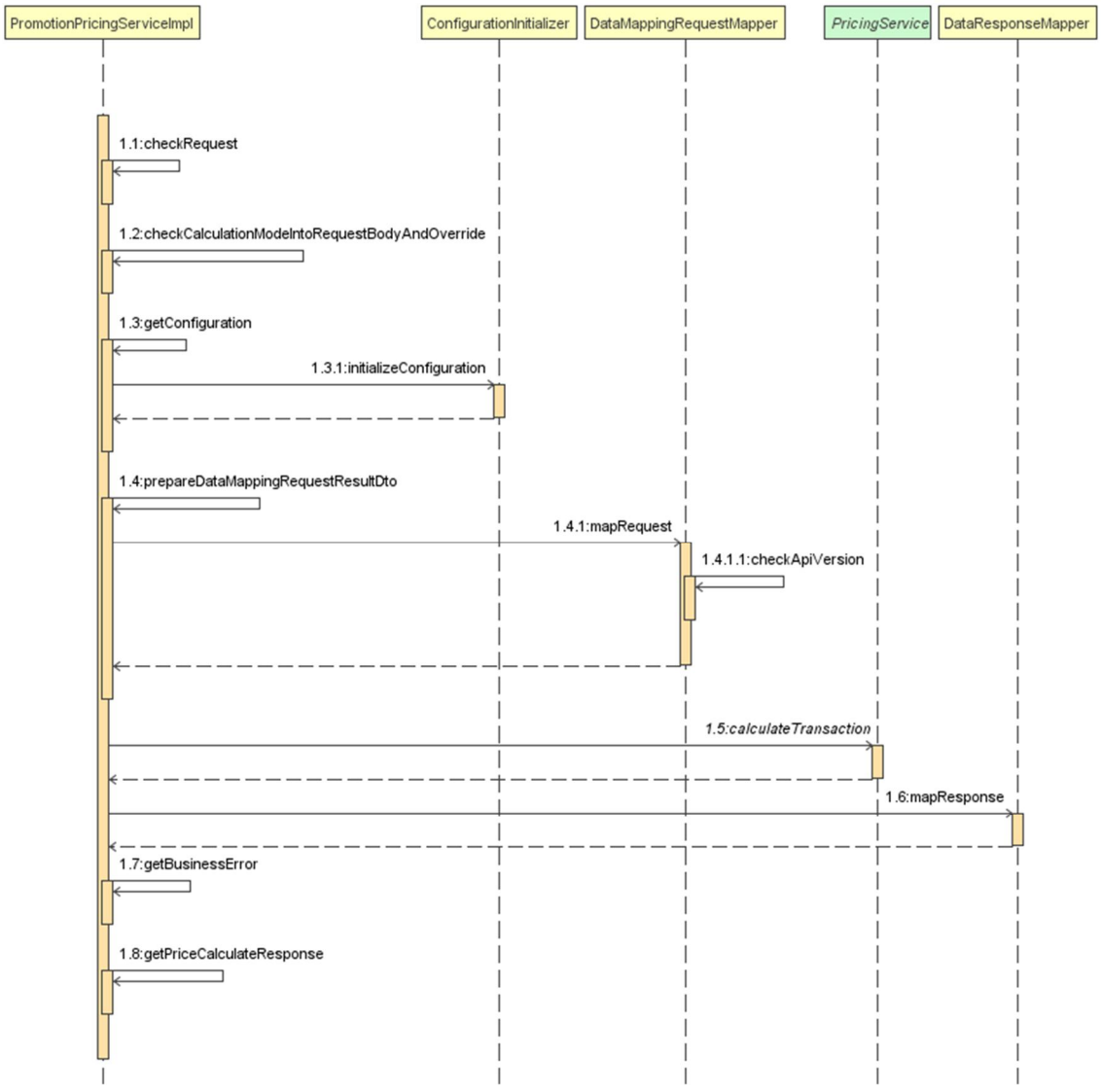
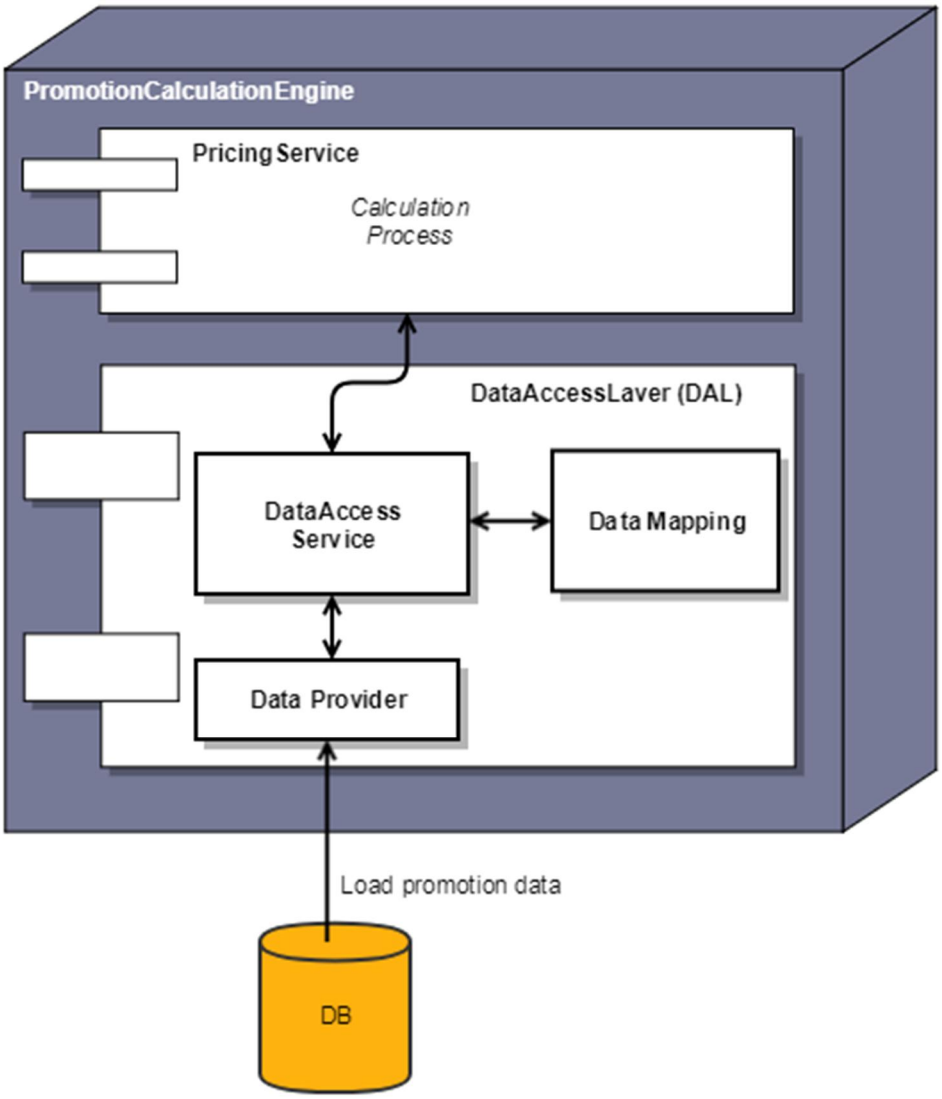


Figure 5 PromotionPricingService call sequence

5.8 The data access layer (DAL)

The Promotion Calculation Engine's Data Access Layer (DAL) module is responsible for the loading of the promotion master data used in the calculation and for the mapping of the environment-specific loaded data to the internal data model.

The Data Access Layer is accessible through the DataAccessService interface (com.gk_software.pricing_engine.common.data_access.DataAccessService) defined in the internal API. All vendor specific DAL implementations must implement this interface.



5.8.1 Spring configuration

The DAL implementation used by the PCE is configured via Spring beans. A delegate pattern is implemented to support seamless changes between different implementations of the DAL. For enabling a custom implementation of the DAL, the dataAccessService bean of type com.gk_software.pricing_engine.core.data_access.DataAccessServiceDelegate must be replaced as the code sample below demonstrates. The delegate property must reference the DataAccessService implementation intended to be used for accessing the persistence layer.

```
<bean id="dataAccessService"

class="com.gk_software.pricing_engine.core.data_access.DataAccessServiceDelegate">
       <property name="delegate" ref="dataAccessServiceVendorSpecific" />
</bean>
```

Code Block 6 DataAccessServiceDelegate bean

5.8.2 Promotion data loading

The DAL is accessible for the PCE through the DataAccessService interface. To make the integration of the DAL seamless for the PCE, only internal data model objects are used by the DataAccessService. The mapping between master data provider-specific data objects and the internal model is the responsibility of the DAL. The following methods are supported by the DataAccessService interface:

> As the DataAccessService is not part of the public API, it is not guaranteed to be release-stable and may change without notice.

Type	Method	Description
List<AssociatedMhgSO>	**findAssociatedMhgByChildId**(Context context, long businessUnitGroupID, String childMhgId, String merchandiseGroupIDQualifier, boolean recursive)	Finds all associations for the specified child merchandise hierarchy group. If the recursive parameter is set to true, the complete sequence of parents will be returned ordered from the closest to the root.
RoundingRuleDO	**findRoundingRuleByKey**(Context context, long businessUnitGroupID, Long roundingRuleID)	Finds the rounding rule by its external rule ID.

Type	Method	Description
int	**getChildEligibilitiesCount**(Context context, long eligibilityID)	Returns the count of child eligibilities of parent eligibility specified by the eligibility ID parameter.
List<PromotionConditionEligibilitySO>	**getChildEligibilitiesForParentEligibility**(Context context, long businessUnitGroupID, long parentInternalEligibilityID)	Loads all child eligibilities of the given parent eligibility.
CombinationPromotionConditionEligibilitySO	**getCombinationEligibility**(Context context, long eligibilityID)	Returns the combination eligibility specified by the given eligibility ID.
List<CouponPromotionConditionEligibilitySO>	**getCouponEligibilities**(Context context, long businessUnitGroupID, String couponNumber, List<SaleReturnTypeCode>saleReturnTypeENUMS)	Returns a list of coupon eligibilities that match the registered coupon.
List<CustomerGroupPromotionConditionEligibilitySO>	**getCustomerGroupEligibilities**(Context context, long businessUnitGroupID, String customerGroupID, List<SaleReturnTypeCode>saleReturnTypeENUMS)	Returns a list of customer group eligibilities that match the registered customer group.
PromotionConditionEligibilitySO	**getEligibility**(Context context, long eligibilityID)	Returns one concrete eligibility specified by the given eligibility ID.
ExternalActionPromotionConditionRuleSO	**getExternalActionRule**(Context context, long ruleID)	Returns the external action rule specified by the given rule ID.

Type	Method	Description
Get3Pay2PromotionConditionRuleSO	**getGet3Pay2Rule**(Context context, long ruleID)	Returns the get 3 pay 2 promotion condition rule specified by the given rule ID. Get3Pay2 rules are not supported by the PCE.
List<ItemPromotionConditionEligibilitySO>	**getItemEligibilitiesByFilter**(Context context, long businessUnitGroupID, Set<ItemInformation> itemsInfoSet)	Returns a list of item eligibilities matching the parameters.
RebatePromotionConditionRuleSO	**getItemRule**(Context context, long ruleID)	Returns the item price derivation rule specified by the given rule ID.
List<ManualPromotionConditionEligibilitySO>	**getManualPromotionEligibilities**(Context context, long businessUnitGroupID, String triggerType, String triggerValue, List<SaleReturnTypeCode>saleReturnTypeENUMS)	Returns a list of all manual trigger eligibilities matching the given trigger type and trigger value.
List<MarketBasketAmountEligibilitySO>	**getMarketBasketAmountEligibilities**(Context context, long businessUnitGroupID, List<SaleReturnTypeCode>saleReturnTypeENUMS)	Returns a list of all market basket amount eligibilities matching the parameters.
List<MHGPromotionConditionEligibilitySO>	**getMerchandiseGroupEligibilitiesByFilter**(Context context, long businessUnitGroupID, Set<ItemInformation> itemsInfoSet)	Batch query for merchandise hierarchy group eligibilities.
List<MerchandiseSetPromotionConditionEligibilitySO>	**getMerchandiseSetEligibilities**(Context context, long businessUnitGroupID, String itemNumber, String uomCode, Set<PromotionMhgFilterSO>promotionMhgFilterSet, List<SaleReturnTypeCode>saleReturnTypeENUMS)	Returns a list of all merchandise set eligibilities related to the given item or merchandise hierarchy group.

Type	Method	Description
List<Long>	**getMerchandiseSetIdsForItem**(Context context, String itemID, String uomCode, Set<PromotionMhgFilterSO>promotionMhgFilterSet, List<SaleReturnTypeCode>saleReturnTypeCodes)	Gets the IDs of all Merchandise Sets the specified item is included in.
List<MatchingItemSO>	**getMixAndMatchItems**(Context context, long ruleID)	Returns the mix and match price derivation item list of the rule specified with the given rule ID.
MixAndMatchPromotionConditionRuleSO	**getMixAndMatchRule**(Context context, long ruleID)	Returns the mix and match price derivation rule specified by the given rule ID.
PromotionSO	**getPromotion**(Context context, long businessUnitGroupID, long promotionID)	Returns the promotion specified by the given business unit group ID and promotion ID.
List<PromotionConditionSO>	**getPromotionConditions**(Context context, long eligibilityID)	Returns a promotion condition list specified by the given eligibility ID. If the eligibility ID is the root eligibility ID, only one promotion condition will be returned. If the eligibility ID is not the root eligibility ID, no promotion conditions will be returned.
PromotionConditionRuleSO	**getRule**(Context context, long ruleID)	Returns the promotion condition rule specified by the given ruleID.

5.8.3 Data mapping

The DAL implementation is responsible for the mapping between the data provider-specific model and the PCE internal data model. The implementation may also perform validation on the retrieved data to make sure it is compatible with the internal model.

The PCE internal data model supports extension fields for the promotion master data retrieved from the DAL. Each internal data object extends the ModelDto interface (com.gk_software.pricing_engine.model.ModelDto), which provides methods for saving the loaded extension fields and for accessing them at any point in the calculation. The DataAccessService implementation must take care of the mapping of these custom fields as well. The following methods are provided in the ModelDto interface for saving and retrieving the extension fields.

Type	Method	Description
T	get(String name)	Supports accessing the extension fields of the entity by name. Each field stored in the extension map is identified by its name. The name of the extension field is the name defined in the data access model's entity and does not necessarily match the name of the DB table column. T is the type of the extension field.
Map<String, ?>	getExtensions()	Accesses the extension data loaded from the Data Access Layer.
void	setExtensions(Map<String, ?> extensions)	Sets the extension data provided by Data Access Layer.

5.9 GK/Retail OmniPOS Integration

5.9.1 Overview

The Promotion Calculation Engine also supports direct integration with GK/Retail OmniPOS. In this case, the PCE is integrated as a module (jar files) to the POS and the Pricing Engine Promotion Service.

A significant difference between the POS and the SAP Promotion Pricing Service use case is that the POS may call the PCE several times within the same transaction, whereas in the SAP use case each transaction is completed with one call to the Pricing Engine Promotion Service. Because the PCE Context is kept within a session and is returned to the PCE with previous calculation results included, it is possible to cache data to

improve performance in the GK context. During request processing, the Clean promotion data from Transaction step is also performed to remove calculated results from previous PCE calls so that those will not influence the calculation.

Similar to the SAP Integration module, there is also a GK Integration main module, which contains the following sub-modules:

- Pricing Engine GK Data Access: GK data access integration.
- Pricing Engine GK Model: GK-specific implementation of the PCE model, contains the wrappers for POS data model to be used directly by the PCE.
- POS Pricing Service API: API for POS integration.
- POS Pricing Engine Component: Base implementation for POS API.
- POS Pricing Connector: Connector for the POS Service. Includes implementation of all functionality that require calling the POS Service.

HOW IT WORKS

The POS creates a PriceCalculationRequest object, which contains a PosSession and a Transaction object. Then from this the PCE PricingEngineRequest model object (com.gk_software.pricing_engine.PricingEngineRequest) is created using the AbstractRequestProcessor from *POS Pricing Engine Component* module and filled with the Transaction. The POS Transaction is wrapped with PCE Transaction object, so it can be used directly by the PCE. The Configuration is also mapped from the POS request. Then the PricingEngine is called with this PCE request.

When the calculation is done, a standard PriceCalculationResponse (com.gk_software.pricing_engine.pos.calculation.responses.CompleteResultResponse) object is returned, which contains POS Transaction object updated with the calculated result and which can be used by the POS.

5.9.2 PCE Model implementation

The Pricing Engine GK Model module contains a GK use case-specific implementation for the PCE Model API. The GK implementation of the model interfaces each extend the AdaptedDto<T> abstract implementation of the ModelDto (com.gk_software.pricing_engine.model.ModelDto), where T is the transaction or promotion data object to be mapped to the internal model. The external model object is saved to the AdaptedDto's T adaptedObject attribute. It is accessible through the methods of the internal model interface implemented. Thus, no extended request mapping from GK-specific transaction and promotion model to the internal PCE model is required, the GK internal model objects wrap the GK model objects.

5.9.2.1 Custom data on GK OmniPOS

In the OmniPOS there is currently no support for the any fields of the client API or the extension map on the SAP Promotion data model.

There is an alternative way to pass additional information dynamically to the PCE. This is done via XXCustom parameters. They are stored in the database and are available for instance in table CO_EL_PRDV as

XX_CUSTOM_0[1-5]_GK.

There are five available custom fields to put information.

```
Name      Null?    Type
--------- -------- -----------------
TENANT_ID NOT NULL VARCHAR2(20 CHAR)
NAME      NOT NULL VARCHAR2(80 CHAR)
STATE     NOT NULL NUMBER(38)
SCHEMA_ID          VARCHAR2(36 CHAR)
Usage: DESCRIBE [schema.]object[@db_link]
Name                          Null?    Type
----------------------------- -------- ------------------
ID_BSNGP                      NOT NULL NUMBER(18)
ID_EL_PRDV                    NOT NULL NUMBER(18)
TY_EL_PRDV                    NOT NULL VARCHAR2(4 CHAR)
PARENT_PRICE_DRV_RULE_EL_ID NOT NULL NUMBER(18)
ROOT_PRICE_DRV_RULE_EL_ID   NOT NULL NUMBER(18)
LEVEL_ID                      NOT NULL NUMBER(38)
EFFECTIVE_DATE_TIME           NOT NULL DATE
EXPIRATION_DATE_TIME          NOT NULL DATE
STATUS_CODE                            VARCHAR2(2 CHAR)
XX_CUSTOM_01_GK                        VARCHAR2(256 CHAR)
XX_CUSTOM_02_GK                        VARCHAR2(256 CHAR)
XX_CUSTOM_03_GK                        VARCHAR2(256 CHAR)
XX_CUSTOM_04_GK                        VARCHAR2(256 CHAR)
XX_CUSTOM_05_GK                        VARCHAR2(256 CHAR)
XID_GK                                 NUMBER(18)
FL_NG_GK                      NOT NULL CHAR(1 CHAR)
```

To get access to this field just use the method "get" available in the PCE model entities:

```
PromotionConditionEligibilitySO eligibility = getEligibility();
String custom = eligibility.get("XXCustom02");
```

In the same way transaction information may be enriched by the OmniPOS and is available via the XXCustom-fields in the transaction objects on PCE side.

It is also possible to implement the GKModelAdapter and GkModelExtractor to automate the conversion from and to any fields for the transaction model specific for an extension. See chapter Extension points for details.

5.9.3 Data Access Layer

A GK-specific implementation is also available for the DataAccessService (com.gk_software.pricing_engine.common.data_access.DataAccessService) for retrieving promotion data from the persistence layer. The promotion service for accessing the data used in the POS integration use case is provided by GK. Data retrieved using this service is again also wrapped by the GK-specific implementation for the PCE promotion data model objects, so it is easily usable by the standard pricing engine business logic and no additional mapping is required.

5.9.3.1 Promotion data cache

In the POS integration use case, one transaction might consist of several PCE calls. In all cases, the PCE calculation is executed the same way, no previous calculation results are taken from the request. As a result, the same calls must be made to the DAL for each calculation, causing some performance issues. To reduce performance overhead it is possible to use a local masterdata cache implemented in the PCE Context's CacheContext (com.gk_software.pricing_engine.api.context.CacheContext), which stores all loaded promotion masterdata for the given transaction. The local masterdata cache can be enabled through the configuration parameter localMasterdataCacheEnabled, the maximum size of the cached data for each promotion object can be set with the configuration parameter cacheSize.

5.9.4 POS Integration Extensions

The PCE supports extensions also for POS Integration. New plugin implementations may be included in the POS Pricing Engine Component module by adding new plugin.xml files like the following example shows:

```
<!-- Load pos pce plugins -->
<import resource="classpath*:META-INF/**/*-pos-plugin-pce.xml"/>
```

For detailed information about extensions, see chapter PCE Extensions.

5.9.4.1 Available POS extension plugins

There are plugin implementations available in the standard delivery for POS that are GK specific and therefor are not included in the core implementation of the PCE. The following table summarizes these extensions.

Extension point	Plugin	Supports criteria	Plugin order	Description
AdhocPromotionLoader	PosSessionPromotionLoaderImpl	Supported in any context	10000	Responsible for loading ad hoc promotions from the yMCA external system. For accessing the available ad hoc promotion list the plugin calls the POS service and then returns the promotion data. Bean definitionavailable in the pricing-pos-connector module: META-INF/com.gk-software.pricing-engine/gk-connector/promotion/ymca-pos-plugin-pce.xml

6 PCE Extensions

While the PCE supports already common features and a rich set of rules and eligibilities, there might be always special requirements that are specific to a customer environment. To allow implementation of additional features for such situations, the PCE Core provides a well-defined API and defined extension points. These extension points are designed for easily adding new functionality for common use cases.

The API for implementing PCE extensions is defined in the libraries pricing-engine-model and pricing-engine-api. The API defines two levels of stability to rely on for extension compatibility with future PCE versions:

- Extension stability: Someone implementing or sub-classing the “stable” entity can be sure that an upgrade does not lead to a syntax error.
- Consumer stability: Someone externally calling / referring to a “stable” entity can be sure that an update does not lead to a syntax error.

Most interfaces and classes in the API modules are sticking to consumer stability, so referencing them in custom code will work with future PCE versions. There are several interfaces defined as extension points that provide extension stability as well. These interfaces are marked with the annotation com.gk_software.pricing_engine.api.ExtensionPoint. An additional property ExtensionType on this annotation defines the kind of extension pattern for this extension point (for details on extension patterns, see chapter Extension Patterns). The following extension types are defined:

- Plugin: The extension point follows the plugin pattern and is integrated automatically via Spring.
- Overriding: This extension point allows overriding the default implementation and is configured via Spring bean replacement.
- Implement: Such an interface can be implemented by a custom extension depending on functional needs. It is used in method parameters or return values, but not defined as Spring bean.

Extensions implemented using PCE core extension points should work with no or minimal adoptions to the code in the SAP Promotion Pricing Service as well as GK/Retail OmniPOS.

6.1 Extension Patterns

6.1.1 Overview

The extensions for the PCE follow some common patterns to extend the PCE functionality at different places with minimal effort and simple configuration. These recurring patterns allow an extension developer to extend the different places in a similar way.

All extension patterns of the PCE are based on extension of the Spring application context of the PCE.

6.1.2 Plugins

The general extension pattern for most extension points follows the plugin design pattern. A plugin is essentially the implementation of a small interface which is dynamically found and injected in the PCE processing flow via Spring. To support easy integration, the PCE plugin handling uses the Spring plugin framework.

All PCE plugin interfaces share some basic principles:

- Small interfaces, easy to implement
- Automatically found by the PCE plugin handling
- Dynamic injection in the PCE application context
- Allow to define an order of processing
- PCE Default Plugins can be replaced via Spring bean replacement.

HOW IT WORKS

Plugin extension points are defined as simple Java interfaces. All PCE plugins extend the org.springframework.plugin.core.Plugin interface. This interface provides a method boolean supports(S delimiter) to allow the plugin framework to check whether a plugin should be used for a specific context. This is done by the PCE core depending on the plugin pattern for a given extension point (see below). Each plugin extension point interface in the PCE is annotated by the ExtensionType.plugin and extends the Plugin interface:

```
@ExtensionPoint(ExtensionType.Plugin)
public interface EligibilityValidator extends Plugin<PromotionConditionEligibilitySO>
```

The available plugins of a given type are collected in the PCE Spring application context. This application context is defined in the pricing-engine-impl module at the path: META-INF/com.gk-software.pricing-engine/core/default-beans.xml.

The collection of the plugins is done automatically using the Spring XML plugin namespace, based on the plugin interface:

```
<!-- List of eligibility validators -->
<alias name="defaultEligibilityValidators" alias="eligibilityValidators"/>
<plugin:list id="defaultEligibilityValidators"
class="com.gk_software.pricing_engine.api.eligibility.EligibilityValidator"/>
```

The PCE provides a number of default plugins that define the core calculation logic. The default plugins are defined in specific application context files named default-plugin-pce.xml in the pricing-engine-impl module. These application context files are organized in subfolders of META-INF/com.gk-software.pricing-engine/core. They are grouped by responsibility:

Group	Description
conditions	Default plugins related to condition loading, handling.
eligibilities	Default plugins related to eligibility handling like EligibilityLoaderFragment and EligibilityResultParser.
filters	Default item filter plugins.
price-modifications	Plugins for handling the calculation logic for standard price modification codes.
rules	All plugins related to processing of standard rule, for example, CalculationRuleProcessor and RebateDistributionProcessor implementations.
transaction-analyzers	Default plugins for retrieving content from transaction and fill the Context object.
validators	Validator plugins for standard validation of conditions, eligibilities, and results.

Details on the bean definitions can be found in chapter Pricing Engine Core Implementation.

THERE IS MORE

Besides the common features for plugins, all plugin extension points are grouped by different usage patterns. These usage patterns for plugins define which plugin shall be used in which order and how multiple plugins for the same extension point are handled. The patterns are based on well-known software design patterns.

There are actually three different kinds of plugin patterns in use in the PCE:

- Strategy pattern
- Filter Chain pattern

- Chain of responsibility pattern

6.1.2.1 Strategy pattern

The plugin strategy pattern is based on the strategy design pattern. Based on the context of use, different implementations will be used.

An extension point based on the strategy pattern does not know which implementation will handle what. All implementing classes for a plugin will be called and the ones that are designed to take action will execute the corresponding logic. It is also possible that more than one called plugin will execute its logic.

HOW IT WORKS

The plugin framework integrated in the PCE Spring application context will collect all plugins implementing a specific plugin interface. The list may be further filtered by checking the supports method of the plugin implementation. All plugins are then called in the defined order. Each plugin decides on its own whether to perform some action or not. This pattern is used for plugins when each plugin is responsible for a specific type handling and all types should be handled accordingly. For example, when loading eligibilities, each EligibiltiyLoaderFragment plugin takes care of the loading of a specific eligibility type. This means there is a CouponEligibilityLoader, an ItemEligibilityLoader, a MerchandiseSetEligibilityLoader etc.

There are several plugin types that are based on this strategy:

Plugin Interface	Supports context	Description
ContextPostProcessor	Context	• Process different parts of the context as the first step of processing.
TransactionAnalyzer	No check done	• Parse different parts of the transaction. • Store information to Context. • Default analyzers for standard transaction parts like coupons, items, manual triggers.
AdhocPromotionLoader	Context	• Load ad hoc promotions from external systems.

Plugin Interface	Supports context	Description
EligibilityLoaderFragment	No check done	• Handle loading and initialization of eligibilities. • Provides eligibility-specific logic. • Called from DefaultEligibilityHandler. • All configured fragments are called in order. • Default implementations for all standard eligibility types expect manual eligibilities.
EligibilityLoader	No check done	• For complete customization of eligibility loading if fragment is not enough. • Needs to handle creation of Wrapper, registration in context and activation. • DefaultEligibilityLoader and ManualEligibilityLoader.
EligibilityResultParser	EligibilityWrapper	• Preprocess eligibility for calculation. • Parse eligibility and result data. • Updates PartialResult. • Default implementations for context specific eligibilities.
ConditionLoaderFragment	Context	• Loads promotion data for the activated eligibilities.
EligibilityRuleHandler	PartialResult	• Takes care of common rule handling and applying • Processes intervals, transaction and position related rules
EligibilityRuleHandlerFragment	PartialResult	• Takes care of eligibility type specific rule handling

Plugin Interface	Supports context	Description
FrequentShopperPointsModifierCustomizer	PartialResult	• Customizer for FrequentShopperPointsModifier result data.
LoyaltyRewardLineItemCustomizer	PartialResult	• Customizer for LoyaltyRewardLineItem result data.
RetailPriceModifierCustomizer	PartialResult	• Customizer for RetailPriceModifier result data.
PriceModificationLineItemCustomizer	PartialResult	• Customizer for PriceModificationLineItem result data.
RetailTransactionPromotionPriceDerivationRuleCustomizer	PartialResult	• Customizer for RetailTransactionPromotionPriceDerivationRule result.

6.1.2.2 Filter Chain Pattern

The filter chain pattern follows the intercepting filter software design pattern. Filter Chain extension points build up a list of multiple matching plugins and execute them in defined order on the target. The methods called on these plugin interfaces typically return a boolean value. All called filters must return true for the processing to carry on.

This kind of pattern is typically used for validation of an object. Only if all validators prove the object is valid, it is accepted for processing.

HOW IT WORKS

Like the strategy pattern, all plugins implementing a specific plugin interface for the filter chain extension point are collected. For each of these filter chain plugins, the supports(...) method is checked to determine whether the filter should be applied in a given context. All matching plugins are then called in order and the filter method is checked. As soon as one method returns false, the chain is stopped and the processing aborted.

A good example is the ConditionValidator extension point. Before a promotion condition is processed, the validators are checked to determine whether it should be applied. One concrete implementation is the

ExclusiveConditionValidator that checks whether there are already exclusive flagged promotions applied in the context that prevent additional promotions from being used.

The following extension points are implemented according to the Filter Chain pattern:

Plugin Interface	Supports context	Description
EligibilityValidator	EligibilityWrapper	• Check eligibility after loading and initialization whether it should be used in the further processing. • Implementations for generic checks (= supports all eligibilities) and eligibility specific checks (i.e. MarketBasketAmountEligibilityValidator).
ConditionValidator	PromotionConditionWrapper	• Check condition can be applied before calculation.
ResultValidator	PartialResult	• Checks a result after calculation. • If invalid result will be skipped and not applied to summary.
RebatableItemFilter	PromotionConditionWrapper	• Filter items for rebate calculation.
EligibilityRuleHandler	PartialResult	• Handle eligibility specific rule calculation. • Process items and intervals. • Default implementations for standard Threshold type eligibilities.
RebateValidator	PartialResult	• Check whether a rebate is valid and can be applied. • Validation is performed in case of valid partial results before the SummaryResult is updated.
RebatableItemFilterPlugin	PromotionConditionWrapper	• Checks whether an item meets the defined criteria and decides if it is rebatable or not.

6.1.2.3 Chain of Responsibility Pattern

This plugin pattern works like the well-known software design pattern of the same name. All plugins found for an extension point are checked in order and the first one that supports the defined context will be used to execute the logic.

For these plugins, the order of plugins is important to allow to handle the processing in a plugin. This means more specific plugins should be ordered before generic plugins that may provide a default implementation and support every possible context.

HOW IT WORKS

In the PCE core logic, the list of chain of responsibility plugins is collected and injected as for all plugins via the Spring context. The plugin list is then iterated in the order provided in the list. For each plugin implementation found, the supports(...) method is checked. When a plugin returns true, this plugin is used in processing the current context. A good example is the PriceModification interface that is looked up for the first implementation supporting the PriceModificationMethodCode of a PromotionConditionRule.

The following extension points are implemented according to the Chain of Responsibility pattern:

Plugin Interface	Supports context	Description
ThresholdActivationCheck	ThresholdType	• Check eligibility activation based on threshold. • Initial check using plain transaction information without promotions.
RuleLoader	RuleType	• Customization of CalculationRule loading. • Overrule default rule loading. • Default implementations for standard rule types.
CalculationRuleProcessor	PromotionConditionRuleSO	• Evaluate Rule. • Fill PartialResult. • Default implementations for standard rule types.
ExclusiveCheckHandler	PromotionConditionSO	• Handle exclusion check at different places. • Extension Point for customizing exclusive checks.

Plugin Interface	Supports context	Description
PriceModification	PromotionConditionRuleSO	• Calculate Price for different price modification method codes. • Calculation varies depending on PriceCalculationContext. • Default implementations for standard method codes.
RebateDistributionProcessor	PromotionConditionRuleSO	• Rule-specific rebate distribution. • Default implementations for standard rule types.
MixAndMatchProrationCustomizer	PartialResult & MatchingItemSO	• Customize Mix and Match proration.
BestPriceCalculator	List of PartialResult	• Validate whether the conflicting PartialResult list provides the best price variant.

6.1.2.4 Plugin Order

To have a reproducible behavior and allow control of used plugins for the different patterns, it is crucial to call the plugins in a defined order. Luckily, the plugin handling supports the ordering via Spring by adding the Order annotation.

HOW IT WORKS

The plugin:list namespace element collects all plugins implementing the defined interface that are found in the application context. The list of plugins will be ordered by Spring checking the @Order annotation found at a plugin implementation class. If no annotation is found, the respecting plugins are ordered after any @Order marked plugin in the order found in the application context.

> It is highly recommended marking all custom plugins with an @Order annotation to guarantee a defined behavior.

The Order annotation has an integer as parameter to define the ordering criteria. Ordering is done numerical by this integer from low to high.

```
@Order(999)
public class CustomCouponTransactionAnalyzer implements TransactionAnalyzer
```

THERE IS MORE

To prevent a custom implementation from getting in conflict with default plugins and to be able to control how a custom plugin is called in respect to the default plugins, there is a reserved number range for the default plugins.

The default plugins for the PCE core have all defined an order in the range of 0 to 100000. Default plugins only use full hundred values (i.e. 100, 200, 1000, 1100, 10000 etc.). All values that are smaller than 0 or greater than 100000 or between the 100s can be used for custom implementations.

SAP reserves the range between -10000000 and 10000000 to be used as Order for custom plugin implementations. Any values within this range that are not reserved for the PCE core plugins may be used.

Details on which order value is defined for the default plugins can be found in the description of the Pricing Engine Module: Impl.

6.1.3 Bean Overrides

Replacing existing functionality that is not provided as plugin extension point is sometimes required to adopt the behavior to customer needs.

To do so, you have to write your own implementation of the related class/interface and define it as a bean in your custom bean context.

All beans that can be replaced are defined with an alias (see below) to easily replace the bean without changing existing definitions.

HOW IT WORKS

For example, to replace the standard ConditionLoader with a custom one add the new, custom bean with alias conditionLoader:

```
<!-- Custom Condition Loader, Overwrite standard one-->
<alias name="customConditionLoader" alias="conditionLoader"/>
<bean id="customConditionLoader" class="foo.bar.CustomConditionLoader"/>
```

THERE IS MORE

Due to the dynamic plugin collection via the plugin framework, it is not possible to replace default plugin implementations via the bean alias. If you really need to completely replace a default plugin, you have to reference the default bean id for this plugin bean.

```
<!-- Extension of Transaction Analyzer overwrite standard one-->
<alias name="customCouponTransactionAnalyzer" alias="defaultCouponTransactionAnalyzer"/>
<bean id="customCouponTransactionAnalyzer"
class="foo.bar.CustomCouponTransactionAnalyzer"/>
```

6.1.4 Extending PCE standard implementations

In special cases you may want to extend some existing functionality without completely rewriting everything. While this is not covered as part of the stable API, subclassing PCE beans could offer a very flexible way to extend the calculation logic. You must consider in this case that there is no guaranteed compatibility between different PCE versions for classes and other artifacts outside the defined api packages.
This means a method signature may change over time, making custom subclass syntactically incorrect on a new PCE version. The probability that a PCE object will be changed in an incompatible way increases from first to last entry in the following list:

- Spring bean ID/alias
- Public constants and Enums
- Java interface (methods may, however, be added)
- Signature of public method of a Java class
- Signature of protected method of a Java class
- Protected attributes of a Java class

6.2 Extension Points

6.2.1 Context

6.2.1.1 Add additional initialization logic to request processing

As a custom requirement, it might be necessary to execute some functionality before any processing was done by the PCE.

The PCE provides the ContextPostProcessor extension point with Context as supports context for such scenarios. As the internal PCE Context is available for the plugin execution, the plugin provides the possibility of adding any additional initialization processing on the request or the request transaction.

The ContextPostProcessor is defined in the interface ContextPostProcessor (com.gk_software.pricing_engine.api.context.ContextPostProcessor).

GETTING READY

For adding custom functionality to be executed before the PCE calculation, the following steps must be taken:

- Add custom implementation of the ContextPostProcessor interface.
- Implement supports method in a way that the plugin is only used in the desired context (Context).
- Mark plugin with @Order annotation in case the order of execution is important. As the execution is done using the Strategy extension pattern, all supported plugins will be executed.
- Add bean to the Spring application context for the custom implementation of the ContextPostProcessor.

HOW IT WORKS

All available implementations of the ContextPostProcessor are gathered by the Spring application context. They are executed as the last step of initializing the internal context, so all information from the request transaction is already available in it, but before any further execution steps would have been made by the PCE calculation. The plugins are executed by the executeContextPostProcessors method in the PricingEngineImpl (com.gk_software.pricing_engine.core.engine.PricingEngineImpl), the PCE core implementation for the PricingEngine interface. Execution is done by using the Strategy extension pattern.

THERE IS MORE

The following PCE core implementations are provided for the ContextPostProcessor plugin:

Plugin	Supports criteria	Plugin order	Description
DataAccessInitializationProcessor	The RequestContext is not null AND The context mode is set to ContextMode.STANDARD	-	The plugin takes the PCE request from the RequestContext and calls the initialize method provided by the SAP Promotion Service.

6.2.1.2 Parse additional information from the transaction

The PCE provides the TransactionAnalyzer extension point with TransactionAnalyzerType (com.gk_software.pricing_engine.model.typecodes.TransactionAnalyzerType) as supports context for adding custom logic for analyzing and processing a specific part of the transaction.

The TransactionAnalyzer is defined in the interface TransactionAnalyzer (com.gk_software.pricing_engine.api.analyzer.TransactionAnalyzer).

GETTING READY

For adding custom transaction analyzers, the following steps must be taken:

- Add custom implementation of the TransactionAnalyzer interface.
- Extend the TransactionAnalyzerType with the value supported by the custom analyzer.
- Implement supports method in a way that the plugin is only used in the desired context (TransactionAnalyzerType).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the TransactionAnalyzer.

HOW IT WORKS

All implementations for the TransactionAnalyzer registered in the Spring application context are executed as a last step of the PCE request processing.

The plugins are executed by the executeTransactionAnalyzer method in the PricingEngineImpl (com.gk_software.pricing_engine.core.engine.PricingEngineImpl). Execution is done by using the Strategy extension pattern, which means all implementing classes for the plugin will be called and the ones that are designed to take action will execute the corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the TransactionAnalyzer plugin:

Plugin	Supports criteria	Plugin order	Description
CouponTransactionAnalyzer	Transaction analyzer type "COUPON"	1000	Analyzes coupon-related transaction segments.
ItemTransactionAnalyzer	Transaction analyzer type "ITEM"	2000	Analyzes item and mhg-related transaction segments.
CustomerGroupTransactionAnalyzer	Transaction analyzer type "CUSTOMER_GROUP"	3000	Analyzes customer group-related transaction segments.
ExternalPromotionsTransactionAnalyzer	Transaction analyzer type "EXTERNAL_PROMOTIONS"	4000	Analyzes external promotion-related transaction segments.
PromotionManualTriggerTransactionAnalyzer	Transaction analyzer type "PROMOTION_MANUAL_TRIGGER"	5000	Analyzes manual trigger-related transaction segments.

When you add custom TransactionAnalyzer plugins in an extension, it is recommended to add an @Order annotation with higher value (>100000) than the default plugins to be able to access already initialized context data like parsed items.

6.2.2 Eligibilities

The PCE defines several extension points regarding eligibility loading, validation, and processing throughout the calculation. For extending the PCE with custom eligibility types, it is necessary as a minimum to implement the plugins responsible for the loading of the eligibility from the data access layer. Depending on the specified functionality, it might be also required to add implementation for other eligibility-related extension points as well. The following section describes all extension points related to the loading, activating, and processing of eligibilities used throughout the PCE calculation process.

6.2.2.1 Loading a custom eligibility

The PCE defines two extension points responsible for the loading of eligibilities: The eligibility loader is responsible for validating and activating the loaded eligibilities and their parent eligibilities, for registering them and for filling the item bucket for each eligibility. The eligibility loader extension point is defined in the interface EligibilityLoader (com.gk_software.pricing_engine.api.eligibility.EligibilityLoader) with extension context EligibilityType (com.gk_software.pricing_engine.model.typecodes.promotion.EligibilityType).

The second extension point available for eligibility loading related plugins is the eligibility loader fragment, which is responsible for handling eligibility-specific steps such as loading of a specific PromotionConditioneligibilitySO from the persistence layer, while the EligibilityLoader takes care about general registration. The eligibility loader fragment is defined in the generic interface EligibilityLoaderFragment (com.gk_software.pricing_engine.api.eligibility.EligibilityLoaderFragment <E extends PromotionConditionEligibilitySO>) with extension context EligibilityType (com.gk_software.pricing_engine.model.typecodes.promotion.EligibilityType).

GETTING READY

For loading custom eligibilities, the following steps must be taken:

- Add custom implementation of the EligibilityLoader interface. This step is only necessary if the registration process of the eligibility differs from the one implemented as the default eligibility loader in the PCE core implementation.
- Add custom, eligibility specific implementation of the EligibilityLoaderFragment interface.
- Implement supports method for both in a way that the plugin is only used in the desired context (EligibilityType).
- Mark both plugins with @Order annotation in case the order of execution is important. As the execution is done using the Strategy extension pattern, in both cases all supported plugins will be executed.
- Add beans to the Spring application context for the custom implementations of the EligibilityLoader and EligibilityLoaderFragment.

HOW IT WORKS

All implementations for the EligibilityLoader that are registered in the Spring application context are executed after PCE request processing was done.

The plugins are executed by the executeEligibilityLoader method in the PricingEngineImpl (com.gk_software.pricing_engine.core.engine.PricingEngineImpl). Execution is done by using the Strategy extension pattern, which means all implementing classes for the plugin will be called and the ones that are designed to take action will execute the corresponding logic.

The eligibility loader processes and registers eligibilities in the following way. The process is executed for each supported eligibility type. New EligibilityLoader implementations may extend this functionality.

- Loads eligibilities from the data access layer based on information saved to the Context on transaction analysis. In case of the PCE default eligibility loader, the type-specific operation is performed by the registered EligibilityLoaderFragment plugins.
- EligibilityWrapper object is created and initialized for each loaded eligibility. The item bucket related to the eligibility and the working bucket are filled.

- The eligibility and the parent eligibilities (if there is any) are registered to the context. The eligibility type-specific registration is handled by the EligibilityLoaderFragment plugins in case of the PCE default eligibility loader.
- In case of threshold eligibilities, the default threshold type-specific validations are performed.
- The eligibility status is set. In case of Active status, the activation is performed on the eligibility and the parent eligibilities, if there are any.

The eligibility loader fragments are responsible for loading a specific type of eligibility from the persistence layer, registering it to the type-specific collections in the context, and getting the items related to the eligibility when loading the EligibilityWrapper buckets.

THERE IS MORE

The following PCE core implementations are provided for the EligibilityLoader plugin:

Plugin	Supports criteria	Plugin order	Description
DefaultEligibilityLoader	All eligibility type codes	1000	Default eligibility loader. Implementation for common eligibility registration process. As most eligibilities require the same registrations steps, it may not be necessary to add a custom implementation for the EligibilityLoader interface. The default loader may be used with a custom EligibilityLoaderFragment plugin.
ManualTriggerEligibilityLoader	Eligibility type code "MANU"	2000	Manual trigger eligibility-specific eligibility loader, since the registration of manual trigger-related eligibilities differs from all other supported eligibilities.

The following PCE core implementations are provided for the EligibilityLoaderFragment plugin:

Plugin	Supports criteria	Plugin order	Description
CouponEligibilityLoader	Eligibility type code "COUP"	1000	It tries to find coupon eligibilities for coupon data collected from the transaction and implements type specific handling.
CustomerGroupEligibilityLoader	Eligibility type code "CGRP"	2000	It tries to find customer group promotion condition eligibilities for customer group data collected from the transaction and implements type specific handling.
ItemEligibilityLoader	Eligibility type code "ITEM"	3000	It tries to find item eligibilities for item data collected from the transaction and implements type specific handling.
MarketBasketAmountEligibilityLoader	Eligibility type code "TOTL"	4000	It tries to find market basket amount eligibilities based on the information available in the transaction and implements type specific handling.

Plugin	Supports criteria	Plugin order	Description
MerchandiseGroupEligibilityLoader	Eligibility type code "MSTR"	5000	It tries to find mhg eligibilities for merchandise hierarch group data collected from the transaction and resulting from the analysis of the transaction and implements type specific handling.
MerchandiseSetEligibilityLoader	Eligibility type code "MSET"	6000	It tries to find merchandise set eligibilities for item data collected from the transaction and implements type specific handling.
AdhocEligibilityLoader	All eligibility types are supported	10000	It tries to find all eligibilities that are part of previously loaded ad hoc promotions. The AdhocEligibilityLoader has a list of eligibility type specific delegate fragments registered that contain all available eligibility loader fragments. The delegates perform the actual type specific handling of these ad hoc eligibilities, but the ad hoc eligibility specific processing is implemented in this plugin.

6.2.2.2 Add validation checks on eligibilities

By adding plugin implementations for the extension point EligibilityValidator, it is possible to extend the eligibility activation process with any custom requirements. The PCE EligibilityValidator extension point has the EligibilityWrapper (com.gk_software.pricing_engine.api.eligibility.EligibilityWrapper) class as its supports context, which makes eligibility type or any other eligibility attribute-specific validation possible.

The EligibilityValidator is defined in the interface EligibilityValidator (com.gk_software.pricing_engine.api.eligibility.EligibilityValidator).

GETTING READY

For adding custom eligibility validators, the following steps must be taken:

- Add custom implementation of the EligibilityValidator interface.
- Implement supports method in a way that the plugin is only used in the desired context (EligibilityWrapper).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the EligibilityValidator.

HOW IT WORKS

The validation of loaded eligibilities is a step of the eligibility loading process implemented by the EligibilityLoader plugins. The eligibility validators registered to the Spring application context are executed using the Filter Chain extension pattern, which means only eligibilities that are deemed valid by all registered plugins may be activated and registered to the context.

The validation is performed by the validateEligibility method of the core PCE implementation of the EligibilityRegistry interface (com.gk_software.pricing_engine.api.eligibility.EligibilityRegistry).

THERE IS MORE

The following PCE core implementations are provided for the EligibilityValidator plugin:

Plugin	Supports criteria	Plugin order	Description
EligibilityExpirationValidator	Supports all eligibilities	1000	Validates if the time validation method is set to eligibilities, and if yes then checks the validity of the eligibility.
EligibilityStatusValidator	Supports all eligibilities	2000	Validates if the eligibility's status is active.
MarketBasketAmountEligibilityValidator	Supports market basket amount eligibilities	3000	Checks whether the total basket amount in the transaction reaches the threshold amount defined by the eligibility.
AdhocEligibilityFulfillmentValidator	Supports ad hoc eligibilities only	4000	Validates that the loaded ad hoc promotion related eligibility is fulfilled by the current transaction or not.

6.2.2.3 Customize threshold check logic

The eligibility activation process checks each threshold type eligibility for a threshold type-specific acceptance criteria before performing the activation. For example, in case of quantity-related threshold types it is checked whether the threshold quantity defined in the master data has been reached and that the threshold quantity is not greater than the limit quantity. The status is set based on the result.

The PCE ThresholdActivationCheck extension point has the ThresholdType (com.gk_software.pricing_engine.model.typecodes.promotion.ThresholdType) class as its supports context.

The ThresholdActivationCheck is defined in the interface ThresholdActivationCheck (com.gk_software.pricing_engine.api.eligibility.ThresholdActivationCheck).

GETTING READY

For adding custom threshold activation check plugins, the following steps must be taken:

- Add custom implementation of the ThresholdActivationCheck interface.
- Implement supports method in a way that the plugin is only used in the desired context (ThresholdType).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the ThresholdActivationCheck.

HOW IT WORKS

The threshold activation checks are performed on already registered eligibilities as part of the eligibility loading process. The checks are performed by the ThresholdEligibilityProcessorImpl, the PCE core implementation for the ThresholdEligibilityProcessor interface (com.gk_software.pricing_engine.common.eligibility.ThresholdEligibilityProcessor). The threshold activation check plugins registered to the Spring application context are executed using the Chain of Responsibility extension pattern, which means that the first one that supports the defined context will be used to execute the logic. Thus, it is of great importance to define the order of registered plugins with the use of the @Order annotation.

THERE IS MORE

The following PCE core implementations are provided for the ThresholdActivationCheck plugin:

Plugin	Supports criteria	Plugin order	Description
ThresholdActivationCheckQUT	Threshold types "QUT" and "QUTI"	1000	Threshold check for eligibilities with quantity related threshold type.
ThresholdActivatonCheckAMT	Threshold types "AMT" and "AMTI"	2000	Threshold check for eligibilities with amount related threshold type.
ThresholdActivationCheckCOMB	Threshold type "COMB"	3000	Threshold check for ItemOr eligibilities (Simple Product Group).
ThresholdActivatonCheckAMQU	Threshold type "AMQU"	4000	Threshold check for eligibilities with quantity-amount related threshold type.

6.2.2.4 Add additional eligibility specific information to promotion calculation

The PCE promotion calculation process requires some additional eligibility-related information besides what is registered to the context. For each applied condition, the information about the number of used eligibilities and type-specific lists of those must be available in the partial result. To provide this data, the so-called eligibility result parsers are executed as part of the initialization of each partial results.

The PCE EligibilityResultParser extension point has the EligibilityWrapper (com.gk_software.pricing_engine.api.eligibility.EligibilityWrapper) class as its supports context.

The eligibility result parser is defined in the interface EligibilityResultParser (com.gk_software.pricing_engine.api.eligibility.EligibilityResultParser).

GETTING READY

For adding custom eligibility parsers, the following steps must be taken.

- Add custom implementation of the EligibilityResultParser interface.

- Implement supports method in a way that the plugin is only used in the desired context (EligibilityWrapper).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the EligibilityResultParser.

HOW IT WORKS

The eligibility result parsers are registered through the Spring application context to the EligibilityServiceImpl, the PCE core implementation for the EligibilityService interface (com.gk_software.pricing_engine.common.eligibility.EligibilityService), by which they are executed. The eligibility parsing is executed during the initialization of the partial result created for each condition calculation. The Strategy extension pattern is used for the plugin execution. Thus, all registered plugins that are supported in the given context will be executed.

THERE IS MORE

The following PCE core implementations are provided for the EligibilityResultParser plugin:

Plugin	Supports criteria	Plugin order	Description
ItemEligibilityParser	Eligibilities with type code "ITEM".	1000	Parser for item eligibilities.
MhgEligibilityParser	Eligibilities with type code "MSTR".	2000	Parser for merchandise hierarchy group eligibilities.
CombinationEligibilityParser	Eligibilities with type code "COMB".	3000	Parser for combination eligibilities.
CouponEligibilityParser	Eligibilities with type code "COUP".	4000	Parser for coupon eligibilities.
TotalEligibilityParser	Eligibilities with type code "TOTL".	5000	Parser for market basket amount eligibilities.
MerchandiseSetEligibilityParser	Eligibilities with type code "MSET".	6000	Parser for merchandise set eligibilities.

6.2.3 Promotion data loading

6.2.3.1 Load conditions

The ConditionLoader component of the PCE is responsible for loading, validating and registering to the context promotion, promotion condition and promotion condition rule master data based on the previously activated eligibilities. In the ConditionLoader the general processing is implemented, but accessing the data from the appropriate sources is done in the ConditionLoaderFragment plugins registered to the loader.

The ConditionLoaderFragment extension point with Context (com.gk_software.pricing_engine.api.context.Context) as supports context is provided by the PCE for retrieving promotion condition data from the designated source.

The conditionLoaderFragment is defined in the interface ConditionLoaderFragment (com.gk_software.pricing_engine.api.engine.ConditionLoaderFragment).

GETTING READY

For adding custom rule loaders, the following steps must be taken:

- Add custom implementation of the ConditionLoaderFragment interface.
- Implement supports method in a way that the plugin is only executed in the desired context.
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the ConditionLoaderFragment.

HOW IT WORKS

The plugins are registered to the ConditionLoaderImpl class' conditionLoaderFragments list via Spring and are executed during the loading of conditions for active eligibilities. Execution is done by using the Strategy extension pattern, which means all implementing classes for the plugin will be called and the ones that are designed to take action will execute the corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the ConditionLoaderFragment plugin:

Plugin	Supports criteria	Plugin order	Description
DefaultConditionLoaderFragment	Supported in any context	1000	The default condition loader fragment is responsible for loading the required master data from the DAL.
AdhocConditionLoaderFragment	Supported in any context	2000	The ad hoc condition loader fragment is responsible for loading promotion data of ad hoc promotions. As ad hoc promotions are stored in the PCE Context, it is retrieved from there.

6.2.3.2 Load additional rule information

As a custom requirement, it might be necessary to introduce a new rule type to the PCE. As a part of this extension, the condition loader must be prepared for loading the newly introduced PromotionConditionRuleSO from the persistence layer.

The RuleLoader extension point with RuleType (com.gk_software.pricing_engine.model.typecodes.promotion.RuleType) as supports context is provided by the PCE for retrieving type-specific promotion condition rules from the data access layer.

The RuleLoader is defined in the interface RuleLoader (com.gk_software.pricing_engine.api.rules.RuleLoader).

GETTING READY

For adding custom rule loaders, the following steps must be taken:

- Add custom implementation of the RuleLoader interface.
- Implement supports method in a way that the plugin supports just the RuleType of the newly introduced rule and this way is used only in the desired context.
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the RuleLoader.

HOW IT WORKS

The type-specific promotion condition rules are loaded during the first step of the condition calculation: load conditions for active eligibilities. The plugins are executed by the loadConditionRule method in the ConditionLoaderImpl (com.gk_software.pricing_engine.core.engine.ConditionLoaderImpl), the core implementation provided by the PCE for the ConditionLoader interface. Execution is done by using the Chain of Responsibility extension pattern, which means the first plugin implementation found supporting the given RuleType will be used to load the promotion master data. Thus, it is crucial to define the order of registered plugins to ensure the expected behavior.

THERE IS MORE

The following PCE core implementations are provided for the RuleLoader plugin:

Plugin	Supports criteria	Plugin order	Description
RebateRuleLoader	Rule type is "RB"	1000	Loads rebate promotion condition rules from the data access layer.
MixAndMatchRuleLoader	Rule type is "MM"	2000	Loads mix and match rules from the data access layer.
ExternalActionRuleLoader	Rule type is "EX"	3000	Loads external action rules from the data access layer.
StandardPriceRuleLoader	Rule type is null	9000	Loads standard promotion condition rules from the data access layer.

6.2.3.3 Load ad hoc promotions

As a custom requirement it may be necessary to load promotion master data from an external system during calculation. To do so a custom implementation of the AdhocPromotionLoader extension point must be added. The plugin's getPromotions method is expected to return the available ad hoc promotions loaded from the external system and mapped to the PCE internal model's PromotionSO. The loaded promotions must include the complete set of promotion data objects in the tree structure. All calls to external systems and required data mapping to the PCE internal model must be implemented in the plugin.

The AdhocPromotionLoader extension point with Context (com.gk_software.pricing_engine.api.context.Context) as supports context is provided by the PCE for the loading of ad hoc promotion data.

The AdhocPromotionLoader is defined in the interface AdhocPromotionLoader (com.gk_software.pricing_engine.api.engine.AdhocPromotionLoader).

GETTING READY

For adding custom ad hoc promotion loaders, the following steps must be taken:

- Add custom implementation of the AdhocPromotionLoader interface.
- Implement supports method in a way that the plugin is only used in the desired context.
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the AdhocPromotionLoader.

HOW IT WORKS

Ad hoc promotions are loaded and saved to the PCE context before the eligibility and condition loaders are executed. The loading of ad hoc is performed by the method loadAdhocPromotions in class AdhocPromotionHandlerImpl (com.gk_software.pricing_engine.core.engine.AdhocPromotionHandlerImpl), which is the PCE core implementation for the AdhocPromotionHandler interface (com.gk_software.pricing_engine.api.engine.AdhocPromotionHandler). Execution is done by using the Strategy extension pattern, which means all implementing classes for the plugin will be called and the ones that are designed to take action will execute the corresponding logic.

THERE IS MORE The PCE core implementation does not provide any AdhocPromotionLoader plugins. However, all custom plugin implementations added to the Spring application context will be processed and considered as explained.

6.2.4 Condition Calculation

6.2.4.1 Validate a condition before processing

Before processing any conditions loaded from the data access layer, it is necessary to validate the retrieved promotion data before the PCE processing can be started. If a condition is found invalid, the calculation of the condition is aborted and the corresponding partial result's status is set to invalid.

The ConditionValidator extension point with PromotionConditionWrapper (com.gk_software.pricing_engine.api.rules.PromotionConditionWrapper) as supports context is provided by the PCE for performing such validations.

The ConditionValidator is defined in the interface ConditionValidator (com.gk_software.pricing_engine.api.calculation.ConditionValidator).

GETTING READY

For adding custom condition validators, the following steps must be taken:

- Add custom implementation of the ConditionValidator interface.
- Implement supports method in a way that the plugin is only used in the desired context (PromotionConditionWrapper).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the ConditionValidator.

HOW IT WORKS

The condition validation is performed by the method isConditionValid in class PartialResultCalculatorImpl (com.gk_software.pricing_engine.core.engine.PartialResultCalculatorImpl), which is the PCE core implementation for the PartialResultCalculator interface (com.gk_software.pricing_engine.common.result.PartialResultCalculator). The condition is validated right after the partial result was initialized, prohibiting any further calculation efforts in case of invalid conditions. The validation is performed in case of position- and transaction-related conditions as well. The execution is based on the Filter Chain extension pattern, which means all registered plugins that can be applied in the given context must deem the condition valid.

THERE IS MORE

The following PCE core implementations are provided for the ConditionValidator plugin:

Plugin	Supports criteria	Plugin order	Description
ConditionSequenceAllowedValidator	Supports transaction related conditions only	1000	Verifies whether the given condition can be applied by checking the external promotions and the maximum applicable sequence.

Plugin	Supports criteria	Plugin order	Description
ExclusivePromotionCombinationValidator	Supports transaction related conditions only	2000	Verifies whether the combination with any available exclusive promotion is valid.
SaleReturnTypeConditionValidator	Supports transaction related conditions only	3000	Determines if the transaction related condition is valid based on its SaleReturnTypeCode and the transaction's TransactionCategoryCode.

6.2.4.2 Filter valid normalized items for condition before processing

It might be necessary to filter out normalized items from the working bucket of the condition calculation based on custom flags or any other requirements. The RebatableItemFilterPlugin extension point provides an interface for such filter implementations. The RebatableItemFilterPlugin is defined in the interface RebatableItemFilterPlugin (com.gk_software.pricing_engine.api.filter.plugin.RebatableItemFilterPlugin) with the supports context PromotionConditionWrapper (com.gk_software.pricing_engine.api.rules.PromotionConditionWrapper).

GETTING READY

For adding RebatableItemFilterPlugins, the following steps must be taken:

- Add custom implementation of the RebatableItemFilterPlugin interface.
- Implement supports method in a way that the plugin is only used in the desired context (PromotionConditionWrapper).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the RebatableItemFilterPlugin.

HOW IT WORKS

The working bucket of eligible normalized items is filtered for each condition as the PartialResult created for the calculation is initialized. The filtering is performed for both position- and transaction-related PartialResults and only RebatableItemFilterPlugins supported in the current context will be executed. The plugin execution follows the Filter Chain extension pattern, which means all filter plugins registered to the Spring application context which are supported must accept a given normalized item to be considered in the calculation.

THERE IS MORE

Plugin	Supports criteria	Plugin order	Description
SaleReturnTypeItemFilter	Supports all promotion condition wrappers	1000	Determines if a given normalized item can be used for applying the condition, based on the condition's SaleReturnTypeCode and the CalculationBase.

6.2.4.3 Validate exclusive condition check handling

It is possible to overwrite the default handling of exclusive conditions provided by the PCE core implementation. An implementation of the ExclusiveCheckHandler extension point should perform all related checks: whether a condition is defined as exclusive and whether an exclusive condition can be applied in the given context for both position- and transaction-related calculation.

The ExclusiveCheckHandler extension point with PromotionConditionSO (com.gk_software.pricing_engine.model.md.promotion.PromotionConditionSO) as supports context is defined in the interface ExclusiveCheckHandler (com.gk_software.pricing_engine.api.calculation.ExclusiveCheckHandler).

GETTING READY

For adding custom exclusive check handlers, the following steps must be taken:

- Add custom implementation of the ExclusiveCheckHandler interface.
- Implement supports method in a way that the plugin is only used in the desired context (PromotionConditionSO).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the EligibilityRuleHandler.

HOW IT WORKS

The ExclusiveCheckHandler plugins registered to the application context are executed using the Chain of Responsibility pattern, meaning that the first supported plugin found will be used for checking exclusivity.

THERE IS MORE

The following PCE core implementations are provided for the ExclusiveCheckHandler plugin:

Plugin	Supports criteria	Plugin order	Description
DefaultExclusiveCheckHandler	Supports all promotion conditions	1000	Default exclusive check handler.

6.2.4.4 Process intervals on custom threshold eligibilities

The EligibilityRuleHandler extension point provides an interface for plugins that handle the execution of rules in regard of item-related threshold type eligibilities. It takes care of processing interval calculation and handles the thresholds and limits defined in the promotion data. The implementation for the plugin always relates to a specific promotion condition eligibility, as it uses the eligibility type-specific information saved to the partial result during eligibility parsing. The implementation of this extension point is required if the PCE is extended with a new threshold type eligibility.

HOW IT WORKS

The EligibilityRuleHandler extension point with PartialResult (com.gk_software.pricing_engine.api.result.PartialResult) as supports context is defined in the interface EligibilityRuleHandler (com.gk_software.pricing_engine.api.rules.EligibilityRuleHandler).

The EligibilityRuleHandler is implemented in the DefaultEligibilityRuleHandler. The DefaultEligibilityRuleHandler is responsible for handling all common logic to be applied to a position or transaction related rules for simple and interval eligibilities. Eligibility type specific logic is handled by eligibility rule handler fragments (com.gk_software.pricing_engine.api.rules.EligibilityRuleHandlerFragment).

GETTING READY

For adding custom eligibility rule handlers and eligibility rule handler fragments, the following steps must be taken:

- Add custom implementation of the EligibilityRuleHandler or the EligibilityRuleHandlerFragment interface.
- Make sure that the getEligibilityType method returns the type of the supported eligibility.
- Implement supports method in a way that the plugin is only used in the desired context (PartialResult).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context.

THERE IS MORE

The following PCE core implementations are provided for the EligibilityRuleHandler plugin:

Fragment	Supports criteria	Order	Description
DefaultEligibilityRuleHandler	PartialResults that are supported by at least one of the registered eligibility rule handler fragments	1000	The eligibility rule handler fragment is responsible for handling general, non eligibility-specific rule handling

The following PCE core implementations are provided for the EligibilityRuleHandlerFragment plugin:

Fragment	Supports criteria	Order	Description
ItemEligibilityRuleHandlerFragment	PartialResults that used eligibility with type "ITEM" in the calculation	1000	The eligibility rule handler fragment is responsible to handler specific logic related to item eligibilities.
MhgEligibilityRuleHandlerFragment	PartialResults that used eligibility with type "MSTR" in the calculation	2000	The eligibility rule handler fragment is responsible to handler specific logic related to MHG eligibilities.
MerchandiseSetEligibilityRuleHandlerFragment	PartialResults that used eligibility with type "MSET" in the calculation	3000	The eligibility rule handler fragment is responsible to handler specific logic related to merchandise set eligibilities.
CombinationEligibilityRuleHandlerFragment	PartialResults that used eligibility with type "COMB" in the calculation	4000	The eligibility rule handler fragment is responsible to handler specific logic related to combination eligibilities.

6.2.4.5 Calculate promotions for new rule types

The CalculationRuleProcessor extension point is available for implementing the rule type-specific promotion condition rule application to grant discounts. The rule processor plugins must implement the logic for applying position- and transaction-related rules as well provide further rule type-specific information that can be used in the calculation.

The CalculationRuleProcessor extension point with PromotionConditionRuleSO (com.gk_software.pricing_engine.model.md.promotion.PromotionConditionRuleSO) as supports context is defined in the interface CalculationRuleProcessor (com.gk_software.pricing_engine.api.rules.CalculationRuleProcessor).

GETTING READY

For adding a calculation rule processor plugin for a new custom rule type, the following steps must be taken:

- Add custom implementation of the CalculationRuleProcessor interface.
- Implement supports method in a way that the plugin is only used in the desired context, so that it only supports the new rule type.
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the CalculationRuleProcessor interface.

HOW IT WORKS

The CalculationRuleProcessor registered to the application context is executed using the Chain of Responsibility extension pattern and called from the EligibilityRuleHandler plugin implementation used for the current calculation. This means that the first CalculationRuleProcessor found supporting the processed rule will execute its corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the CalculationRuleProcessor plugin:

Plugin	Supports criteria	Plugin order	Description
SimpleDiscountCalculationRule	Rules with type code "RB"	1000	Calculation rule class for simple discounts rule.
ManualPriceCalculationRule	Rules with type code "MA"	2000	Calculation rule class for manual price discounts rule.
MixAndMatchCalculationRule	Rules with type code "MM"	3000	Calculation rule class for mix and match discounts rule.
ExternalActionCalculationRule	Rules with type code "EX"	4000	Calculation rule class for external discounts rule.
NoDiscountCalculationRule	Rules with type code "NO"	5000	Calculation rule class for no discount rule.

6.2.4.6 Prorate rebates for new rule types

Newly introduced rule types may require rebate distribution handling to be different from the PCE default. The RebateDistributionProcessor extension point is available for implementing custom proration logic.

The RebateDistributionProcessor extension point with PromotionConditionRuleSO (com.gk_software.pricing_engine.model.md.promotion.PartialResult) as supports context is defined in the interface RebateDistributionProcessor (com.gk_software.pricing_engine.api.rules.RebateDistributionProcessor).

GETTING READY

For adding a rebate proration plugin for a new custom rule type, the following steps must be taken:

- Add custom implementation of the RebateDistributionProcessor interface.
- Implement supports method in a way that the plugin is only used in the desired context, so that it only supports the new rule type.
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the RebateDistributionProcessor interface.

HOW IT WORKS

If rebate proration is necessary for transaction-related results or total position-related conditions, the RebateDistributionProcessors registered to the application context will be executed using the Chain of Responsibility extension pattern. This means that the first RebateDistributionProcessor found supporting the processed rule will execute its corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the RebateDistributionProcessor plugin:

Plugin	Supports criteria	Plugin order	Description
SimpleDiscountRebateDistributionService	Rules with type code "RB"	1000	Distribution service for simple discount rules.
ManualPriceRebateDistributionService	Rules with type code "MA"	2000	Distribution service for manual rules.
MixAndMatchRebateDistributionService	Rules with type code "MM"	3000	Distribution service for mix and match rules.
ExternalActionRebateDistributionService	Rules with type code "EX"	4000	Distribution service for external action rules.
NoDiscountRebateDistributionService	Rules with type code "NO"	5000	Distribution service for no discount rebate rules.

6.2.4.7 Prorate MixAndMatch on a custom item set

The MixAndMatchProrationCustomizer extension point allows customization of the MixAndMatch Rebate proration. The MixAndMatchProrationCustomizer extension point is defined in the interface MixAndMatchProrationCustomizer (com.gk_software.pricing_engine.api.rules.MixAndMatchProrationCustomizer).

GETTING READY

For adding a mix and match proration customizer, the following steps must be taken:

- Add custom implementation of the MixAndMatchProrationCustomizer interface.
- Implement supports method in a way that the plugin is only used in the desired context.
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the MixAndMatchProrationCustomizer interface.

HOW IT WORKS

The Chain of Responsibility extension pattern is used to execute the MixAndMatchProrationCustomizer plugins, which means that the first Customizer registered to the application context that returns true on supports method will be used. Thus, all proration customizer plugin implementations should be annotated with Spring @Order annotation to define order of appliance. If no supported customizer is found, standard proration on the matching item is executed by the default MixAndMatchProrationCustomizer provided by the PCE core.

THERE IS MORE

The PCE core implementation does not provide any MixAndMatchProrationCustomizer plugins. However, all custom plugin implementations added to the Spring application context will be processed and considered as explained.

6.2.4.8 Introduce new PriceModification codes

As a custom requirement it might be necessary to introduce a new price calculation method for promotion calculation rules and a new method code for it. Each price modification methods supported by the PCE is implemented as a PriceModification plugin.

The PriceModification extension point with a PromotionConditionRuleSO (com.gk_software.pricing_engine.model.md.promotion.PromotionConditionRuleSO) as supports context is defined in the interface PriceModification (com.gk_software.pricing_engine.api.price_modification.PriceModification).

GETTING READY

For adding a custom price modification method code, the following steps must be taken:

- Extend the PriceModificationMethod interface (com.gk_software.pricing_engine.model.typecodes.promotion.PriceModificationMethod) with the new method code enum value.
- Add custom implementation of the PriceModification interface. The abstract AbstractPriceModification super class with common methods used between price modification plugins is available in the core PCE implementation and can be utilized by new plugins.
- Implement supports method in a way that the plugin is only used in the desired context, with the desired price modification type code (PromotionConditionRuleSO).
- Mark plugin with @Order annotation for predictable execution, if necessary.
- Add bean to the Spring application context for the custom implementation of the PriceModification.

HOW IT WORKS

The price modification plugin implementations perform the calculation of the actual rebate to be granted for the applied condition. All price modification plugins must handle both position- and transaction-related condition calculations as well as monetary rebate and frequent shopper point calculation. Total rebate and rebate share value calculation should also be handled if supported. The different price modification modes available for each price modification method code are listed in the enums PositionPriceModificationMode and TransactionPriceModificationType (defined in package com.gk_software.pricing_engine.api.price_modification). All of these must be handled by the plugin implementations. If a price modification mode for position-related calculation or a transaction price modification type for transaction-related calculation is not supported for the price modification code, then an UnsupportedOperationException should be thrown.

The price modification plugins registered to the Spring application context are executed using the Chain of Responsibility mode, which means the first found, supported plugin will execute the corresponding logic. If multiple implementations are registered for the same price modification method code, the use of the @Order annotation is crucial for consistent, predictable calculation results.

THERE IS MORE

The following PCE core implementations are provided for the PriceModification plugin:

Plugin	Supports criteria	Plugin order	Description
PercentageDiscountPriceModification	Rules with price modification method code discount percent ("RP")	1000	Price modification methods for percentage discounts.
AbsoluteDiscountPriceModification	Rules with price modification method code rebate single ("RS")	2000	Price modification methods for absolute discounts.
NewPricePriceModification	Rules with price modification method code fixed price ("PS")	3000	New price modification methods.
NewTotalPricePriceModification	Rules with price modification method code fixed price total ("PT")	4000	Price modification methods for new total price discounts.
TotalAbsoluteDiscountPriceModification	Rules with price modification method code discount total ("RT")	5000	Price modification methods for total absolute discounts.
TotalPercentageDiscountPriceModification	Rules with price modification method code discount percent total ("TP")	6000	Price modification methods for total percentage discounts.
NewSetPricePriceModification	Rules with price modification method code set total price ("ST")	7000	New set total modification calculation.
TotalPercentageDiscountRoundedTwicePriceModification	Rules with price modification method code discount percent total 2 ("T2")	8000	Price modification methods for total percentage discounts.

6.2.4.9 Customize best price determination

The best price calculation algorithm is used only in case that the calculation of two or more valid conditions with the same sequence and resolution dependent on each other have resulted in valid partial results. In this case, the order of the applied conditions may affect the final total benefit. The goal of the best price calculation is to always provide the best possible price in case of collisions. The best price calculator extension point makes it possible to use a custom algorithm for finding the best price.

The BestPriceCalculator extension point with a PartialResult list (com.gk_software.pricing_engine.api.result.PartialResult) as supports context is defined in the interface BestPriceCalculator (com.gk_software.pricing_engine.api.calculation.BestPriceCalculator).

GETTING READY

For adding a custom best price calculator, the following steps must be taken:

- Add custom implementation of the BestPriceCalculator interface.

- Implement supports method in a way that the plugin is only used in the desired context (PartialResult list).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the BestPriceCalculator.

HOW IT WORKS

The best price calculation is performed as part of the conflict handling execution by the class DefaultConflictHandler, the core PCE implementation of the ConflictHandler interface (com.gk_software.pricing_engine.core.engine.DefaultConflictHandler). The BestPriceCalculator plugins registered to the Spring application context are executed using the Chain of Responsibility extension pattern, which means the order defined for the plugins controls which plugin will be used to execute the corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the BestPriceCalculator plugin:

Plugin	Supports criteria	Plugin order	Description
DefaultBestPriceCalculator	Supported in any context	1000	Default Best Price checking. Checks the best price is used if the price of a partial calculation is lower than the current best price.

6.2.4.10 Validate the result of a calculation

After calculating the partial result for each condition, a validation of the result is performed. The ResultValidator extension point provides a plugin interface for these validators. The partial result is validated the same way for both position- and transaction-related conditions.

The ResultValidator extension point with a PartialResult (com.gk_software.pricing_engine.api.result.PartialResult) as supports context is defined in the interface ResultValidator (com.gk_software.pricing_engine.api.result.ResultValidator).

GETTING READY

For adding a custom result validator, the following steps must be taken:

- Add custom implementation of the ResultValidator interface.
- Implement supports method in a way that the plugin is only used in the desired context (PartialResult).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the ResultValidator.

HOW IT WORKS

The partial result validation is performed by the validatePartialResult method of the PartialResultCalculatorImpl class, the PCE core implementation of the PartialResultCalculator interface (com.gk_software.pricing_engine.core.engine.PartialResultCalculatorImpl). The result of each successfully calculated position- and transaction-related condition is validated. However, this step is not performed if the partial result was already invalidated during calculation. The result validator plugins registered to the Spring application context are executed using the Filter Chain extension pattern. Therefore, the calculated result must be deemed valid by all supported plugins in order for the condition to be applied.

THERE IS MORE

The following PCE core implementations are provided for the ResultValidator plugin:

Plugin	Supports criteria	Plugin order	Description
ResultActualPurchaseBaseValidator	Only supports TransactionPartialResult	1000	Verifies whether actual purchase base is valid for applying transaction rebate. This method compares the actual purchase base with the market basket threshold amount from all market basket amount eligibilities active for this condition.
ResultCouponCountValidator	Supported in any context	2000	Verifies whether there are enough coupons available for using condition.
PositionResultInfluencedPositionValidator	Only supports PositionPartialResult	3000	Verifies whether the position partial result calculation has influenced any of the positions.
TransactionResultInfluencedPositionValidator	Only supports TransactionPartialResult	4000	Verifies whether the transaction partial result calculation has influenced any of the positions.
PositionResultConditionAmountValidator	Only supports PositionPartialResult	5000	Verifies whether the position partial result's condition value is allowed. The amount of the rebate or the points is allowed if it is greater than zero or if the zero discount is allowed.
TransactionResultConditionAmountValidator	Only supports TransactionPartialResult	6000	Verifies whether the transaction partial result's condition value is allowed. The amount of the rebate or the points is allowed if it is greater than zero or if the zero discount is allowed.

6.2.4.11 Validate applied rebates

It might be necessary to add additional validation for the applied rebate, based on its amount or any other information available in the PartialResult after successful calculation.
The RebateValidator extension point allows such custom validations.

The RebateValidator extension point with a PartialResult (com.gk_software.pricing_engine.api.result.PartialResult) as supports context is defined in the interface RebateValidator<S extends SummaryResult> (com.gk_software.pricing_engine.api.result.RebateValidator).

GETTING READY

For adding a custom rebate validator, the following steps must be taken:

- Add custom implementation of the RebateValidator interface.
- Implement supports method in a way that the plugin is only used in the desired context (PartialResult).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the RebateValidator.

HOW IT WORKS

The RebateValidator plugins registered to the application context are called using the Filter Chain extension pattern after a successful PartialResult calculation, but before the result would have been saved to the SummaryResult. The calculated rebate can only be applied this way, if it was deemed valid by all registered rebate validators.

THERE IS MORE

The PCE core implementation does not provide any RebateValidator plugins. However, all custom plugin implementations added to the Spring application context will be processed and considered as explained.

6.2.4.12 Customize Response Values

Sometimes you would like to add additional information to the user by customizing the returned response. This can be an additional description taken from the database or adding additional information based on the calculation result. For this purpose, the PCE allows you to customize the resulting elements created for the response.

For each of these elements a specialization of ResultDtoCustomizer exists that allow modifying the created elements.

The following elements are created by the PCE in the transaction for the response and the related customizer interface:

- FrequentShopperPointsModifier: FrequentShopperPointsModifierCustomizer
- LoyaltyRewardLineItem: LoyaltyRewardLineItemCustomizer
- RetailPriceModifier: RetailPriceModifierCustomizer
- PriceModificationLineItem: PriceModificationLineItemCustomizer
- RetailTransactionPromotionPriceDerivationRule: RetailTransactionPromotionPriceDerivationRuleCustomizer

GETTING READY

For adding a result customizer, the following steps must be taken:

- Add custom implementation of the corresponding ResultDtoCustomizer interface.
- Implement supports method in a way that the plugin is only used in the desired context (PartialResult).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the ResultDtoCustomizer.

HOW IT WORKS

The ResultDtoCustomizer plugins registered to the application context are managed by the ResultCustomizerService bean. They are called using the Strategy extension pattern after a successful PartialResult calculation, when the result is saved to the SummaryResult. All configured ResultDtoCustomizer that are matching the given PartialResult context are called to customize the result. This means they could override each other's values, so be aware when using multiple custom ResultDtoCustomizers.

Here is an example how to customize the "*reasonDescription*" property of a RetailPriceModifier based on some extension field stored in PromotionCondition.

In line 7, we check whether the PromotionCondition related to the PartialResult contains a String attribute with key "*extendedReason*". If this is true, then the customize method is called by the ResultCustomizerService.
In line 13, we set the value from the "*extendedReason*" attribute to the "*reasonDescription*" property of the created RetailPriceModifier.

```
public class RetailPriceModifierReasonDescriptionCustomizer implements
RetailPriceModifierCustomizer {

  private final static String EXTENDED_REASON = "extendedReason";

  @Override
  public boolean supports(PartialResult<?> partialResult) {
    return partialResult.getConditionWrapper().getPromotionCondition().get(EXTENDED_REASON)
instanceof String;
```

```
  }

  @Override
  public void customize(Context context, RetailPriceModifier modelObject, PartialResult<?>
partialResult,
      LineItemSummaryResult lineItemSummaryResult) {
    modelObject.setReasonDescription((String)
partialResult.getConditionWrapper().getPromotionCondition().get(EXTENDED_REASON));
  }
}
```

THERE IS MORE

The PCE core implementation does not provide any ResultDtoCustomizer plugins. However, all custom plugin implementations added to the Spring application context will be processed and considered as explained.

6.2.5 Custom field mapping

6.2.5.1 Mapping from any fields (Transaction Model)

The PCE provides two extension points for mapping the XXCustom[0-15] fields in the GK model from any fields in the PCE transaction model. By implementing plugins for these extension points it is possible to store such custom fields in the PCE internal model and access and use them during calculation.

The ModelExtractor extension point with TransactionGk (com.gk_software.pricing_engine.gk.model.transaction.TransactionGk) as supports context is defined in the interface GKModelExtractor (com.gk_software.pricing_engine.gk.model.plugin.GKModelExtractor). It is responsible for mapping the any fields to XXCustom[0-15] fields.

GETTING READY

For adding a new custom field mapper plugin, the following steps must be taken:

- Add custom implementation of the corresponding GKModelExtractor interface for implementing the correct mapping from any fields to the XXCustom fields.
- Implement supports method in a way that the plugin is only used in the desired context (TransactionGk).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the extension point.

HOW IT WORKS

Execution is done by using the Strategy extension pattern, which means all implementing classes for the plugin will be called and the ones that are designed to take action will execute the corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the GKModelExtractor plugin:

Plugin	Supports criteria	Plugin order	Description
GKModelExtractorImpl	All non-null transactions are supported	1000	Default GK model extractor responsible to map any fields to XXCustom[0-15] fields.

6.2.5.2 Mapping to any fields (Transaction Model)

The PCE provides two extension points for mapping the XXCustom[0-15] fields in the GK transaction model to any fields in the PCE transaction model. By implementing plugins for these extension points it is possible to provide custom values in the response.

The GKModelAdapter extension point with TransactionGk (com.gk_software.pricing_engine.gk.model.transaction.TransactionGk) as supports context is defined in the interface GKModelAdapter (com.gk_software.pricing_engine.gk.model.plugin.GKModelAdapter). It is responsible for mapping the custom values in the opposite direction: the GK XXCustom[0-15] fields into any fields.

GETTING READY

For adding a new custom field mapper plugin, the following steps must be taken:

- Add custom implementation of the corresponding GKModelAdapter interface for implementing the correct mapping from the XXCustom fields to any fields.
- Implement supports method in a way that the plugin is only used in the desired context (TransactionGk).
- Mark plugin with @Order annotation for predictable execution.
- Add bean to the Spring application context for the custom implementation of the extension point.

HOW IT WORKS

Execution is done by using the Strategy extension pattern, which means all implementing classes for the plugin will be called and the ones that are designed to take action will execute the corresponding logic.

THERE IS MORE

The following PCE core implementations are provided for the GKModelAdapter plugin:

Plugin	Supports criteria	Plugin order	Description
GKModelAdapterImpl	All non-null transactions are supported	1000	Default GK model adapter responsible to map GK XXCustom[0-15] fields into any fields.

6.3 Custom Extension Samples

The PCE standard delivery includes a lot of commonly used features. However custom enhancements might be requested, e.g. making customer specific promotions available or performing custom validation checks. This chapter describes through concrete examples how such a customer specific feature could be implemented using the Extension Concept provided by the PCE.

6.3.1 Plugin implementation of the PCE part

In the scope of the plugin-based extension of the PCE part, a specific plugin is implemented for each logical step of the processing of the transaction according to the corresponding eligibilities and rules. Different plugins are implemented for different types of promotions. As an example of plugins for a specific type of promotion, one of the implemented plugins realizes the transaction analyzer logic, the other one handles the loading of the related eligibilities and registration of them into the plugin context and another one handles the processing of the rules and calculation of the rebate according to these rules.

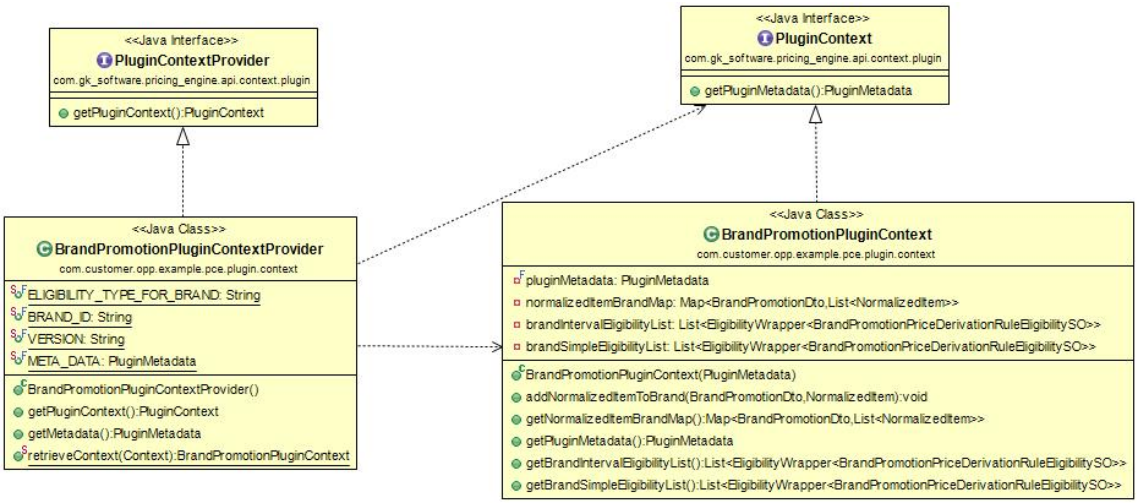
The implemented plugins for each type of promotion can share a number of specific contexts which are called plugin contexts. A plugin context holds the transaction-related data specific to the corresponding type of promotion. This data can include the normalized line items created for internal processing by using the retail transaction line items, and the simple and interval eligibilities which are activated for the transaction. The plugin registry context of the application context stores the plugin contexts for different types of promotions.

The implementation of the processing of brand promotions is integrated into PCE based on the introduced plugin concept. In order to avoid unnecessary coupling of implementation classes and hierarchies, the development of this plugin has focused on the PCE API level interfaces. On this way, each exposed class/interface implements or extends a related PCE API interface when necessary. For instance, the brand promotion eligibility is modeled by the BrandPromotionPriceDerivationRuleEligibilitySO interface and implemented by the BrandPromotionPromotionPriceDerivationRuleEligibilityDefaultSO class. Here in order to avoid unnecessary coupling of classes, the implementation contains all known items (fields) related to the eligibility object. The implementation is in general only dependent of PCE data access, PCE core API, model, common, standard java classes and the related Spring Framework libraries.

In order to make the comprehension of the plugin-based processing of the transactions concrete, the following items describe the plugins created for handling brand promotions during transaction processing.

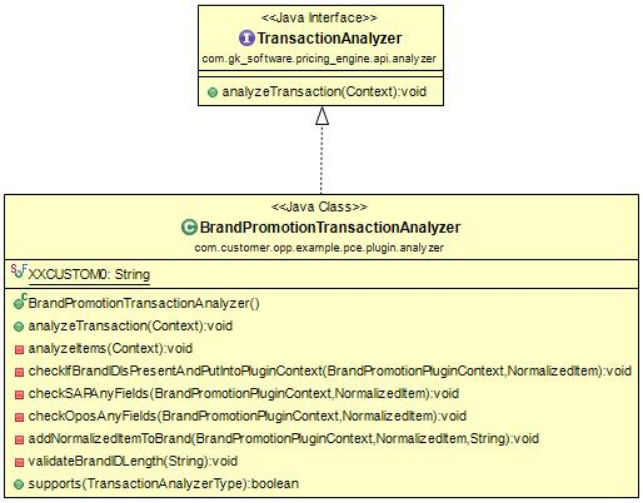
6.3.1.1 1. Initialization of Brand Promotion Plugin Context

The brand promotion plugin context provider object (an instance of the BrandPromotionPluginContextProvider class) is responsible for creating and initializing the brand promotion plugin context shared by the brand promotion plugins. This is performed during the registration of the plugin contexts with the application context. The brand promotion plugin context (an instance of the BrandPromotionPluginContext class) stores the normalized items for the current transaction mapped by their brands, and the activated brand promotion eligibilities for the transaction. Interval and simple eligibilities are stored in different lists. The brand promotion plugin context instance can be retrieved by the other brand promotion plugins via calling a specific method of the brand promotion plugin context provider object. The below figure shows the corresponding classes and their relationships for creating and retrieving the brand promotion plugin context.



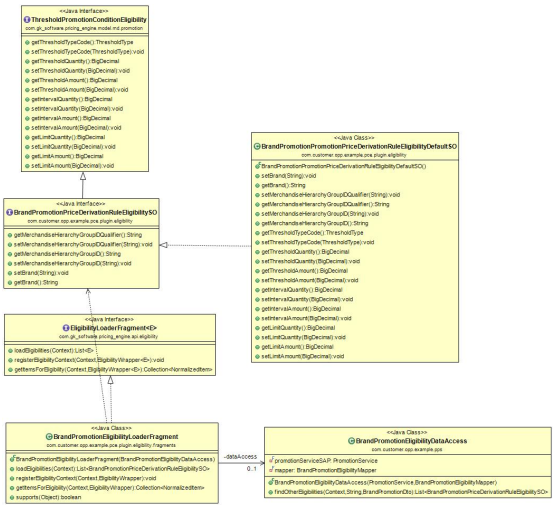
6.3.1.2 2. Brand Promotion Transaction Analyzer

The plugin for analyzing transactions for brand promotions (an instance of the BrandPromotionTransactionAnalyzer class) is responsible for selecting the appropriate normalized line items according to the corresponding brand by querying the transaction analyzer context stored by the application context. The selected normalized line items are mapped according to their line item keys and stored into the brand promotion plugin context. The below figure shows the BrandPromotionTransactionAnalyzer class and the interface implemented by it.



6.3.1.3 3. Brand Promotion Eligibility Loader Fragment

The eligibility loader fragment for brand promotions (an instance of the BrandPromotionEligibilityLoaderFragment class) is responsible for loading brand promotion eligibilities into PCE by querying the master data, filtering the normalized items for filling the item buckets of the corresponding eligibility wrappers and registering the activated brand promotion eligibilities into the brand promotion plugin context. The BrandPromotionEligibilityLoaderFragment class implements the EligibilityLoaderFragment interface which declares the general operations for the eligibility loader fragments. The brand promotion eligibility loader fragment uses an instance of the BrandPromotionEligibilityDataAccess class to query the master data for loading the brand promotion eligibilities. The brand promotion eligibilities is modeled by the BrandPromotionPriceDerivationRuleEligibilitySO interface and the BrandPromotionPriceDerivationRuleEligibilityDefaultSO class, which implements this interface and contains the brand field and setter/getter methods for this field as shown in the below class diagram.

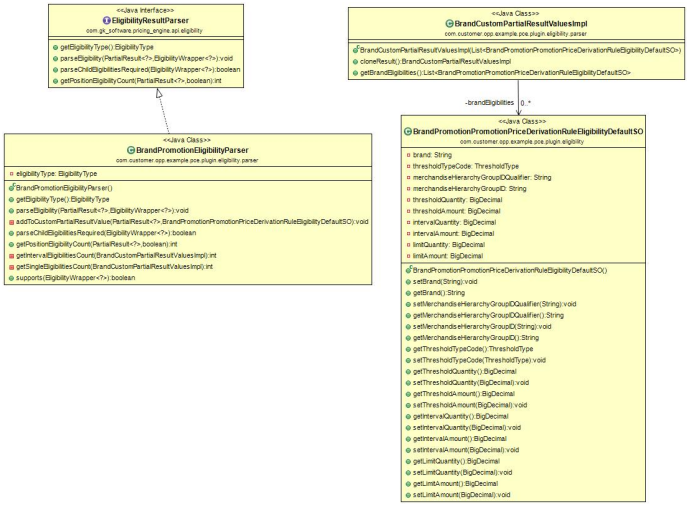


6.3.1.4 4. Brand Promotion Eligibility Rule Handler

The rule handler plugin for brand promotions is implemented by the BrandPromotionEligibilityRuleHandler class. It implements the logic for the application of the position and transaction rules for simple and interval brand promotion eligibilities. It implements the EligibilityRuleHandler interface, which declares the general operations for the eligibility rule handlers. The diagram below shows this class and the EligibilityRuleHandler interface.

6.3.1.5 5. Brand Promotion Eligibility Parser

The eligibility plugin parser is useful at the calculation time to inform the rule handler if there is simple or interval eligibilities related to brand promotion to apply for the current partial result. These information is stored into the partial result as extended custom partial result values to be carry through during the calculation process.



Customer Extension 1.0 (2).pdf

6.3.2 Concurrence Control Vector Extension Sample

The PCE's exclusive promotion price derivation rule checking feature provided in the default delivery enables the customer to define which promotions can be applied on top of each other and which cannot be controlled by the promotion price derivation rule's exclusiveFlag. The default exclusivity checks are implemented by the DefaultExclusiveCheckHandler plugin. As a custom requirement, it might be necessary however to further detail the exclusion criteria. The ExclusiveCheckHandler interface provides an extension point for such custom plugins, as described in chapter Extension Points. Furthermore, the PCE internal data model's PromotionPriceDerivationRule.ConcurrenceControlVector field serves as an exclusion control vector to allow further customization. The ConcurrenceControlVector consists of 10 digits. Each digit serves as a flag for the exclusion of discount.

The following example shows a possible extension of the PCE to handle custom exclusions. The ConcurrenceControlVectorExclusiveCheckHandler plugin implements the ExclusiveCheckHandler interface and utilizes the ConcurrenceControlVector to apply custom exclusion rules. The ConcurrenceControlVectorExclusiveCheckHandler plugin is not part of the standard delivery, but can be used as a template for fulfilling custom requirements.

The ExclusiveCheckHandler plugins registered to the application context are executed using the Chain of Responsibility pattern, meaning that the first supported plugin found will be used for checking exclusivity. The Order annotation of the custom plugin must be less than the 1000, which is the order of the default implementation.

HOW IT WORKS

The ConcurrenceControlVectorExclusiveCheckHandler processes the ConcurrenceControlVector with the following meaning for each digit to determine whether a given promotion price derivation rule can be applied or not. Only the below listed values and digits are used in this example. For more complex requirements the remaining digits and values may also be utilized.

Digit	Value	Explanation
0	0	A line item-related promotion price derivation rule must not be deleted by an exclusive transaction-related promotion price derivation rule.
	1	A line item-related promotion price derivation rule is deleted by an exclusive transaction-related promotion price derivation rule. Same if ConcurrenceControlVector is null.
	2	A transaction-related promotion price derivation rule does not delete line item discounts.
	3	A transaction-related promotion price derivation rule deletes line item discounts. Same if ConcurrenceControlVector is null.
1	0	A transaction-related promotion price derivation rule can be combined with exclusive and non-exclusive item-related promotion price derivation rules.
	1	The calculation base of a transaction-related promotion price derivation rule considers only line items for which no exclusive promotion price derivation rules were applied before.
	2	The calculation base of a transaction-related promotion price derivation rule considers only line items for which any exclusive promotion price derivation rules were applied before.
	3	The calculation base of a transaction-related promotion price derivation rule considers only line items for which no promotion price derivation rules were applied before.
3	0	If an exclusive line item-related price derivation rule is applied, line item-related price derivation rules with a higher sequence cannot be applied. If an exclusive transaction-related price derivation rule is applied, transaction-related price derivation rules with a higher sequence cannot be applied.
	1	If an exclusive line item-related price derivation rule is applied, line item-related price derivation rules with a higher sequence can be applied anyway. If an exclusive transaction-related price derivation rule is applied, transaction-related price derivation rules with a higher sequence can be applied anyway.

The ExclusiveCheckHandler interface declares the following methods, which must be implemented by all custom exclusivity plugins.

Type	Method	Description	Used in the core implementation for
boolean	isExclusiveTransactionCondition (PromotionConditionSO currentCondition, PromotionConditionSO condition)	Checks if the promotion price derivation rule can be applied with previously applied exclusive transaction-related promotion price derivation rule. Parameter currentCondition is the promotion price derivation rule to apply. Parameter condition is the promotion price derivation rule which had been applied for the line item already.	Only checked for transaction-related promotion price derivation rule calculations. When initializing the transaction summary result, the check is performed for each position, to see if it can be used as trigger position and if it should be included in the calculation base. The method is called for each previously applied promotion price derivation rule and must return true for each for the current promotion price derivation rule to be used.
boolean	isExclusiveLineItemCondition (PromotionConditionSO condition)	Checks if the promotion price derivation rule is an exclusive line item-related promotion price derivation rule.	The check is performed before both line item-related and transaction-related calculations. The return value is required for the promotion price derivation rule calculation, the PCE Context's ExclusiveTransactionConditionUsed flag is set based on it.
boolean	canApplyConditionExclusiveCheck (Context context, SummaryResult summaryResult, PromotionConditionSO promotionCondition)	Checks if the line item-related promotion price derivation rule can be applied based on the previously already applied promotion price derivation rules.	The check is made before each promotion price derivation rule calculation, the calculation is only performed in case True is returned by the method.
boolean	canApplyExclusiveCondition (Context context, PromotionConditionSO condition, Long priceDerivationRuleEligibilityID)	Checks if the passed promotion price derivation rule can be applied based on the previously already applied promotion price derivation rules.	In case of line item-related promotion price derivation rule the check is made on each line item when performing the discountable line item check. It is performed against all previously applied price and points modifiers. In case of transaction-related calculation the check is made while validating the promotion price derivation rule, before the calculation.

For the sample implementation please refer to Appendix B.

6.3.3 Item Discount Control Vector Extension Sample

The PCE's item discount-ability filtering provided in the default delivery enables the customer to define which items may receive discounts or bonus points. The item filtering is performed by the RebatableLineItemSummaryResultBucketFilter, which executes all registered RebatableItemFilterPlugins.

As a custom requirement it might be necessary to apply additional acceptance criteria determining which items can receive benefits from a given promotion price derivation rule. The RebatableItemFilterPlugin interface provides an extension point for such custom plugins, as described in the previous chapter. In addition the PromotionConditionSO.ItemDiscountControlVector field together with the SaleReturnLineItem.DiscountTypeCode filed can be used for further customization. The ItemDiscountControlVector's digits are serving as flags which describe which discount DiscountTypeCodes are allowed for usage. The possible values of each digit are 0 meaning the discount is not possible to apply and 1 meaning the discount is possible to apply.

HOW IT WORKS

The ItemDiscountControlVector is not considered in the standard PCE delivery for determining the discountable items. The following example shows a possible extension of the PCE which implements custom item filtering based on the vector. The DiscountControlVectorFilterPlugin implements the RebatableItemFilterPlugin interface and utilizes the ItemDiscountControlVector for this purpose as follows.

If the discountVectorMethod configuration parameter's value is not "ALL", then the item is considered discountable. If the discountVectorMethod configuration parameter is set to "ALL", then the following applies:

DiscountTypeCode	ItemDiscountControlVector	Description
null	not checked	The item is considered discountable.
< 0	not checked	The item is not considered discountable.
> 9		
> ItemDiscountControlVector's length		
otherwise	0	The item is not considered discountable.
	1	The item is considered discountable.

For the sample implementation please refer to chapter Appendix B.

7 Troubleshooting

7.1 Business Error Codes

To inform the caller about input that is not well-formed or unexpected behavior while processing a request, Business Errors were defined for the different possible error scenarios. Business Error Codes with ErrorID and further details are sent back in the PriceCalculateResponse response object's ARTS header.

As the following example shows, the PriceCalculateResponse contains the original PriceCalculateBody element from the request message unmodified and a BusinessError element in the ARTSHeader with details of the error.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PriceCalculateResponse xmlns="http://www.sap.com/IXRetail/namespace/"
InternalMajorVersion="2" InternalMinorVersion="0">
    <ARTSHeader ActionCode="Calculate" MessageType="Response">
        <MessageID>ee2c780a-a44e-4c4e-a8e3-eede4484caff</MessageID>
        <DateTime>2017-04-12T16:14:22.483+02:00</DateTime>
        <Response ResponseCode="Rejected">
            <RequestID>9a89f2edfd1e413ea147e334b9c2ed4b</RequestID>
            <ResponseTimestamp>2017-04-12T16:14:22.482+02:00</ResponseTimestamp>
            <BusinessError Severity="Error">
                <ErrorID>GKR-100500</ErrorID>
                <Description>Value not supported. Element LineItem.SequenceNumber. Sent: 55.
Expected: 0.</Description>
            </BusinessError>
        </Response>
        <BusinessUnit TypeCode="RetailStore">OS01</BusinessUnit>
    </ARTSHeader>
    <PriceCalculateBody TransactionType="SaleTransaction" NetPriceFlag="true"
CalculationMode="LineItem">
        <TransactionID>9a89f2edfd1e413ea147e334b9c2ed4b</TransactionID>
        <DateTime>2501-03-09T00:00:00.000Z</DateTime>
        <ShoppingBasket>
            <LineItem>
                <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
                    <ItemID>ROC_SA_2</ItemID>
                    <RegularSalesUnitPrice Currency="EUR">24.55000</RegularSalesUnitPrice>
                    <Quantity Units="1" UnitOfMeasureCode="PCE">1</Quantity>
                    <TaxIncludedInPriceFlag>false</TaxIncludedInPriceFlag>
                    <NonPieceGoodFlag>false</NonPieceGoodFlag>

<FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
                    <PriceTypeCode>00</PriceTypeCode>
                    <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
                </Sale>
                <SequenceNumber>55</SequenceNumber>
```

```
                <MerchandiseHierarchy ID="1">RF16231</MerchandiseHierarchy>
            </LineItem>
            <LineItem>
                <Sale ItemType="Stock" NonDiscountableFlag="false" FixedPriceFlag="false">
                    <ItemID>ROC_SA_1</ItemID>
                    <RegularSalesUnitPrice Currency="EUR">25.21008</RegularSalesUnitPrice>
                    <Quantity Units="1" UnitOfMeasureCode="CT">1</Quantity>
                    <TaxIncludedInPriceFlag>false</TaxIncludedInPriceFlag>
                    <NonPieceGoodFlag>false</NonPieceGoodFlag>

<FrequentShopperPointsEligibilityFlag>true</FrequentShopperPointsEligibilityFlag>
                    <PriceTypeCode>00</PriceTypeCode>
                    <NotConsideredByPriceEngineFlag>false</NotConsideredByPriceEngineFlag>
                </Sale>
                <SequenceNumber>1</SequenceNumber>
                <MerchandiseHierarchy ID="1">RF16231</MerchandiseHierarchy>
            </LineItem>
        </ShoppingBasket>
    </PriceCalculateBody>
</PriceCalculateResponse>
```

Code Block 7 Error code (GKR-100500) in response, line 9

7.1.1 Error message format

The business errors are returned in the following format.

```
<error number> - <narrative> [LineItem(<SequenceNumber of the current line item>). ][Element
<name of the current element>(<number of the current element>). ][Sent: <sent content>.
[Expected: <expected content>.]]
```

7.1.2 Business Error Codes

The following table contains all defined error scenarios and the corresponding Business Errors with all information defined for them. Each BusinessError must have an ErrorID, a description of the scenario, and the expected and received content as well.

Error number	Narrative	LineItem info available?	Specific handling regarding to the element name	Sent content	Expected content
GKR-100500	Value not supported	x		The content of the element.	Enum where possible.
GKR-100501	Value too big	x		The length of the content of the element.	The maximum length.
GKR-100502	Too many decimal places	x		The number of decimal places of the content of the element.	The maximum count of decimal places.
GKR-100503	Value negative	x		The content of the element	Not available
GKR-100504	Structure wrong	Depends on where the error occurs.		The content of the element.	Not available.
GKR-100505	ID not unique	x		The content of the element.	Not available.
GKR-100506	Missing element or attribute	x	"Coupon"	"none"	"PrimaryLabel element"
		x	"Quantity"	"none"	"UnitOfMeasureCode attribute"
		x	"Coupon"	"none"	"RewardType element"
		x	"Coupon"	"none"	"RewardValue element"
		-	"ARTSHeader"	"none"	"MessageID element"
		x	"RetailPriceModifier"	"none"	"Quantity element"
		x	"PriceDerivationRuleEligibility"	"none"	"ReferenceID element"
		x	"PriceDerivationRuleEligibility"	"none"	"ReferenceSequenceNumber element"
		x	"ExternalAction"	"none"	"ID attribute"
		x	"ExternalActionText"	"none"	"element itself"
		x	"ExternalActionText"	"none"	"ID attribute"
		x	"ExternalActionParameter"	"none"	"element itself"
		x	"ExternalActionParameter"	"none"	"ID attribute"
		x	"FrequentShopperPointsModifier"	"none"	"AppliedQuantity element"
GKR-100507	Too many Quantity elements	x	Not available	The number of Quantity elements in the line item.	"1"

Error number	Narrative	LineItem info available?	Specific handling regarding to the element name	Sent content	Expected content
GKR-100508	Not existing or unknown	-	Not available	The content of the element	Not available
GKR-100509	Line item not allowed in line item calculation mode	x	"Coupon"	Not available	Not available
GKR-100510	External benefit not allowed in line item calculation mode	x		"SystemOriginatorFlag = true"	"SystemOriginatorFlag = false or missing"
GKR-100520	Calculation quantity limit reached	-	Not available	The total quantity of items in the corresponding request (calculated by considering units).	The maximum quantity of items allowed in a request that was configured.

7.2 Debugging

Sometimes you have to check what the PCE is doing internally. Either you are looking for a bug or try to see how an extension is used.

There are some typical use cases and situations that you will often encounter when checking the PCE. Here we describe some starting points in the code where you can begin your investigation.

7.2.1 Request/Response mapping

If a value is not mapped from request to response, this can pose a problem regarding the client API request mapping to GK model and/or response mapping to client API:

To debug the call regarding the client api request mapping:

DataMappingRequestMapper.mapRequest(PriceCalculate priceCalculate, Configuration configuration)

To debug the mapping to client api response:

DataResponseMapper.mapResponse(Configuration conf, Transaction transaction, PriceCalculate request)

These classes are the starting point to debug request and response mapping. Nevertheless, there are more related sub services that have the responsibility to map certain specific parts.

In the following table, you find more details.

7.2.1.1 Request mapping

All request mapper are found under the package:

com.gk_software.pricing_engine.psi.sapmapping.request.*

Reference and entry point to debug	Description
DataMappingRequestMapper.mapRequest(...)	Main entry point to start the request mapping. This class in composed by two sub services: DataMappingRequestHeader and DataMappingRequestBody.
DataMappingRequestHeader.mapARTSHeader(...)	Mapper responsible for mapping all information regarding ARTS headers.
DataMappingRequestBody.mapBody(...)	Mapper responsible for mapping the request body. This class checks also constraints regarding calculation mode LINE_ITEM and consistence of LineItem sequence numbers.

Request Body section

Reference and entry point to debug	Description
DataMappingRequestRetailTransactionCustomerGroupAssignment. mapCustomerGroup(...)	Mapper responsible for mapping the customer group.
DataMappingRequestLineItem.mapLineItems(...)	Starting point to map LineItems.

Into the LineItem mapping (DataMappingRequestLineItem), there is other sub service described below:

Reference and entry point to debug	Description
DataMappingRequestItemBase. mapLineItemChoiceBase(...)	Starting point to map line items: **DataMappingRequestRetailPriceModifier.mapRetailPriceModifiers(...)**: Responsible for mapping retail price modifiers. **DataMappingRequestPromotionPriceDerivationRuleReference.mapPromotionPriceDerivationRuleReference(...)**: Responsible for mapping the data regarding the price derivation rules. **DataMappingRequestFrequentShopperPointsModifier.mapFrequentShopperPointsModifier(...)**: Responsible for mapping the data regarding frequent shopper points. **DataMappingRequestMerchandiseHierarchyGroup.mapMerchandiseHierarchyCommonData(...):** Responsible for mapping the data regarding merchandise hierarchy. **DataMappingRequestSaleReturnLineItemPromotionTriggerType.mapPromotionManualTriggerTypeToSaleReturnLineItemPromotionTrigger(...)**: Responsible for mapping the data regarding promotion manual trigger. **DataMappingRequestRetailTransactionLineItemAssociation.mapRetailTransactionLineItemAssociation(...)**: Responsible for mapping the data regarding line item association (ItemLink).
DataMappingRequestPriceDerivationRuleEligibility. mapPriceDerivationRuleEligibility(...)	Responsible for mapping the data regarding a price derivation rule eligibility.
DataMappingRequestPriceDerivationRuleBase. mapPriceDerivationRuleBase(...)	Responsible for mapping the data regarding a price derivation rule base.
DataMappingRequestExternalActionText. mapExternalActionTextType(...)	Responsible for mapping the data regarding external action.
DataMappingRequestExternalActionParameter. mapExternalActionParameterType(...)	Responsible for mapping the data regarding external action.
DataMappingRequestLoyaltyReward. mapLoyaltyRewardBase(...)	Responsible for mapping the data regarding loyalty reward.
DataMappingRequestMerchandiseHierarchyGroup. mapMerchandiseHierarchyCommonData(...)	Responsible for mapping the data regarding merchandise hierarchy group.

Reference and entry point to debug	Description
DataMappingRequestPriceModificationLineItem. mapRetailPriceModifierFromDiscount(...)	Responsible for mapping the data regarding price modifications.
DataMappingRequestRetailTransactionCouponSummary. mapTenderCouponBase(...)	Responsible for mapping the data regarding a transaction coupon.
DataMappingRequestPromotionExternalTriggerType. mapPromotionExternalTriggerType(...)	Responsible for mapping the data regarding a promotion external trigger.
DataMappingRequestSaleReturnLineItemSalesOrder. mapSaleReturnLineItemSalesOrder(...)	Responsible for mapping the data regarding sales order.
DataMappingRequestRetailTransactionPromotionTrigger. mapPromotionManualTriggerTypeToRetailTransactionPromotionTrigger(...)	Responsible for mapping the data regarding transaction promotion trigger.

7.2.1.2 Response mapping

All response mappers are found under the package:

com.gk_software.pricing_engine.psi.sapmapping.response.*

Reference and entry point to debug	Description
DataResponseMapper.mapResponse(...)	Main entry point to start the response mapping. This class in composed by two sub services: DataResponseHeader and DataResponseBody.
DataResponseHeader.mapHeader(...)	Mapper responsible for mapping all header information back including response code and error messages if existing.
DataResponseBody.mapBody(...)	Mapper responsible for mapping the extended response body.

The response body section (DataResponseBody) is also composed by several mapper sub services:

Reference and entry point to debug	Description
DataResponseLineItemMapperService. mapLineItemItem(...)	Starting point to mapping back extended line items: **DataResponseRetailPriceModifierDomainSpecificMapperService.mapRetailPriceModifierDomainSpecific(...):** Responsible for mapping the data regarding retail price modifiers. **DataResponseFrequentShopperPointsModifierTypeMapperService.mapFrequentShopperPointsModifierType(...):** Responsible for mapping the data regarding frequent shopper points. **DataResponsePromotionPriceDerivationRuleReferenceTypeMapperService.mapPromotionPriceDerivationRuleReferenceType(...):** Responsible for mapping the data regarding price derivation rules. **DataResponsePromotionManualTriggerTypeMapperService.mapPromotionManualTriggerType(...):** Responsible for mapping the data regarding promotion manual triggers (LineItem level).
DataResponsePromotionManualTriggerTypeMapperService. mapPromotionManualTriggerType(...)	Responsible for mapping the data regarding promotion manual triggers (Transaction level).
DataResponseLoyaltyRewardService. mapLineItemLoyaltyReward(...)	Responsible for mapping the data regarding loyalty reward.
DataResponseCouponMapperService. mapTenderCouponBase(...)	Responsible for mapping the data regarding coupon.
DataResponseExternalTriggerMapperService. mapPromotionExternalTriggerTypeFromRequest(...)	Responsible for mapping the data regarding external triggers.
ReponseDataMapperCommons	Utility mapper that puts together all common things used by others mappers.

7.2.2 PCE Configuration

Sometimes results are not as expected. Therefore, is important to check some configurations to be sure and discard possible bug issues:

Symptom	To check
If an expected eligibility/promotion is loaded but not considered.	Check whether the validity dates are still valid. A starting debug point: **com.gk_software.pricing_engine.common.utils.DateUtils.isDateValid(...)** Additionally, you have to check the "**timeValidationMethod**" (see below).
If an already invalidated eligibility is still active.	Check the validity level stored in configuration property "**timeValidationMethod**" to find out which are the invalidation criteria. Possible values are: PROMOTION or ELIGIBILITIES. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**
If you change some eligibility/promotion value in the data base, but this change is not reflected in the calculation.	Check whether the caching property "**localMasterdataCacheEnabled**" with cache size "**cacheSize**" is active. An active cache means that all already loaded data from the Data Access Layer in a transaction are cached. The default cache size is 300. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in the PCE **Context** via the method **getConfiguration().**
If you get similar expected results that are not 100% according to your mathematical calculations.	Check the property "**rebateShareRoundingMethod**" which specifies the rounding strategy. Possible values are: NO_ROUNDING, HALF_UP, DOWN, UP and HALF_DOWN. See more details in enum com.gk_software.pricing_engine.model.typecodes.RoundingMethodEnum. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**
If your rebate is not applied in expected order.	Check the property "**itemChooseMethod**". It defines the order in which benefits apply to items. Possible values are: • Lowest: Take items with lowest price first (lowest benefit first). • Highest: Take items with highest price first (highest benefit first). • LowestPerInterval: Lowest benefit per interval first. • HighestPerInterval: Highest benefit per interval first. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**
If you get different results after a long debugging session.	Probably you exceeded the time calculation limit which defines the time limit in milliseconds (ms) for the best price algorithm. It is defined in property "**calculationTimeLimit**". The default value is 1000. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**

Symptom	To check
If a line item does not get an expected monetary discount.	Check whether the line item is able to get a monetary discount. The property "**itemTypesWithoutReceiptRelatedDiscounts**" contains a list of line item types which are not allowed for receipt-related monetary discounts. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**
If the expected decimal places are always fix.	Check the property "**rebateShareDecimalPlacesCount**" which defines the number of decimal places. Possible values are 0 to 4. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**
If you expected a zero rebate, but it does not appear in the results	Check whether zero rebates are enabled. This property is defined in "**allowZeroRebate**". The default value is false. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration().**
If the line items are calculated independently of the whole transaction.	Check the current calculation mode "**calculationMode**". This value is set to false in the configuration. But the value can appear into a request overriding the default config. This feature is available since API major version 2. The properties are mapped on: **com.gk_software.pricing_engine.core.configuration.ConfigurationInitializer.initializeConfiguration(...)** and available overall in PCE **Context** via the method **getConfiguration()** and controlled in the request in class: **com.gk_software.pricing_engine.psi.service.PromotionPricingServiceImpl.checkCalculationModeIntoRequestBodyAndOverride(...)**
If the expected multi-language feature does not appear in the response.	Check the requested API version. This feature is available since API major version 2. The feature is checked with help of method: **com.gk_software.pricing_engine.psi.util.ApiVersionUtils.checkAPIVersionWithNonnullValue(...).**

7.2.3 Calculate Transaction

Reference and entry point to debug	Description
DefaultPricingServiceImpl.calculateTransaction(...)	This is the entry point to the Pricing Engine. In this method, a PricingEngineRequest is created which is processed inside the Pricing Engine. The result is stored in the PricingEngineResponse.

7.2.4 PCE Processing

Start Business Logic.

Reference and entry point to debug	Description
PricingEngineImpl.process(...)	The main Pricing Engine process. Here the configuration and the calculation mode are checked and the execution of all pricing engine steps is started.
PricingEngineImpl.execute(...)	Here all steps of the calculation described below are executed. Finally, after the calculation the transaction will be updated via the TransactionUpdateHandler.
PricingEngineImpl.initializeContextItems(...)	Context initializations. Place to identify all relevant line items related to positions, transaction, and sale returns.
PricingEngineImpl.clearPromotionDataFromTransaction()	Clear all already used promotion data from the transaction.
PricingEngineImpl.executeTransactionAnalyzer()	Place to execute all registered transaction analyzers. For instance: CouponTransactionAnalizer, CustomCouponTransactionAnalyzer, CustomerGroupTransactionAnalyzer, ExternalPromotionsTransactionAnalyzer, ItemTransactionAnalyzer and PromotionManualTriggerTransactionAnalyzer
PricingEngineImpl.executeEligibilityLoader()	Place to execute all registered eligibility loaders. For instance: DefaultEligibilityLoader and ManualTriggerEligibilityLoader. Inside the DefaultEligibilityLoader, all EligibilityLoaderFragments are executed to load respective eligibility information via the Data Access Layer. Some important debugging points are: • **DefaultEligibilityLoader.executeFragment**: Execution of all fragments to load respective eligibility information. • **DefaultEligibilityLoader.fillItemBucket**: Here the item Bucket for the requested fragment is filled. • **DefaultEligibilityLoader.activateEligibility**: After loading the eligibilities regarding every fragment, here is the place to activate them. The eligibilities are loaded via **com.gk_software.pricing_engine.sapmapping.DataAccessServiceSAP.** Just check the desired fragment to find out which methods are used to access the Data Access Layer. After loading eligibilities via DataAccessServiceSAP, this information is mapped to the GK Model. Just check **com.gk_software.pricing_engine.sapmapping.DataMappingSAP2GKImpl.**

Reference and entry point to debug	Description
PricingEngineImpl.executeConditionLoader()	Place to load all conditions related to active eligibilities. See also **com.gk_software.pricing_engine.core.engine.ConditionLoaderImpl.execute(..).** The promotion conditions are loaded from Context. If no promotion conditions are found, they are loaded with help of DataAccessServiceSAP. Just check **com.gk_software.pricing_engine.sapmapping.DataAccessServiceSAP.getPromotionConditions().** Regarding every promotion condition, the respective condition rule is loaded: **com.gk_software.pricing_engine.core.engine.ConditionLoaderImpl.loadConditionRule(..)**
PricingEngineImpl.executeConditionCalculator()	Place to calculate all related conditions. See also **com.gk_software.pricing_engine.common.engine.PromotionCalculator.** Some important debugging points are: • **PromotionCalculatorImpl.calculateLineItemsConditions:** Performs calculation of line item conditions. • **PromotionCalculatorImpl.calculateTransactionConditions:** Performs calculation of transaction conditions. • **PromotionCalculatorImpl.checkExclusiveTransactionCondition:** Checks whether exclusive transaction conditions exist and whether they are used in the calculation. • **PromotionCalculatorImpl.applyCalculationResult**: Applies all valid PartialResults to the SummaryResult. In case of multiple valid PartialResults, the possible conflicts are handled with the best price calculation. To debug how conflicts are handled, just check **com.gk_software.pricing_engine.common.engine.ConflictHandler.**
TransactionUpdateHandlerImpl	Finally, the place to update the transaction after the calculation (SummaryResult -> TransactionResult). Just check **com.gk_software.pricing_engine.core.transaction.TransactionUpdateHandlerImpl.**

7.3 Performance Logging

7.3.1 Overview

The PCE core includes some basic performance monitoring features.

The monitoring is based on the JPerf Open Source library. This allows a fine-grained performance logging with some advanced analysis capabilities. The monitoring is controlled via logger settings.

There are different measurement points with additional new loggers that allow controlling the detail of the output.

7.3.2 Usage

To enable the performance logging, set the corresponding logger log level to INFO, DEBUG or TRACE level and log the information to a specific performance.log file.

There are different loggers available, see below for details.

The performance.log contains log entries in the following format (depending on your log-format configuration).

```
INFO [main]: start[1494336965104] time[171] tag[pricing.service.calculate.success]
message[MessageId: cha_basic_not_discountable_flag1_request]
```

Explanation:

INFO: Log level at which the message is logged.

[main]: Thread name where the log is created.

start[1494336965104] : Timestamp (System.currentTimeMillis) at which the log was produced.

time[171] : This is the time in ms that was measured for the operation.

tag[pricing.service.calculate.success] : The tag is a formalized definition of the operation that is measured. For details on the available tags, see below.

message[MessageId: cha_basic_not_discountable_flag1_request] : Some log entries have an optional message that gives some more details on the measurement (that is, messageId of the request).

For example:

```
DEBUG [http-nio-9081-exec-1]: start[1494517222819] time[12]
tag[pricing.engine.internal.execute.prepareContext]
DEBUG [http-nio-9081-exec-1]: start[1494517222832] time[1]
tag[pricing.engine.internal.execute.clearPromotionDataFromTransaction]
DEBUG [http-nio-9081-exec-1]: start[1494517222834] time[18]
tag[pricing.engine.internal.execute.executeTransactionAnalyzer]
TRACE [http-nio-9081-exec-1]: start[1494517222853] time[184]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
TRACE [http-nio-9081-exec-1]: start[1494517223039] time[40]
tag[pricing.data_access.DataAccessServiceDelegate.getMarketBasketAmountEligibilities]
```

```
DEBUG [http-nio-9081-exec-1]: start[1494517222852] time[227]
tag[pricing.engine.internal.execute.executeEligibilityLoader]
INFO  [http-nio-9081-exec-1]: start[1494517222819] time[262] tag[pricing.engine.process]
INFO  [http-nio-9081-exec-1]: start[1494517222723] time[362]
tag[pricing.service.calculate.success] message[MessageId:
i_can_calculate_large_baskets_1_request]
DEBUG [http-nio-9081-exec-2]: start[1494517223292] time[5]
tag[pricing.engine.internal.execute.executeTransactionAnalyzer]
TRACE [http-nio-9081-exec-2]: start[1494517223299] time[5]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
TRACE [http-nio-9081-exec-2]: start[1494517223304] time[4]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
TRACE [http-nio-9081-exec-2]: start[1494517223309] time[3]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
TRACE [http-nio-9081-exec-2]: start[1494517223312] time[3]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
TRACE [http-nio-9081-exec-2]: start[1494517223316] time[2]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
TRACE [http-nio-9081-exec-2]: start[1494517223319] time[6]
tag[pricing.data_access.DataAccessServiceDelegate.getMarketBasketAmountEligibilities]
DEBUG [http-nio-9081-exec-2]: start[1494517223299] time[26]
tag[pricing.engine.internal.execute.executeEligibilityLoader]
INFO  [http-nio-9081-exec-2]: start[1494517223292] time[34] tag[pricing.engine.process]
INFO  [http-nio-9081-exec-2]: start[1494517223283] time[46]
tag[pricing.service.calculate.success] message[MessageId:
i_can_calculate_large_baskets_5_request]
DEBUG [http-nio-9081-exec-3]: start[1494517223390] time[45]
tag[pricing.engine.internal.execute.executeTransactionAnalyzer]
TRACE [http-nio-9081-exec-3]: start[1494517223436] time[5]
tag[pricing.data_access.DataAccessServiceDelegate.getItemEligibilities]
```

For performance logging, you should create a new log appender that logs all loggers starting with performance.pricing-engine,

that is, for LogBack:

<logger name="performance.pricing-engine" level="info" additivity="false">

<appender-ref ref="PERF" />

</logger>

Recommended log formatter pattern for Log4j:

<param name="conversionPattern" value="%-5p [%t]: %m%n"/>

Recommended log formatter pattern for LogBack:

<layout>

```
<pattern>%-5level [%thread]: %message%n</pattern>
</layout>
```

7.3.2.1 Measurement Points

All measurements are using a logger depending on the module and class name.

The logger name pattern used is: performance.<ModuleName>.<ClassName>,

for example: performance.engine.com.gk_software.pricing_engine.engine.PricingEngineImpl

Depending on the log level set for the different loggers, there may be additional more detailed performance log entries measured. The overhead of the performance logging is very low (<1% overhead), but it is recommended to use the DEBUG and TRACE level only if you need some more detailed information, as it will add a lot more data to the performance log.

All operations that are measured with less than 1ms are not written to the performance log.

Module	Class name	Log Level	Tag	Description
psi-sap	com.gk_software.pricing_engine.psi.service.PromotionPricingServiceImpl	INFO	pricing.service.calculate.success pricing.service.calculate.error	Overall processing time of a PCE service call. Error and success tags used depending on result (BusinessError or Success).

Module	Class name	Log Level	Tag	Description
engine	com.gk_software.pricing_engine.engine.PricingEngineImpl	INFO	pricing.engine.process	Overall processing time of the promotion calculation (excluding request and response parsing)
		DEBUG	pricing.engine.internal.execute.prepareContext	Time to initialize engine
		DEBUG	pricing.engine.internal.execute.clearPromotionDataFromTransaction	Time to clear previous promotion data from transaction
		DEBUG	pricing.engine.internal.execute.executeTransactionAnalyzer	Time to parse and analyze the transaction data
		DEBUG	pricing.engine.internal.execute.executeAdhocPromotionHandler	Time to load adhoc promotion data

Module	Class name	Log Level	Tag	Description
		DEBUG	pricing.engine.internal.execute.executeEligibilityLoader	Overall time to load all relevant eligibilities, check, and activate them
		DEBUG	pricing.engine.internal.execute.executeConditionLoader	Time to load all conditions and rules for active eligibilities
		DEBUG	pricing.engine.internal.execute.executeConditionCalculator	Time to do the promotion calculation for all viable promotions
		DEBUG	pricing.engine.internal.execute.transactionUpdateHandler>	Time to update transaction data with calculation results
engine	com.gk_software.pricing_engine.engine.PromotionCalculatorImpl	DEBUG	pricing.calculation.calculateLineItemsConditions	Time to calculate all Position (LineItem) related promotions

Module	Class name	Log Level	Tag	Description
		DEBUG	pricing.calculation.calculateTransactionConditions	Time to calculate all Transaction (Basket) related promotions
data_access	com.gk_software.pricing_engine.api.data_access.DataAccessServiceDelegate	TRACE	pricing.data_access.DataAccessServiceDelegate.getEligibility	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getChildEligibilitiesCount	Detailed logging of single data access method for loading promotion data from DB

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.getItemEligibilities	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getCustomerGroupEligibilities	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getCouponEligibilities	Detailed logging of single data access method for loading data from DB

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.getMerchandiseGroupEligibilities	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getMerchandiseSetEligibilities	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getCombinationEligibility	Detailed logging of single data access method for loading promotion data from DB

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.getMarketBasketAmountEligibilities	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getParentEligibilitiesForMerchandiseGroup	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getChildEligibilitiesForParentEligibilitiesForMerchandiseGroup	Detailed logging of single data access method for loading promotion data from DB

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.getParentCombinationEligibilitiesForMerchandiseGroup	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getPromotion	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getPromotionConditions	Detailed logging of single data access method for loading promotion data from DB

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.getItemRule	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getGet3Pay2Rule	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.findAssociatedMhgByChildId	Detailed logging of single data access method for loading promotion data from DB.

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.getMixAndMatchRule	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getExternalActionRule	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getMixAndMatchItems	Detailed logging of single data access method for loading promotion data from DB

Module	Class name	Log Level	Tag	Description
		TRACE	pricing.data_access.DataAccessServiceDelegate.findRoundingRuleByKey	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getManualPromotionEligibilities	Detailed logging of single data access method for loading promotion data from DB
		TRACE	pricing.data_access.DataAccessServiceDelegate.getRule	Detailed logging of single data access method for loading promotion data from DB

7.3.2.2 Analysis of Performance logs

The JPerf library has some built-in log parser to ease the evaluation of the performance logs.

Use it on the command line like follows (having the jperf-1.0.3.jar downloaded or got from *webapp/lib* folder):

```
java -jar jperf-1.0.3.jar log/performance.log -o log/performance-summary.log <options>
```

Calling this script creates a new performance-summary.log file that summarizes the execution times of the measured data and generates statistical information.

An example is shown in the following section:

```
Performance Statistics   2017-05-09 15:36:00 - 2017-05-09 15:36:30
Tag                                                                    Avg(ms)
Min         Max     Std-Dev       Count       Total
pricing.calculation.calculateLineItemsConditions                         204,0
9         582       229,0           8        1632
pricing.calculation.calculateTransactionConditions                        14,0
14          14         0,0           1          14
pricing.data_access.DataAccessServiceDelegate.getChildEligibilitiesCount  2,5
1           6         2,1           4          10
pricing.data_access.DataAccessServiceDelegate.getCombinationEligibility   3,8
3           6         1,3           4          15
pricing.data_access.DataAccessServiceDelegate.getCouponEligibilities      2,3
2           3         0,5           7          16
pricing.data_access.DataAccessServiceDelegate.getCustomerGroupEligibilities  1,7
1           3         0,7          18          31
pricing.data_access.DataAccessServiceDelegate.getExternalActionRule       5,0
3           7         2,0           2          10
pricing.data_access.DataAccessServiceDelegate.getItemEligibilities        3,3
1          52         4,2         155         519
pricing.data_access.DataAccessServiceDelegate.getItemRule                 1,8
1           6         1,3          12          22
pricing.data_access.DataAccessServiceDelegate.getManualPromotionEligibilities  2,6
1           7         1,3          14          36
pricing.data_access.DataAccessServiceDelegate.getMarketBasketAmountEligibilities  3,9
2          24         3,9          32         126
pricing.data_access.DataAccessServiceDelegate.getMerchandiseGroupEligibilities  2,0
1           4         0,9          24          49
pricing.data_access.DataAccessServiceDelegate.getMixAndMatchItems         5,0
5           5         0,0           1           5
pricing.data_access.DataAccessServiceDelegate.getMixAndMatchRule          7,0
7           7         0,0           1           7
pricing.data_access.DataAccessServiceDelegate.getPromotion                3,8
1          22         4,9          15          57
pricing.data_access.DataAccessServiceDelegate.getPromotionConditions      10,1
3          50        10,0          18         181
pricing.engine.internal.execute.executeConditionCalculator               183,6
9         582       223,9           9        1652
pricing.engine.internal.execute.executeConditionLoader                    26,4
4          88        23,6          12         317
pricing.engine.internal.execute.executeEligibilityLoader                  34,7
```

```
5         341        59,7          32        1111
pricing.engine.internal.execute.executeTransactionAnalyzer                    8,2
3          66        11,2          32         261
pricing.engine.internal.execute.prepareContext                                6,0
6           6         0,0           1           6
pricing.engine.internal.execute.transactionUpdateHandler                    114,3
4         460       180,4           9        1029
pricing.engine.process                                                      148,0
10        1089       270,4          30        4441
pricing.service.calculate.error                                               9,8
5          18         4,2           6          59
pricing.service.calculate.success                                           182,5
16        1098       290,7          30        5474
Performance Statistics   2017-05-09 15:36:30 - 2017-05-09 15:37:00
Tag                                                                        Avg(ms)
Min         Max     Std-Dev       Count       Total
pricing.calculation.calculateLineItemsConditions                            124,7
16         557       193,9           6         748
pricing.calculation.calculateTransactionConditions                         5151,8
52       25534     10191,1           5       25759
pricing.data_access.DataAccessServiceDelegate.getChildEligibilitiesCount      2,0
1           3         0,7           4           8
pricing.data_access.DataAccessServiceDelegate.getCombinationEligibility       3,5
3           4         0,5           4          14
pricing.data_access.DataAccessServiceDelegate.getExternalActionRule           2,7
2           3         0,5           3           8
pricing.data_access.DataAccessServiceDelegate.getItemEligibilities            2,9
2           9         1,7          16          47
pricing.data_access.DataAccessServiceDelegate.getItemRule                     1,4
1           3         0,6          14          20
pricing.data_access.DataAccessServiceDelegate.getMarketBasketAmountEligibilities    4,2
2          17         4,4          11          46
pricing.data_access.DataAccessServiceDelegate.getMerchandiseGroupEligibilities    2,0
2           2         0,0           2           4
pricing.data_access.DataAccessServiceDelegate.getPromotion                    1,7
1           3         0,7          17          29
pricing.data_access.DataAccessServiceDelegate.getPromotionConditions          6,6
5          10         1,5          17         112
pricing.engine.internal.execute.executeConditionCalculator                 2410,5
16       25535      7314,1          11       26515
pricing.engine.internal.execute.executeConditionLoader                        19,0
10          53        11,5          11         209
pricing.engine.internal.execute.executeEligibilityLoader                      17,0
8           37         9,5          11         187
pricing.engine.internal.execute.executeTransactionAnalyzer                 1530,9
3        16795      4826,9          11       16840
pricing.engine.internal.execute.transactionUpdateHandler                    130,1
7         511       147,6          10        1301
pricing.engine.process                                                      269,2
61        1105       291,3          10        2692
pricing.service.calculate.success                                           279,1
69        1113       290,9          10        2791
```

For each operation, you get information on average execution time, minimum and maximum execution times, standard deviation, invocation count of the method, and total execution time of all calls.

The parser generates statistics for each 30 seconds time slice by default. This is configurable by passing the parameter -t (time in ms). By passing the -r parameter, you get additional statistical rollups that summarize tags with the same prefix.

There are the following additional options that can be used (use --help to print them out).

```
Arguments:
  logInputFile - The log file to be parsed. This is log/performance in the analyze-
performance script.
  -o|--out|--output outputFile - The file where generated statistics should be written. If
not specified, statistics are written to stdout. This is fixed to log/performance-
summary.log in the analyze-performance script
  -g|--graph graphingOutputFile - The file where generated perf graphs should be written. If
not specified, no graphs are written.
  -t|--timeslice timeslice - The length of time (in ms) of each timeslice for which
statistics should be generated. Defaults to 30000 ms.
  -r - Whether or not statistics rollups should be generated. If not specified, rollups are
not generated.
  -f|--format text|csv - The format for the statistics output, either plain text or CSV.
Defaults to text.
                         If format is csv, then the columns output are tag, start, stop,
mean, min, max, stddev, and count.
Note that out, stdout, err and stderr can be used as aliases to the standard output streams
when specifying output files.
```

You can also output the data in a graphical format or as CSV to import into Excel.

See the JPerf documentation for more details.

7.3.3 Development

As a developer of a PCE extension, you may want to add additional measurement points in your own code.

The basic class for measurement of execution times form JPerf library is LoggingStopWatch. Create an instance of this class and use start, lap, and stop methods for trigger performance measurement. See also JPerf developer guide for more details.

The creation of a LoggingStopWatch object is very cheap, so it adds no performance overhead.

There is a helper class com.gk_software.pricing_engine.common.utils.PerformanceUtils in commons project to support you in doing so. This class contains several static methods to help you in using performance measurement.

The following code can be used to create and start a new stopWatch at the beginning of a method:

```
LoggingStopWatch logWatch = PerformanceUtils.start(PERF_LOGGER, PerformanceLevel.DEBUG,
        "pricing.engine.internal.execute", null);
```

You should create a new watch on each method call to be thread-safe. While this adds some Garbage collector overhead, this is negligible compared to the business method execution.

The newStopWatch method automatically adds a threshold of 1 ms to the LoggingStopWatch, which means methods executing in less than 1ms are not logged. The PerformanceLevel defines on which log level the measurement is logged.

You could also get a performance logger in your class and create stop watches from there:

```
private static final Logger PERF_LOGGER = PerformanceUtils.getPerfLog("engine",
PricingEngineImpl.class);

....

LoggingStopWatch watch = PerformanceUtils.newStopWatch(PERF_LOGGER, PerformanceLevel.INFO);
```

It is recommended to use a separate logger for performance logging to allow easy filtering in the log config.

When you create a new instance of a LoggingStopWatch, it is automatically initialized with current time, so there is no need to start it. If you want to reset or restart it inside your code, use the start() or lap() methods.

To stop and log a watch, use the stop() method.

```
if(logWatch.isLogging()) {
  PerformanceUtils.lap(logWatch, "pricing.engine.internal.execute.prepareContext", null);
}

PerformanceUtils.stop(logWatch, "pricing.engine.internal.execute.transactionUpdateHandler",
null);
```

Use watch.isLogging() when you generate some more complex tags to prevent unnecessary overhead when performance monitoring is not enabled. You can pass the tag in start stop or lap method as you need, recommended is to pass it on stop. You may optionally use stop(String,String) to pass an additional message.

The tag should have several levels separated by a point (.), a similar syntax like a Log4j logger name. It is used for grouping entries during the analysis. Best practice is to prefix the tag with some fixed entry based on the module, class, and/or method measured and then some details if needed for granular measurement.

8 Configuration

For details on the configuration refer to the document Functional Guide for the Promotion Calculation Engine.

9 Glossary

The glossary is intended to support understanding of the main terms used in this documentation.

Term	Definition
Ad hoc promotion	Promotions loaded from external systems if available and not from the Data Access Layer.
ARTS	The Association for of Retail Technology Standards (ARTS) is an organization providing application standards and guidance to the retail industry.
Business Error	Feedback regarding problems caused by the data sent in with the *request*. Currently, only errors caused by wrong or missing data in the request are supported.
Calculation Rule	Describes the benefit to be granted to the customer in case that the condition is applied.
Condition	A condition can be understood as a single action granting a benefit to the customer. It consists of a set of triggers (*eligibilities*), time validity, and a rule describing the benefit. Multiple conditions can belong to a *promotion*.
Context	Holds the complete state for transaction processing and promotion calculation.
Domain (Object) Model	Conceptual model of a problem or business domain.
Eligibility	Precondition which has to be fulfilled in order to apply the condition. An eligibility triggers a *condition*.
Extension Map	Possibility to define references to external data structures for the objects of a *transaction*.
Line Item	An element of a transaction that contains information on goods or services delivered.
Normalized Item	Representation of a product, an article, a good, or a service that is bought or returned by a customer. An item is the smallest unit or customer pack that can be ordered independently and that cannot be split further into any smaller units.
Pricing Engine	The core of the Promotion Calculation Engine. The Pricing Engine passes the request from the application, performs the price determination and returns the result.
Pricing Service Interface (PSI)	Standard interface schema provided by *ARTS* for integrating a pricing service with a price requesting or selling system.
Promotion	Functional mapping of a campaign granting a benefit to the customer. A promotion consists of one or more *conditions*.
Request	Message that contains common information pieces which are needed in order to process the message: A shopping basket with multiple line items and other information being relevant for the calculation of the benefits for the shopping basket. The request is provided via the client API.

Term	Definition
Response	Responding message regarding the *request* sent in. This can be the updated shopping basket including calculated discounts and bonus points as well as information about used promotion master data and applied coupons, or an error information in case that the request could not be processed.
Technical Error	Error due to technical problems which prevents the calculation to be fulfilled (for example, a database exception).
Transaction	The shopping basket with line item data and other information relevant for price calculation.

10 Appendix

10.1 Appendix A: Request and Response Mapping

10.1.1 Mapping of PCE internal model to/from the interface

In order to enable the processing of data on the PCE, SAP data objects found under package com.sap.ppengine.client.dto are mapped to internal models defined at package com.gk_software.pricing_engine.model.transaction. This chapter describes the related mapping from Client API to PCE Domain Object (DO) model.

The Promotion Calculation Engine receives a request via the client API. The request contains a shopping basket with multiple line items. The service applies monetary discounts and loyalty points to the given shopping basket based on its content (items, prices, scanned coupons, assigned customer groups, etc.) on the one hand and the master data accessible via the data access API on the other hand. Finally, it responds the shopping basket with added information about the applied promotions. The transmitted content is based on the ARTS Pricing Service Interface schema version 4.0.0. Where needed, extensions were added to it.

10.1.1.1 Client API DTOs

10.1.1.1.1 PriceCalculate

The request sent to the calculation engine.

Element/Attribute	DO Entity	DO Field	Rule/Note
ARTSHeader	-	-	**GKR-100506** is sent back if the header is not included.
PriceCalculateBody	-	-	
InternalMajorVersion	-	-	Must not be less than the running PCE's major version, otherwise **GKR-100500** is sent back. Must not be null, otherwise **GKR-100508** is sent back.

Element/Attribute	DO Entity	DO Field	Rule/Note
InternalMinorVersion	-	-	If null, then the mapped field will be set to 0. Must be 0, otherwise it is a business error. If the major version provided by the Client API matches the running PCE version, then must not be lower than the running PCE's minor version, otherwise **GKR-100500** is sent back.
Any	-	-	

10.1.1.1.2 PriceCalculateResponse

The response of the calculation engine. The content of the request is taken over and extended where nothing else is stated below.

Element/Attribute	DO Entity	DO Field	Rule/Note
ARTSHeader	-	-	
PriceCalculateBody	-	-	
InternalMajorVersion	-	-	The internal major version sent in the request message is mapped back.
InternalMinorVersion	-	-	The internal minor version sent in the request message is mapped back.
Any	-	-	

10.1.1.1.3 ARTSCommonHeaderType

Element/Attribute	DO Entity	DO Field	Rule/Note
MessageID	-	-	In the request the client sends a UUID. If null or empty **GKR-100506** is sent back. To the response a new UUID generated by the PCE is set.
DateTime	-	-	The date and time when this message header was created. For example, 2015-01-13T04:48:30.427-05:00.
Response	-	-	
Response.any	-	-	
Requestor	-	-	Is routed through simply.
BusinessUnit	Transaction	BusinessUnitID	

Element/Attribute	DO Entity	DO Field	Rule/Note
BusinessUnit.TypeCode	-	-	It is ensured by the client that minimally one BusinessUnit element of TypeCode "RetailStore" resp. with no type information is sent in the request. The first BusinessUnit element meeting that condition is considered. If no BusinessUnit sent in the request does not exist, business error**GKR-100506** is sent back.
WorkstationID	Transaction	WorkstationID	Is routed through simply.
RequestedLanguage	-	-	Language code in ISO 639-1 format for texts requested from client (deprecated: use RequestedMultiLanguage instead). If missing in the request, the first found language in the database is returned.
RequestedMultiLanguage			List of Language codes in ISO 639-1 format for texts requested from client.
Any	-	-	
ActionCode	-	-	Only possible values are "Calculate" and "Lookup", this is checked in OData. "Calculate" (default): Pricing is requested for all line items in the shopping basket together. "Lookup": Pricing is requested per single line item without taking into consideration any other content of the shopping basket. This is not part of version 1. If the attribute is missing in the request, it is to be set to the default.
MessageType	-	-	Only possible values are "Request" and "Response", this is checked in OData. "Request": Within PriceCalculate. "Response": Within PriceCalculateResponse. If the attribute is missing in the request, it is to be set to "Request". If "Response" is used in PriceCalculate, business error**GKR-100500** is sent back.

10.1.1.1.4 ResponseCommonData

Element/Attribute	DO Entity	DO Field	Rule/Note
RequestID	-	-	The MessageID of the request.
ResponseTimestamp	-	-	The DateTime of the request header (ARTSCommonHeaderType).
BusinessError	-	-	Only filled in case a BusinessError has occurred.
BusinessError.any	-	-	
ResponseCode	-	-	"Rejected" is used when a business error (price calculation impossible) occurred. Requests for which a price calculation could be processed are responded with "OK".

10.1.1.1.5 PriceCalculateBase

Element/Attribute	DO Entity	DO Field	Rule/Note
TransactionID	Transaction	TransactionID	No pricing-relevant information. Is routed through simply. **If it exceeds 128 chars, it is truncated, the request message can be processed further. If it is not provided in the request, an internal identifier is generated by the PCE.**
DateTime	RetailTransaction	PromotionTimestamp	Time zone considered.
Loyalty	-	-	See mapping of LoyaltyAccountCommonData.
ShoppingBasket	-	-	See mapping of ShoppingBasketBase. Only one shopping basket per request is supported.
RegularSalesUnitPriceRoundingRule	-	-	Rounding rule for regular prices (element RegularSalesUnitPrice and ExtendedAmount calculation from request). For more information see mapping of RoundingRuleType.
Any	-	-	
TransactionType	-	-	This information is not checked. The line item type is an important information for correct pricing. Currently, only sales transactions are supported. Of course, they can contain return line items. Assuming that a return transaction consists of return line items, it could be priced correctly either.

Element/Attribute	DO Entity	DO Field	Rule/Note
NetPriceFlag	-	-	This information is not checked.
CalculationMode	com.gk_software.pricing_engine.api.context.Configuration	calculationMode	Calculation mode for the price calculation. The default is Basket. Valid values are Basket and LineItem. If not included in the request, **GKR-100500** will be thrown. For version one only Basket mode is allowed, if other value is sent, then **GKR-100500** will be thrown.

10.1.1.1.6 LoyaltyAccountCommonData

Element/Attribute	DO Entity	DO Field	Rule/Note
CustomerID	RetailTransactionCustomer	CustomerID	If the CustomerID element is set and the length of it in the request exceeds 40 chars, business error **GKR-100501** is sent back. If the request contains minimally one LoyaltyProgramID element, but no CustomerID element, business error **GKR-100506** is sent back.
LoyaltyProgram	-	-	See mapping of LoyaltyAccountType.
CustomerIsEmployeeFlag	-	-	Currently, we do not need this information as the only way for granting employee discounts in this context is to handle employees as customers (no employee discount groups available). Mapped back to the response.

10.1.1.1.7 LoyaltyAccountType

Element/Attribute	DO Entity	DO Field	Rule/Note
LoyaltyProgramID	RetailTransactionCustomerGroupAssignment	CustomerGroupID	If the LoyaltyProgramID element length exceeds 40 chars, business error **GKR-100501** is sent back. If the LoyaltyProgramID element does not exist, business error **GKR-100506** is sent back.
any	-	-	

10.1.1.1.8 ShoppingBasketBase

Element/Attribute	DO Entity	DO Field	Rule/Note
LineItem	-	-	See mapping of LineItemDomainSpecific.
any	-	-	

10.1.1.1.9 LineItemDomainSpecific

-> extends LineItemBase-> extends LineItemChoiceDomainSpecific-> extends LineItemChoiceBase

Element/Attribute	DO Entity	DO Field	Rule/Note
LineItemChoiceBase.Sale	-	-	See mapping of SaleBase.
LineItemChoiceBase.SaleForDelivery	-	-	See mapping of SaleForDeliveryBase.
LineItemChoiceBase.SaleForPickup	-	-	See mapping of SaleForPickupBase.
LineItemChoiceBase.Return	-	-	See mapping of ReturnBase.
LineItemChoiceBase.ReturnForDelivery	-	-	See mapping of ReturnForDeliveryBase.
LineItemChoiceBase.ReturnForPickup	-	-	See mapping of ReturnForPickupBase.
LineItemChoiceBase.CustomerOrderForDelivery	-	-	See mapping of CustomerOrderForDeliveryBase.
LineItemChoiceBase.CustomerOrderForPickup	-	-	See mapping of CustomerOrderForPickupBase.
LineItemChoiceBase.Discount	-	-	See mapping of DiscountBase.
LineItemChoiceBase.LoyaltyReward	-	-	See mapping of LoyaltyRewardBase.
LineItemChoiceBase.Coupon	-	-	See mapping of TenderCouponBase.

Element/Attribute	DO Entity	DO Field	Rule/Note
LineItemChoiceDomainSpecific.PromotionManualTrigger	-	-	See mapping of PromotionManualTriggerType.
LineItemChoiceDomainSpecific.PromotionExternalTrigger	-	-	See mapping of PromotionExternalTriggerType.
LineItemChoiceDomainSpecific.any	-	-	

Element/Attribute	DO Entity	DO Field	Rule/Note
LineItemBase.SequenceNumber	RetailTransactionLineItem	RetailTransactionLineItemSequenceNumber	For sub-types SaleReturnLineItem, PriceModificationLineItem, LoyaltyRewardLineItem only. This is as at DO side, coupons, manual triggers, and external triggers are not handled as line sub-types of RetailTransactionLineItem. In the request, if LineItem sub-type is one of the following: • Sale • SaleForDelivery • SaleForPickup • Return • ReturnForDelivery • ReturnForPickup • CustomerOrderForDelivery • CustomerOrderForPickup • Discount • LoyaltyReward and SequenceNumber does not fall into the smallint/short range, then **GKR-100501** if sent back. If the LineItem sub-type is one of the listed ones and the sequence number is not unique within the line items of those sub-types in the message, business error **GKR-100505** is sent back. If for the above mentioned LineItem sub-types the sequence number is not sent or empty, **GKR-100506** is sent back.

Element/Attribute	DO Entity	DO Field	Rule/Note
LineItemBase.MerchandiseHierarchy	SaleReturnLineItemMerchandiseHierarchyGroup	MerchandiseHierarchyGroupID	Calculation engine needs information about all merchandise hierarchy groups which are assigned to the line item directly or indirectly. Makes sense for sub-type SaleReturnLineItem only, otherwise it is to be ignored. If the length of the element in the request exceeds 30 chars, business error **GKR-100501** is sent back.
LineItemBase.MerchandiseHierarchy.ID	SaleReturnLineItemMerchandiseHierarchyGroup	MerchandiseHierarchyGroupIDQualifier	See explanation of LineItemBase.MerchandiseHierarchy. If the length of the element in the request exceeds 4 chars, business error **GKR-100501** is sent back.

10.1.1.1.10 SaleBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "SI" (sale) and
- SaleReturnLineItem.ItemType = "CO" (common)

10.1.1.1.11 SaleForDeliveryBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "SI" (sale) and
- SaleReturnLineItem.ItemType = "CO" (common)

plus to a SaleReturnLineItemSalesOrder related to the SaleReturnLineItem with

- SaleReturnLineItemSalesOrder.SalesOrderDeliveryTypeCode = "01" (delivery)

10.1.1.1.12 SaleForPickupBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "SI" (sale) and
- SaleReturnLineItem.ItemType = "CO" (common)

plus to a SaleReturnLineItemSalesOrder related to the SaleReturnLineItem with

- SaleReturnLineItemSalesOrder.SalesOrderDeliveryTypeCode = "03" (pickup other store)

10.1.1.1.13 ReturnBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "RI" (return) and
- SaleReturnLineItem.ItemType = "CO" (common)

10.1.1.1.14 ReturnForDeliveryBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "RI" (return) and
- SaleReturnLineItem.ItemType = "CO" (common)

plus to a SaleReturnLineItemSalesOrder related to the SaleReturnLineItem with

- SaleReturnLineItemSalesOrder.SalesOrderDeliveryTypeCode = "01" (delivery)

10.1.1.1.15 ReturnForPickupBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "RI" (return) and
- SaleReturnLineItem.ItemType = "CO" (common)

plus to a SaleReturnLineItemSalesOrder related to the SaleReturnLineItem with

- SaleReturnLineItemSalesOrder.SalesOrderDeliveryTypeCode = "03" (pickup other store)

10.1.1.1.16 CustomerOrderForDeliveryBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "SI" (sale) and
- SaleReturnLineItem.ItemType = "SO" (sales order)

10.1.1.1.17 CustomerOrderForPickupBase

Extends ItemDomainSpecific. Mapping to SaleReturnLineItem at DO side is the same with the following extensions:

- SaleReturnLineItem.ActionCode = "SI" (sale) and
- SaleReturnLineItem.ItemType = "PU" (sales order pickup)

10.1.1.1.18 ItemDomainSpecific

extends ItemBase

Element/Attribute	DO Entity	DO Field	Rule/Note
ItemBase.ItemID	SaleReturnLineItem	ItemID	In the schema, there is a choice between ItemID and MerchandiseHierarchyGroup element. In case that no ItemID is provided, it is a merchandise category
ItemBase.MerchandiseHierarchy	SaleReturnLineItem	MainMerchandiseHierarchyGroupID	

Element/Attribute	DO Entity	DO Field	Rule/Note
	SaleReturnLineItem	POSItemID	sale corresponding to the following SaleReturnLineItem structure: • SaleReturnLineItem.ItemID = ItemBase.MerchandiseHierarchy • SaleReturnLineItem.SubItemType = "MSTR" (Merchandise Category Sale) If the length of the MerchandiseHierarchy element exceeds 30 chars, business error **GKR-100501** is sent back. Regarding the length of the itemID: please consider the restrictions of the environment the PCE is used. Validation of the length of this field is not performed by the PCE.
ItemBase.MerchandiseHierarchy.ID	SaleReturnLineItem	MainMerchandiseHierarchyGroupIDQualifier	If the length of the qualifier exceeds 4 chars, business error **GKR-100501** is sent back.
ItemBase.RegularSalesUnitPrice	SaleReturnLineItem	ActualUnitPrice	It is ensured by the client that this element is provided for FixedPriceFlag = true and for the use case with the client being responsible for price lookup.
ItemBase.RegularSalesUnitPrice.Currency	-	-	
-	SaleReturnLineItem	ExtendedAmount	SaleReturnLineItem.Quantity * SaleReturnLineItem.Units * SaleReturnLineItem.ActualUnitPrice, rounded according to RoundingRuleType
ItemBase.ExtendedAmount	-	-	As there is no corresponding element in the DO, this information needs to be calculated for response as: SaleReturnLineItem.ExtendedAmount + SaleReturnLineItem.ExtendedDiscountAmount + ∑(RetailPriceModifier.Amount with ProrateFrom <> null for the considered line item)
ItemBase.ExtendedAmount.Currency	-	-	

Element/Attribute	DO Entity	DO Field	Rule/Note
ItemBase.ExtendedDiscountAmount	SaleReturnLineItem	ExtendedDiscountAmount	Mapped from the request only. As there is no corresponding element in the DO (SaleReturnLineItem.ExtendedDiscountAmount does not fit as it does not differentiate between discounts and increasements), this information needs to be calculated for response as: absolute value of ∑(RetailPriceModifier.Amount with ProrateFrom is null and Amount < 0 for the considered line item)
ItemBase.ExtendedDiscountAmount.Currency	-	-	
ItemBase.Quantity	SaleReturnLineItem	Quantity	The DO element does not support decimal places. In case decimal places are sent in request, a business error**GKR-100502** is sent back. The DO element does not support multiple quantities. In case that the Quantity element occurs more than once, a business error **GKR-100500** is sent back. If the sent quantity value falls out of the Integer range, **GKR-100501** will be sent back. **In case that no Quantity element occurs in request, 1 is used.**
ItemBase.Quantity.Units	SaleReturnLineItem	Units	In case that this attribute is missing, 1 is used. It must fall into the (11,3) range. If the number of decimal places exceeds 8 business error **GKR-100502** is sent back. If the length of the number otherwise exceeds 11, **GKR-100501** is sent back.
ItemBase.Quantity.UnitOfMeasureCode	SaleReturnLineItem	UnitOfMeasureCode	If the element is missing in the request, business error **GKR-100506** is sent back. If the length of the element in the request exceeds 4 chars, business error **GKR-100501** is sent back.
ItemBase.RetailPriceModifier			See mapping of RetailPriceModifierDomainSpecific.
ItemBase.ItemLink	RetailTransactionLineItemAssociation	ToLineItemSequenceNumber	RetailTransactionLineItem.RetailTransactionLineItemSequenceNumber
	RetailTransactionLineItemAssociation	LineItemSequenceNumber	If it does not fall into the smallint/short range or is not unique in the context of the given line item, business error **GKR-100501** is sent back. Otherwise, the given value is taken. It is not checked whether that sequence number exists within the transaction.

Element/Attribute	DO Entity	DO Field	Rule/Note
	RetailTransactionLineItemAssociation	LineItemAssociationTypeCode	"LINK"
ItemBase.ItemType	SaleReturnLineItem	ItemType	Mapping: Stock: "CO" or "SO" or "PU" (see above) GiftCertificate: "GC" any other: business error **GKR-100500** **If this element is not provided in request, it is considered as "Stock".** Note: The calculation engine cannot handle unknown values. If you want to use other values than those ones mapped above, you have to overwrite the enum adapter.
ItemBase.NonDiscountableFlag	SaleReturnLineItem	DiscountFlag	Mapping: invert. **If this element is not provided in request, default "false" is used.**
ItemBase.FixedPriceFlag	-	-	Affects price lookup only: Price must not be read again. However, promotions can be applied on top. If this element is not provided in the request, default "false" is used.
ItemDomainSpecific.TaxIncludedInPriceFlag	-	-	Mapped back to response as is.
ItemDomainSpecific.any	-	-	
ItemDomainSpecific.NonPieceGoodFlag	SaleReturnLineItem	QuantityInputMethod	Mapping: false: "01" (piece) true: "06" (quantity with decimal places, automatic input request) **If this element is not provided in the request, default "false" is used.**
ItemDomainSpecific.FrequentShopperPointsEligibilityFlag	SaleReturnLineItem	FrequentShopperPointsEligibilityFlag	If this element is not provided in the request, default "false" is used.

Element/Attribute	DO Entity	DO Field	Rule/Note
ItemDomainSpecific.DiscountTypeCode	SaleReturnLineItem	DiscountTypeCode	If the length of the element in the request exceeds 1 char, business error **GKR-100501** is sent back. If the element in the request does not contain a numeric char, business error **GKR-100500** is sent back.
ItemDomainSpecific.PriceTypeCode	SaleReturnLineItem	PriceTypeCode	If the request element does not contain a value of the enumeration • 00 • 01 business error **GKR-100500** is sent back. If this element is not provided in the request or is empty, default "00" is used.
ItemDomainSpecific.NotConsideredByPriceEngineFlag	SaleReturnLineItem	NotConsideredByLoyaltyEngineFlag	If this element is not provided in the request, default "false" is used.
ItemDomainSpecific.FrequentShopperPointsModifier	-	-	See mapping of FrequentShopperPointsModifierType.
ItemDomainSpecific.PromotionPriceDerivationRuleReference	-	-	See mapping of PromotionPriceDerivationRuleReferenceType.
ItemDomainSpecific.PromotionManualTrigger	-	-	See mapping of PromotionManualTriggerType.

10.1.1.1.19 RetailPriceModifierDomainSpecific

extends RetailPriceModifierBase

Element/Attribute	DO Entity	DO Field	Rule/Note
RetailPriceModifierBase.SequenceNumber	RetailPriceModifier	RetailPriceModifierSequenceNumber	If it does not fall into the smallint/short range or is not unique over all retail price modifiers of the current line item, business error **GKR-100501** is sent back. If it is missing from the request **GKR-100506** will be sent back.
	PriceModificationLineItem	RetailTransactionLineItemSequenceNumber	
RetailPriceModifierBase.Amount	RetailPriceModifier	Amount	For Action = "Add" or "Subtract", absolute value.

<table>
<tr><th>Element/Attribute</th><th>DO Entity</th><th>DO Field</th><th>Rule/Note</th></tr>
<tr><td></td><td>PriceModificationLineItem</td><td>Amount</td><td></td></tr>
<tr><td rowspan="2">RetailPriceModifierBase.Amount.Action</td><td>RetailPriceModifier</td><td>AdjustmentMethodCode</td><td>Mapping:<table><tr><td>Add</td><td>"IC"</td></tr><tr><td>Subtract</td><td>"DC"</td></tr><tr><td>Replace</td><td>n.a. - business error</td></tr><tr><td>For</td><td>n.a. - business error</td></tr><tr><td>any other</td><td>business error GKR-100500</td></tr></table></td></tr>
<tr><td>PriceModificationLineItem</td><td>-</td><td>Only Action = "Subtract" is supported in that context.</td></tr>
<tr><td>RetailPriceModifierBase.Amount.Currency</td><td>-</td><td>-</td><td></td></tr>
<tr><td rowspan="2">RetailPriceModifierBase.Percent</td><td>RetailPriceModifier</td><td>Percent</td><td>Action "Subtract" and "Add" are supported otherwise business error GKR-100500 is sent back. If it contains more than 4 decimal places, business error GKR-100501 is sent back.</td></tr>
<tr><td>PriceModificationLineItem</td><td>Percentage</td><td>Action "Subtract" and "Add" are supported otherwise business error GKR-100500 is sent back. If it contains more than 4 decimal places, business error GKR-100502 is sent back.</td></tr>
<tr><td rowspan="2">RetailPriceModifierBase.Percent.Action</td><td>RetailPriceModifier</td><td>AdjustmentMethodCode</td><td>Mapping:<table><tr><td>Add</td><td>"IC"</td></tr><tr><td>Subtract</td><td>"DC"</td></tr><tr><td>Replace</td><td>n.a. - business error</td></tr><tr><td>For</td><td>n.a. - business error</td></tr><tr><td>any other</td><td>business error GKR-100500</td></tr></table></td></tr>
<tr><td>PriceModificationLineItem</td><td>-</td><td>Only Action = "Subtract" is supported in that context.</td></tr>
<tr><td>RetailPriceModifierBase.PreviousPrice</td><td>RetailPriceModifier</td><td>ExtendedAmountBeforeModification</td><td></td></tr>
</table>

Element/Attribute	DO Entity	DO Field	Rule/Note
	PriceModificationLineItem	ExtendedAmountBeforeModification	
RetailPriceModifierBase.PreviousPrice.Currency	-	-	
RetailPriceModifierBase.NewPrice	RetailPriceModifier	ExtendedAmountAfterModification	
	PriceModificationLineItem	ExtendedAmountAfterModification	
RetailPriceModifierBase.NewPrice.Currency	-	-	
RetailPriceModifierBase.PromotionID	-	-	See PriceDerivationRuleBase
RetailPriceModifierBase.ItemLink	RetailPriceModifier	ProrateFrom	If it does not fall into the smallint/short range, business error **GKR-100501** is sent back. If multiple ItemLink elements occur in the request, **only the first one is processed**.
	PriceModificationLineItem	-	
RetailPriceModifierBase.Quantity	RetailPriceModifier	AppliedQuantity	Regarding to the request: Maximum length is 11 including 3 decimal places. If the number of integer digits exceeds 8 business error **GKR-100502** is returned. If the number of decimal places exceeds 3, **GKR-100501** is sent back. If it is missing for a RetailPriceModifier being not prorated from a Discount, business error **GKR-100506** is sent back. Regarding to the response: In case of prorated RetailPriceModifier which was generated by the calculation engine, this element should not be provided in response.
	PriceModificationLineItem	-	
RetailPriceModifierBase.Rounding	RetailPriceModifier	RoundingAmount	Absolute value. If the request contains a negative value, business error **GKR-100503** is sent back.
	PriceModificationLineItem	RoundingAmount	
RetailPriceModifierBase.Rounding.Currency	-	-	

Element/Attribute	DO Entity	DO Field	Rule/Note
RetailPriceModifierBase.Rounding.RoundingDirection	-	-	(RetailPriceModifier) • if RoundingAmount < 0.0: Down • if RoundingAmount > 0.0: Up • else: no RoundingDirection attribute If the request contains a value not being part of the enumeration (Up, Down), business error **GKR-100500** is sent back.
	-	-	(PriceModificationLineItem) • if RoundingAmount < 0.0: Down • if RoundingAmount > 0.0: Up • else: no RoundingDirection attribute If the request contains a value not being part of the enumeration (Up, Down), business error **GKR-100500** is sent back.
RetailPriceModifierBase.ComputationBaseAmount	RetailPriceModifier	CalculationBaseAmount	Available since interface version 3.0.
RetailPriceModifierBase.ComputationBaseAmount.Currency	-	-	
RetailPriceModifierDomainSpecific.PriceDerivationRule	-	-	If more than one PriceDerivationRule element is sent in request, business error **GKR-100504** is sent back. See mapping of PriceDerivationRuleBase.PriceDerivationRuleBase.
any	-	-	
RetailPriceModifierDomainSpecific.ManualTriggerSequenceNumber	RetailPriceModifier	TriggerSequenceNumber	If it does not fall into the smallint/short range, business error **GKR-100501** is sent back.
	PriceModificationLineItem	TriggerSequenceNumber	

Element/Attribute	DO Entity	DO Field	Rule/Note
RetailPriceModifierDomainSpecific.ExtraAmount	RetailPriceModifier	ExtraAmount	Absolute value. If the request contains a negative value, business error **GKR-100503** is sent back.
	PriceModificationLineItem	ExtraAmount	
RetailPriceModifierDomainSpecific.ExtraAmount.Currency	-	-	
RetailPriceModifierDomainSpecific.ExternalSystemOriginatorFlag	RetailPriceModifier	ExternalSystemOriginatorFlag	If the element is missing in the request, "false" is to be used.
	PriceModificationLineItem	ExternalSystemOriginatorFlag	

10.1.1.1.20 LoyaltyRewardBase

Element/Attribute	DO Entity	DO Field	Rule/Note
PromotionID	LoyaltyRewardLineItem	PromotionID	If the element is missing from the request or is empty, **GKR-100506** will be returned.
	FrequentShopperPointsModifier	PromotionID	
PointsAwarded	LoyaltyRewardLineItem	PointsAwardedCount	Maximum length is 11 including 3 decimal places. If the number of integer digits exceeds 8 business error **GKR-100502** is returned. If the number of decimal places exceeds 3, **GKR-100501** is sent back.
	FrequentShopperPointsModifier	FrequentShopperPointsEarnedCount	
PointsAwarded.Type	-	-	Fix "PointsEarned"
ManualTriggerSequenceNumber	LoyaltyRewardLineItem	TriggerSequenceNumber	If it does not fall into the smallint/short range, business error **GKR-100501** is sent back.
	FrequentShopperPointsModifier	TriggerSequenceNumber	
PointsAwardedAmount	LoyaltyRewardLineItem	FrequentShopperPointsEarnedAmount	

Element/Attribute	DO Entity	DO Field	Rule/Note
	FrequentShopperPointsModifier	FrequentShopperPointsEarnedAmount	
PointsAwardedAmount.Currency	-	-	
ComputationBaseAmount	LoyaltyRewardLineItem	ComputationBaseAmount	
	FrequentShopperPointsModifier	ComputationBaseAmount	
ComputationBaseAmount.Currency	-	-	
ExternalSystemOriginatorFlag	LoyaltyRewardLineItem	ExternalSystemOriginatorFlag	
	FrequentShopperPointsModifier	ExternalSystemOriginatorFlag	
PriceDerivationRule	-	-	See mapping of PriceDerivationRuleBase.
any	-	-	
TypeCode	-	-	If the request element does not contain the value "Award", then business error **GKR-100500** is sent back.

10.1.1.1.21 PriceDerivationRuleBase

In case of prorated RetailPriceModifier or prorated FrequentShopperPointsModifier which were generated by the Pricing Engine, this element should not be provided in response.

Element/Attribute	DO Entity	DO Field	Rule/Note
-	RetailTransactionPromotionPriceDerivationRule	TransactionID	= Transaction.TransactionID
RetailPriceModifierBase.PromotionID	RetailTransactionPromotionPriceDerivationRule	PromotionID	If it is not provided in the request, an internal negative identifier is generated. Note: Generated IDs of promotions are negative in order to avoid collisions with existing internal ones. Generated IDs are kept in response mapping. They can be detected by their negative sign.
LoyaltyRewardBase.PromotionID			

Element/Attribute	DO Entity	DO Field	Rule/Note
PriceDerivationRuleID	RetailTransactionPromotionPriceDerivationRule	PriceDerivationRuleID	The PriceDerivationRuleID element of the request is converted into longint. Even in case of failure, the request message can be processed further - an internal negative identifier is generated in that case. Note: This generated value will be responded back instead of the wrong request value. Entry in RetailTransactionPromotionPriceDerivationRule is created only if the combination TransactionID + PriceDerivationRuleID does not exist yet.
-	RetailTransactionPromotionPriceDerivationRule	PriceDerivationRuleEligibilityID	When mapping the request, a negative value is generated as this information is not included in it. Note: Generated IDs of price derivation rule eligibilities are negative in order to avoid collisions with existing internal ones.
-	RetailPriceModifier	PriceDerivationRuleID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleID
	FrequentShopperPointsModifier	PriceDerivationRuleID	
	PriceModificationLineItem	PriceDerivationRuleID	
	LoyaltyRewardLineItem	PriceDerivationRuleID	
-	RetailPriceModifier	PromotionID	= RetailTransactionPromotionPriceDerivationRule.PromotionID
	FrequentShopperPointsModifier	PromotionID	
	PriceModificationLineItem	PromotionID	
	LoyaltyRewardLineItem	PromotionID	
-	RetailPriceModifier	PriceDerivationRuleEligibilityID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleEligibilityID
	FrequentShopperPointsModifier	PriceDerivationRuleEligibilityID	
	PriceModificationLineItem	PriceDerivationRuleEligibilityID	
	LoyaltyRewardLineItem	PriceDerivationRuleEligibilityID	
Eligibility	-	-	See mapping of PriceDerivationRuleEligibility.

Element/Attribute	DO Entity	DO Field	Rule/Note
any	-	-	
PromotionDescription	RetailTransactionPromotionPriceDerivationRule	PromotionDescription	If it exceeds 255 chars, it is truncated; the request message can be processed further.
PromotionDescriptionMultiLanguage	PromotionTextSO		Descriptions of the OPP promotion in different languages.
PromotionPriceDerivationRuleSequence	RetailTransactionPromotionPriceDerivationRule	PromotionPriceDerivationRuleSequence	If the element is not provided in request, 0 is taken as default.
PromotionPriceDerivationRuleResolution	RetailTransactionPromotionPriceDerivationRule	PromotionPriceDerivationRuleResolution	If the element is not provided in request, 0 is taken as default.
PromotionPriceDerivationRuleTypeCode	RetailTransactionPromotionPriceDerivationRule	PromotionPriceDerivationRuleTypeCode	If it exceeds 4 chars, it is truncated; the request message can be processed further.
TransactionControlBreakCode	RetailTransactionPromotionPriceDerivationRule	TransactionControlBreakCode	If the request element does not contain a value of the enumeration • PO • PC • SP • SU business error **GKR-100500** is sent back. If the request element is missing at all, "SU" is taken as default.
PriceDerivationRuleDescription	RetailTransactionPromotionPriceDerivationRule	PriceDerivationRuleDescription	If it exceeds 255 chars, it is truncated; the request message can be processed further.
PromotionOriginatorTypeCode	RetailTransactionPromotionPriceDerivationRule	PromotionOriginatorTypeCode	If it exceeds 2 chars, it is truncated; the request message can be processed further.
TriggerQuantity	RetailTransactionPromotionPriceDerivationRule	TriggerQuantity	Maximum length is 11 including 3 decimal places. If the number of integer digits exceeds 8 business error **GKR-100502** is returned. If the number of decimal places exceeds 3, **GKR-100501** is sent back.

Element/Attribute	DO Entity	DO Field	Rule/Note
DiscountMethodCode	RetailTransactionPromotionPriceDerivationRule	DiscountMethodCode	If the request element does not contain a value of the enumeration • 00 • 01 • 02 • 03 • 04 business error **GKR-100500** is sent back. If the request element is missing at all, "00" is taken as default.
FrequentShopperPointsFlag	RetailTransactionPromotionPriceDerivationRule	FrequentShopperPointsFlag	If the request element is missing at all, "false" is taken as default.
CustomerGroupLoyaltyPointsDefaultQuantity	RetailTransactionPromotionPriceDerivationRule	CustomerGroupLoyaltyPointsDefaultQuantity	Maximum length is 11 including 3 decimal places. If the number of integer digits exceeds 8 business error **GKR-100502** is returned. If the number of decimal places exceeds 3, **GKR-100501** is sent back.
ProhibitPrintFlag	RetailTransactionPromotionPriceDerivationRule	ProhibitPrintFlag	If the request element is missing at all, "false" is taken as default.
TenderTypeCode	RetailTransactionPromotionPriceDerivationRule	TenderTypeCode	If the length of the element in the request exceeds 4 chars, business error **GKR-100501** is sent back.
PointsConversionAmount	RetailTransactionPromotionPriceDerivationRule	PointsConversionAmount	
PointsConversionAmount.Currency	-	-	
NoEffectOnSubsequentPriceDerivationRulesFlag	RetailTransactionPromotionPriceDerivationRule	NoEffectOnSubsequentPriceDerivationRulesFlag	If the request element is missing at all, "false" is taken as default.
ProhibitTransactionRelatedPriceDerivationRulesFlag	RetailTransactionPromotionPriceDerivationRule	ProhibitTransactionRelatedPriceDerivationRulesFlag	If the request element is missing at all, "false" is taken as default.
ExclusiveFlag	RetailTransactionPromotionPriceDerivationRule	ExclusiveFlag	If the request element is missing at all, "false" is taken as default.
ConcurrenceControlVector	RetailTransactionPromotionPriceDerivationRule	ConcurrenceControlVector	If the length of the element in the request exceeds 20 chars, business error **GKR-100501** is sent back.
AppliedCount	RetailTransactionPromotionPriceDerivationRule	AppliedCount	Maximum length is 3 including no decimal places, if the value does not fit with this limitation, the request cannot be processed.

Element/Attribute	DO Entity	DO Field	Rule/Note
ReceiptLine	RetailTransactionPromotionPriceDerivationRule	ReceiptPrinterName	The text to be printed on the receipt (deprecated: use ReceiptLineMultiLanguage instead). The length must not exceed 40 characters in the request, otherwise the text is truncated.
ReceiptLineMultiLanguage	PromotionTextSO	ReceiptText	The texts to be printed on the receipt in different languages.
ExternalPromotionID	RetailTransactionPromotionPriceDerivationRule	ExternalPromotionID	If it exceeds 255 chars, it is truncated; the request message can be processed further.
ExternalPriceDerivationRuleID	RetailTransactionPromotionPriceDerivationRule	ExternalPriceDerivationRuleID	If it exceeds 255 chars, it is truncated; the request message can be processed further.
CouponPrintoutID	RetailTransactionPromotionPriceDerivationRule	CouponPrintoutID	If it exceeds 40 chars, it is truncated; the request message can be processed further.
CouponPrintoutRule	RetailTransactionPromotionPriceDerivationRule	CouponPrintoutRule	For DiscountMethodCode = 04 only: If the request element does not contain a value of the enumeration • 00 • 01 business error **GKR-100500** is sent back. If the request element is missing at all, "00" is taken as default.
CouponPrintoutText	RetailTransactionPromotionPriceDerivationRule	CouponPrintoutText	
PrintoutValidityPeriod	RetailTransactionPromotionPriceDerivationRule	PrintoutValidityPeriod	Maximum length is 3 including no decimal places, if the value does not fit with this limitation, it is truncated; the request message can be processed further. If the value is not provided at all, then default 0 is chosen.
GiftCertificateExpirationDate	RetailTransactionPromotionPriceDerivationRule	GiftCertificateExpirationDate	The end of the date range in which the gift certificate is valid. This value is only needed for DiscountMethodCode = 03.
ExternalAction	RetailTransactionPromotionPriceDerivationRule		Contains an external action, for example, Shake hands with customer. For more information, see mapping of ExternalActionType.
OperatorDisplayText	RetailTransactionPromotionPriceDerivationRule	PromotionTexts	Texts in different languages shown to the operator.

Element/Attribute	DO Entity	DO Field	Rule/Note
CustomerDisplayText	RetailTransactionPromotionPriceDerivationRule	PromotionTexts	Texts in different languages shown to the customer.
PromotionType	RetailTransactionPromotionPriceDerivationRule	PromotionTypeName	The PromotionType is used to classify promotions. Note: The content of this element is passed to the requestor.
CalculationBaseSequence	RetailTransactionPromotionPriceDerivationRule	CalculationBaseSequence	The CalculationBaseSequence is used to define the basis for applying a promotion. Note: The type of this element is a String although its content has a numeric value. It could also contain a negative value. In that case, the corresponding sign is reflected as a postfix. For example, value "-1" is reflected in this element by type String as follows: "1-". The content of this element is passed to the requestor.

<table>
<tr><th>Element/Attribute</th><th>DO Entity</th><th>DO Field</th><th>Rule/Note</th></tr>
<tr><td>ApplicationType</td><td>RetailTransactionPromotionPriceDerivationRule</td><td>PriceModificationMethodCode</td><td>Mapping:<table>
<tr><td>Manual</td><td>"RM"</td></tr>
<tr><td>Item</td><td>n.a. - business error</td></tr>
<tr><td>MixMatch</td><td>"MM"</td></tr>
<tr><td>FixPrice</td><td>"PS"</td></tr>
<tr><td>PromotionalAward</td><td>n.a. - business error</td></tr>
<tr><td>BuyOneGetOne</td><td>"GP"</td></tr>
<tr><td>None</td><td>"NO"</td></tr>
<tr><td>DiscountSingle</td><td>"RS"</td></tr>
<tr><td>DiscountPercent</td><td>"RP"</td></tr>
<tr><td>DiscountTotal</td><td>"RT"</td></tr>
<tr><td>FixPriceTotal</td><td>"PT"</td></tr>
<tr><td>DiscountPercentTotal</td><td>"TP"</td></tr>
<tr><td>DiscountPercentTotal2</td><td>"T2"</td></tr>
<tr><td>DiscountTotalInterval</td><td>n.a. - business error</td></tr>
<tr><td>ExternalAction</td><td>"EX"</td></tr>
<tr><td>any other</td><td>business error GKR-100500</td></tr>
</table>Note: The calculation engine cannot handle unknown values. If you want to use other values than those mapped above, you have to overwrite the enum adapter.</td></tr>
</table>

10.1.1.1.22 PriceDerivationRuleEligibility

Element/Attribute	DO Entity	DO Field	Rule/Note
-	SaleReturnLineItemModifierCoupon	PromotionID	

<table>
<tr><th>Element/Attribute</th><th>DO Entity</th><th>DO Field</th><th>Rule/Note</th></tr>
<tr><td></td><td>RetailTransactionModifierCoupon</td><td></td><td>= RetailTransactionPromotionPriceDerivationRule.PromotionID</td></tr>
<tr><td rowspan="2">-</td><td>SaleReturnLineItemModifierCoupon</td><td rowspan="2">PriceDerivationRuleID</td><td rowspan="2">= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleID</td></tr>
<tr><td>RetailTransactionModifierCoupon</td></tr>
<tr><td rowspan="2">-</td><td>SaleReturnLineItemModifierCoupon</td><td rowspan="2">PriceDerivationRuleEligibilityID</td><td rowspan="2">= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleEligibilityID</td></tr>
<tr><td>RetailTransactionModifierCoupon</td></tr>
<tr><td rowspan="2">ReferenceID</td><td>SaleReturnLineItemModifierCoupon</td><td rowspan="2">CouponNumber</td><td rowspan="2">Multiple coupons = multiple eligibilities. If the length of the element in the request exceeds 40 chars, business error GKR-100501 is sent back. If the element in the request is missing, business error GKR-100506 is sent back.</td></tr>
<tr><td>RetailTransactionModifierCoupon</td></tr>
<tr><td>any</td><td>-</td><td>-</td><td></td></tr>
<tr><td rowspan="2">ReferenceSequenceNumber</td><td>SaleReturnLineItemModifierCoupon</td><td rowspan="2">CouponSequenceNumber</td><td rowspan="2">If it does not fall into the smallint/short range, business error GKR-100501 is sent back. If the element in the request is missing, business error GKR-100506 is sent back.</td></tr>
<tr><td>RetailTransactionModifierCoupon</td></tr>
<tr><td rowspan="4">Type</td><td>RetailPriceModifier</td><td>EligibilityTypeCode</td><td rowspan="4">Mapping in request:<table><tr><td>StoreCoupon</td><td>"COUP"</td></tr><tr><td>any other value</td><td>GKR-100500 is sent back</td></tr></table>In response: fix StoreCoupon. Note: The calculation engine cannot handle unknown values. If you want to use other values than this one mapped above, you have to overwrite the enum adapter.</td></tr>
<tr><td>FrequentShopperPointsModifier</td><td>EligibilityTypeCode</td></tr>
<tr><td>PriceModificationLineItem</td><td>-</td></tr>
<tr><td>LoyaltyRewardLineItem</td><td>-</td></tr>
</table>

10.1.1.1.23 ExternalActionType

Element/Attribute	DO Entity	DO Field	Rule/Note
ID	RetailTransactionPromotionPriceDerivationRule	ExternalActionID	If the length of the element in the request exceeds 60 chars, business error **GKR-100501** is sent back. If the element in the request is missing, business error **GKR-100506** is sent back.
Text	-	-	See mapping of ExternalActionTextType.
Parameter	-	-	See mapping of ExternalActionParameterType.
any	-	-	

10.1.1.1.24 ExternalActionTextType

Element/Attribute	DO Entity	DO Field	Rule/Note
-	RetailTransactionPromotionExternalActionText	PromotionID	= RetailTransactionPromotionPriceDerivationRule.PromotionID
-	RetailTransactionPromotionExternalActionText	PriceDerivationRuleID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleID
-	RetailTransactionPromotionExternalActionText	PriceDerivationRuleEligibilityID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleEligibilityID
Value (the content of the element itself)	RetailTransactionPromotionExternalActionText	Text	If the length of the element in the request exceeds 255 chars, business error **GKR-100501** is sent back. If the element in the request is missing or empty, business error **GKR-100506** is sent back.
ID	RetailTransactionPromotionExternalActionText	Key.TextID	If the length of the element in the request exceeds 60 chars, business error **GKR-100501** is sent back. If the element in the request is missing, business error **GKR-100506** is sent back. It must be unique in the context of the current external action (applied price derivation rule), otherwise business error **GKR-100505** is sent back.

10.1.1.1.25 ExternalActionParameterType

Element/Attribute	DO Entity	DO Field	Rule/Note
-	RetailTransactionPromotionExternalActionParameter	PromotionID	= RetailTransactionPromotionPriceDerivationRule.PromotionID
-	RetailTransactionPromotionExternalActionParameter	PriceDerivationRuleID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleID
-	RetailTransactionPromotionExternalActionParameter	PriceDerivationRuleEligibilityID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleEligibilityID
Value (the content of the element itself)	RetailTransactionPromotionExternalActionParameter	ParameterValue	If the length of the element in the request exceeds 255 chars, business error **GKR-100501** is sent back. If the element in the request is missing or empty, business error **GKR-100506** is sent back.
ID	RetailTransactionPromotionExternalActionParameter	Key.ParameterID	If the length of the element in the request exceeds 60 chars, business error **GKR-100501** is sent back. If the element in the request is missing, business error **GKR-100506** is sent back. It must be unique in the context of the current external action (applied price derivation rule), otherwise business error **GKR-100505** is sent back.

10.1.1.1.26 FrequentShopperPointsModifierType

extends LoyaltyRewardBase

Element/Attribute	DO Entity	DO Field	Rule/Note
SequenceNumber	FrequentShopperPointsModifier	Key.FrequentShopperPointsModifierSequenceNumber	It must fall into the smallint/short range, otherwise business error **GKR-100501** is sent back. It must be unique in the context of the current line item, otherwise business error **GKR-100505** is sent back.

Element/Attribute	DO Entity	DO Field	Rule/Note
AppliedQuantity	FrequentShopperPointsModifier	AppliedQuantity	Regarding to the request: Maximum length is 11 including 3 decimal places. If the number of integer digits exceeds 8 business error **GKR-100502** is returned. If the number of decimal places exceeds 3, **GKR-100501** is sent back. If it is missing for a FrequentShopperPointsModifier being not prorated from a LoyaltyReward, business error **GKR-100506** is sent back. Regarding to the response: In case of prorated FrequentShopperPointsModifier which was generated by the calculation engine, this element should not be provided in response.
ItemLink	FrequentShopperPointsModifier	ProrateFrom	It must fall into the smallint/short range, otherwise business error **GKR-100501** is sent back. If multiple ItemLink elements occur in the request, only the first one is processed.

10.1.1.1.27 PromotionPriceDerivationRuleReferenceType

Element/Attribute	DO Entity	DO Field	Rule/Note
PromotionID	SaleReturnLineItemModifierReference	Key.PromotionID	= RetailTransactionPromotionPriceDerivationRule.PromotionID
PriceDerivationRuleID	SaleReturnLineItemModifierReference	Key.PriceDerivationRuleID	= RetailTransactionPromotionPriceDerivationRule.PriceDerivationRuleID
ReferenceQuantity	SaleReturnLineItemModifierReference	ReferenceQuantity	Maximum length is 11 including 3 decimal places. If the number of integer digits exceeds 8 business error **GKR-100502** is returned. If the number of decimal places exceeds 3, **GKR-100501** is sent back.
any	-	-	

10.1.1.1.28 PromotionManualTriggerType

Contains manual triggers which were created by the client in order to make the transaction resp. line item eligible for promotions containing eligibilities of type "manual trigger".

Element/Attribute	DO Entity	DO Field	Rule/Note
ManualTriggerSequenceNumber	RetailTransactionPromotionTrigger	Key.TriggerSequenceNumber	It must fall into the smallint/short range, otherwise business error **GKR-100501** is sent back. It must be unique in the context of the current line item (for line item related-manual triggers) resp. the transaction (for transaction-related manual triggers), otherwise business error**GKR-100505** is sent back.
	SaleReturnLineItemPromotionTrigger	TriggerSequenceNumber	
ManualTriggerType	RetailTransactionPromotionTrigger	TriggerType	If the length of the element in the request exceeds 2 chars, business error **GKR-100501** is sent back.
	SaleReturnLineItemPromotionTrigger	TriggerType	
ManualTriggerValue	RetailTransactionPromotionTrigger	TriggerValue	If the length of the element in the request exceeds 255 chars, business error **GKR-100501** is sent back.
	SaleReturnLineItemPromotionTrigger	TriggerValue	
PrivilegeType	RetailTransactionPromotionTrigger	PrivilegeType	If missing from the request **GKR-100506** is sent back. If the sent value is not from values • RS • RP • PS • AM , then **GKR-100500** is sent back.
	SaleReturnLineItemPromotionTrigger	PrivilegeType	
PrivilegeValue	RetailTransactionPromotionTrigger	PrivilegeValue	

Element/Attribute	DO Entity	DO Field	Rule/Note
	SaleReturnLineItemPromotionTrigger	PrivilegeValue	If the privilege type field does not have value "AM" and the privilege value is missing from the request or is empty, then **GKR-100500** is returned. If it contains negative value, then **GKR-100503** is returned.
PrivilegeValue.Currency	-	-	
ManualTriggerSequenceAddend	RetailTransactionPromotionTrigger	TriggerSequenceAddend	
	SaleReturnLineItemPromotionTrigger	TriggerSequenceAddend	
any	-	-	

10.1.1.1.29 PromotionExternalTriggerType

Element/Attribute	DO Entity	DO Field	Rule/Note
ExternalTriggerType	RetailTransactionExternalTrigger	Key.TriggerTypeCode	If it is missing from the request, then **GKR-100506** is sent back.
ExternalTriggerAmount	RetailTransactionExternalTrigger	TriggerAmount	
ExternalTriggerAmount.Currency	-	-	
any	-	-	

10.1.1.1.30 DiscountBase

extends RetailPriceModifierDomainSpecific

Element/Attribute	DO Entity	DO Field	Rule/Note
ProratedFlag	-	-	Fix true, that is: If the attribute is not provided in request, this means "true". If "false" is provided in request, business error **GKR-100500** is sent back. In the response, "true" is always used.

10.1.1.1.31 TenderCouponBase

Element/Attribute	DO Entity	DO Field	Rule/Note
Quantity	RetailTransactionCouponSummary	InputCount	Maximum length is 3 including no decimal places. If the number of integer digits exceeds 3 business error **GKR-100502** is returned. If the number of decimal places exceeds 0, **GKR-100501** is sent back. **If the attribute is missing in the request, it is to be set to 1**.
PrimaryLabel	RetailTransactionCouponSummary	Key.CouponNumber	If the length of the element in the request exceeds 40 chars, business error **GKR-100501** is sent back. If it is not provided in the request at all, business error **GKR-100506** is sent back. If it is not unique (that is, if the same coupon number is used in multiple coupon line items), then **GKR-100505** is sent back.
RewardValue	RetailTransactionCouponSummary	PrivilegeValue	If RewardValue is sent but RewardType is not sent **GKR-100506** is sent back. If RewardValue is sent but RewardType is not valid, business error **GKR-100500** is sent back. If RewardValue is not sent in the request but RewardType is valid, business error **GKR-100506** is sent back.
any	-	-	
AppliedQuantity	RetailTransactionCouponSummary	AppliedCount	In the request the maximum allowed length is 3 including no decimal places. If the number of integer digits exceeds 3 business error **GKR-100502** is returned. If the number of decimal places exceeds 0, **GKR-100501** is sent back. **If the attribute is missing in the request, it is to be set to 0.**
RewardType	RetailTransactionCouponSummary	PrivilegeType	If the request element exists but does not contain a value of the enumeration • RP - reduction percentage • RS - reduction amount • PS - new price amount business error **GKR-100500** is sent back.

10.1.1.1.32 RoundingRuleType

This information is used for rounding the RegularSalesUnitPrice of the line items in the shopping basket. Only those items with FixedPriceFlag = false are subject to rounding. Rounding of them is done in a second

step after the mapping. Furthermore, this information is used for rounding the SaleReturnLineItem.ExtendedAmount at all its occurrences.

Element/Attribute	DO Entity	DO Field	Rule/Note
any	-	-	
RoundingMethod	-	-	If the attribute exists but does not contain a value of the enumeration • Up • Down • Commercial business error **GKR-100500** is sent back. **If the attribute is missing in the request, Commercial is used**.
Multiple	-	-	If the attribute exists but does not contain a valid value (valid means that the digit sequence must consist of 0..* zeroes + either one digit "1" or one digit "5" + 0..* zeroes, a decimal point may occur), business error**GKR-100500** is sent back. **If the attribute is missing in the request, 0.01 is used**.

10.1.1.1.33 DescriptionCommonData

DescriptionCommonData extends String.
This entity contains the descriptive text as, for example, the description text of a business error is in the BusinessErrorCommonData entity:
<Description Language="EN">Text of the business error</Description>
The complete list of possible errors is described in chapter Business Error Codes.

Name	Type	Attribute / Element	Cardinality	Required	Available since interface version	Description / Remarks
Language	String	Attribute		No	2.0	Language code in ISO 639-1 format for the text. The default is EN (English).
value	String				1.0	Description test

10.1.1.2 Enumerations

10.1.1.2.1 CalculationMode

This enum was introduced with version 2.0 and consists of the following values:

- Basket
- LineItem

The values are mapped to the Configuration property calculationMode internally.

10.1.1.2.2 PriceDerivationApplicationTypeCode

This enum consists of the following values and is mapped in the response from com.gk_software.pricing_engine.common.typecodes.PriceModificationMethodTx.

These values correspond to promotion master data object com.gk_software.pricing_engine.model.md.promotion.RebatePromotionConditionRuleSO field priceModificationMethodCode.

PriceDerivationApplicationTypeCode	PriceModificationMethodTx	Description / Remarks
Manual	RM	
Item	-	Not supported by PCE
MixMatch	MM	
FixedPrice	PS	
PromotionalAward	-	Not supported by PCE
BuyOneGetOne	GP	
None	NO	
DiscountSingle	RS	
DiscountPercent	RP	
DiscountTotal	RT	
FixPriceTotal	PT	

PriceDerivationApplicationTypeCode	PriceModificationMethodTx	Description / Remarks
DiscountPercentTotal	TP	
DiscountPercentTotal2	T2	
DiscountTotalInterval	TI	
ExternalAction	EX	
SetPriceTotal	ST	Was introduced with version 2.0 for version 1.0. Requests ST type is mapped to FixPriceTotal.

10.1.2 Request Mapper

The request mapping from client API to internal model is handled by DataMappingRequest classes in package com.gk_software.pricing_engine.psi.sapmapping.request.

Class	Description
DataMappingARTSCodes	Static constants used in mapping.
DataMappingRequestBody	Responsible of map com.sap.ppengine.client.dto.PriceCalculateBase.
DataMappingRequestCustomExtension	Mapping for data not available in PCE DO model as custom extensions.
DataMappingRequestExternalActionParameter	Mapping for com.sap.ppengine.client.dto.ExternalActionParameterType.
DataMappingRequestExternalActionText	Mapping for com.sap.ppengine.client.dto.ExternalActionTextType.
DataMappingRequestFrequentShopperPointsModifier	Mapping for com.sap.ppengine.client.dto.FrequentShopperPointsModifierType.
DataMappingRequestHeader	Responsible to map ARTSCommonHeaderType to transaction.
DataMappingRequestItemBase	Mapping for com.sap.ppengine.client.dto.ItemBase.
DataMappingRequestLineItem	Responsible to map LineItems from ShoppingBasket to transaction.
DataMappingRequestLoyaltyReward	Mapping for com.sap.ppengine.client.dto.LoyaltyRewardBase.
DataMappingRequestMapper	Basic entry point for request mapping.
DataMappingRequestMerchandiseHierarchyGroup	Mapping for com.sap.ppengine.client.dto.MerchandiseHierarchyCommonData.

Class	Description
DataMappingRequestPriceDerivationRuleBase	Mapping for com.sap.ppengine.client.dto.PriceDerivationRuleBase.
DataMappingRequestPriceDerivationRuleEligibility	Mapping for com.sap.ppengine.client.dto.PriceDerivationRuleEligibility.
DataMappingRequestPriceModificationLineItem	Mapping for com.sap.ppengine.client.dto.DiscountBase.
DataMappingRequestPromotionExternalTriggerType	Mapping for com.sap.ppengine.client.dto.PromotionExternalTriggerType.
DataMappingRequestPromotionPriceDerivationRuleReference	Mapping for com.sap.ppengine.client.dto.PromotionPriceDerivationRuleReferenceType.
DataMappingRequestRetailPriceModifier	Mapping for com.sap.ppengine.client.dto.RetailPriceModifierDomainSpecific.
DataMappingRequestRetailTransactionCouponSummary	Mapping for com.sap.ppengine.client.dto.TenderCouponBase.
DataMappingRequestRetailTransactionCustomerGroupAssignment	Mapping of com.sap.ppengine.client.dto.LoyaltyAccountType to custom extension map.
DataMappingRequestRetailTransactionLineItemAssociation	Mapping for com.sap.ppengine.client.dto.ItemBase.ItemLink.
DataMappingRequestRetailTransactionPromotionTrigger	Mapping for com.sap.ppengine.client.dto.PromotionManualTriggerType on basket level.
DataMappingRequestSaleReturnLineItemPromotionTriggerType	Mapping for com.sap.ppengine.client.dto.PromotionManualTriggerType on item level.
DataMappingRequestSaleReturnLineItemSalesOrder	Special mapping for SalesOrder type items.
DataMappingRequestAnyMapperHelper	Any fields request mapping related helper methods.
DataMappingUtils	Common helper methods for mapping.

10.1.3 Response Mapper

The request mapping from client API to internal model is handled by DataResponse classes in package com.gk_software.pricing_engine.psi.sapmapping.response.

Class	Description
DataResponseBody	Response body data mapping.
DataResponseCouponMapperService	Response mapper service for Coupon.

Class	Description
DataResponseExternalActionMapperService	Data response mapper for ExternalAction.
DataResponseExternalTriggerMapperService	Response mapper service for External Triggers.
DataResponseFrequentShopperPointsModifierTypeMapperService	Response mapper service for FrequentShopperPointsModifierType.
DataResponseHeader	Response header data mapping.
DataResponseLineItemMapperService	Service to map LineItemDomainSpecific to the response.
DataResponseLoyaltyRewardService	Data response mapper for LoyaltyReward.
DataResponseMapper	The response of the Calculation Engine.
DataResponseMultiLanguagesService	Data response mapper for MultiLanguages.
DataResponsePromotionManualTriggerTypeMapperService	Data Response mapper for PromotionManualTriggerType.
DataResponsePromotionPriceDerivationRuleReferenceTypeMapperService	Data response mapper for PromotionPriceDerivationRuleReferenceType.
DataResponseRetailPriceModifierDomainSpecificMapperService	Response mapper service for RetailPriceModifierDomainSpecific.
ReponseDataMapperCommons	Response data mapper commons.
DataMappingResponseAnyMapperHelper	Any fields response mapping related helper methods.

10.2 Appendix B: Extension Sample code

10.2.1 Custom Bean context

com.gk-software.pricing-engine/extension/beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <!-- Import core bean contexts -->
  <import resource="classpath:/com.gk-software.pricing-engine/psi-sap/common.xml" />
  <import resource="classpath:/com.gk-software.pricing-engine/psi-sap/beans.xml" />
  <import resource="classpath:/com.gk-software.pricing-engine/dataaccess-sap/beans.xml" />

  <!-- Extension of Transaction Analyzer, Overwrite standard one-->
  <alias name="customCouponTransactionAnalyzer" alias="couponTransactionAnalyzer"/>
  <bean id="customCouponTransactionAnalyzer"
class="com.gk_software.pricing_engine.extension.analyzer.CustomCouponTransactionAnalyzer" />

  <!-- Extension of Eligibility Loader -->
  <bean id="customCouponEligibilityLoader"
class="com.gk_software.pricing_engine.extension.eligibilities.CustomCouponEligibilityLoader"
parent="abstractEligibilityLoader" />
  <!-- Add to existing eligibility loader list -->
  <alias name="customEligibilityLoaderList" alias="eligibilityLoaderList"/>
  <bean id="customEligibilityLoaderList" parent="gkEligibilityLoaderList"
class="org.springframework.beans.factory.config.ListFactoryBean">
    <property name="sourceList">
        <list merge="true">
            <ref bean="customCouponEligibilityLoader"/>
        </list>
    </property>
  </bean>

</beans>
```

10.2.2 External Coupon Context

A custom context for storing external coupon ids:

```
package com.gk_software.pricing_engine.extension;
import java.util.HashSet;
import java.util.Set;
/**
 * Custom context for external coupons
 *
 */
public class ExternalCouponContext {
  private Set<String> externalCouponNumbers = new HashSet<String>();
  public Set<String> getExternalCouponNumbers() {
    return externalCouponNumbers;
  }
  public void addExternalCouponNumber(String externalCouponNumber) {
```

```
    this.externalCouponNumbers.add(externalCouponNumber);
  }
}
```

10.2.3 Extension Context

A helper for accessing the extension context more easily:

```
package com.gk_software.pricing_engine.extension;
import java.util.HashMap;
import java.util.Map;
import com.gk_software.pricing_engine.api.context.Context;

public class ExtensionContext {
  public static final String EXTENSION_KEY = "SAMPLE_EXTENSION";
  /**
   * Get an extension context from custom extensions map
   *
   * @param ctx
   * @param contextType
   * @return
   */
  @SuppressWarnings("unchecked")
  public static <T> T getExtensionContext(Context ctx, Class<T> contextType) {
    Map<Object, Map<Class<?>, Object>> customExtensionMap = ctx.getCustomExtensionMap();
    if(customExtensionMap == null) {
      customExtensionMap = new HashMap<Object, Map<Class<?>,Object>>();
      ctx.setCustomExtensionMap(customExtensionMap);
    }
    Map<Class<?>, Object> ctxExtensionMap = customExtensionMap.get(EXTENSION_KEY);
    if(ctxExtensionMap == null) {
      ctxExtensionMap = new HashMap<Class<?>, Object>();
      customExtensionMap.put(EXTENSION_KEY, ctxExtensionMap);
    }
    return (T) ctxExtensionMap.get(contextType);
  }
  /**
   * Add an extension context to custom extension map
   *
   * @param ctx
   * @param extensionContext
   */
  public static <T> void addExtensionContext(Context ctx, T extensionContext) {
    Map<Object, Map<Class<?>, Object>> customExtensionMap = ctx.getCustomExtensionMap();
    if (customExtensionMap == null) {
      customExtensionMap = new HashMap<Object, Map<Class<?>, Object>>();
      ctx.setCustomExtensionMap(customExtensionMap);
    }
```

```
    Map<Class<?>, Object> ctxExtensionMap = customExtensionMap.get(EXTENSION_KEY);
    if (ctxExtensionMap == null) {
      ctxExtensionMap = new HashMap<Class<?>, Object>();
      customExtensionMap.put(EXTENSION_KEY, ctxExtensionMap);
    }
    ctxExtensionMap.put(extensionContext.getClass(), extensionContext);
  }
}
```

10.2.4 Custom Coupons Handling

10.2.4.1 Custom Coupon Transaction Analyzer

Replacement of standard coupon transaction analyzer to store external coupons in the extension context:

```
public class CustomCouponTransactionAnalyzer implements TransactionAnalyzer {
  /**
   * Logger for this class
   */
  private static final Logger LOGGER =
LoggerFactory.getLogger(CustomCouponTransactionAnalyzer.class);
  @Override
  public void analyzeTransaction(Context context) {
    LOGGER.debug("ENTER [analyzeTransaction]");
    this.getCouponNumbers(context);
    this.logAnalyzedData(context);
    LOGGER.debug("LEAVE [analyzeTransaction]");
  }
  /**
   * Analyze the coupons of the transaction for special external type
   *
   * @param context
   */
  protected void getCouponNumbers(Context context) {
    LOGGER.debug("ENTER [getCouponNumbers]");
    // Get transaction coupon data from context
    Transaction transaction = context.getTransaction();
    RetailTransaction retailTransaction = transaction.getRetailTransaction();
    List<RetailTransactionCouponSummary> retailTransactionCouponSummaryList =
retailTransaction
        .getRetailTransactionCouponSummaryList();
    if (CollectionUtils.isEmpty(retailTransactionCouponSummaryList)) {
      LOGGER.debug("LEAVE [getCouponNumbers] No transaction coupons");
      return;
    }
    for (RetailTransactionCouponSummary retailTransactionCouponSummary :
```

```
retailTransactionCouponSummaryList) {
      String couponNumber = retailTransactionCouponSummary.getKey().getCouponNumber();
      if (StringUtils.isNotEmpty(couponNumber)) {
        // Add coupon to standard context.
        context.getTransactionAnalyzerContext().addCouponNumber(couponNumber);
        // Use SecureLogger to log external provided values to prevent CWE-117:
        // Improper Output Neutralization for Logs
        SecureLogger.debug(LOGGER, "Add  couponNumber {} to context", couponNumber);
        // required for POS mode
        if (context.getConfiguration().getIncrementalCalculationMode() !=
IncrementalCalculationMode.Disabled
            && retailTransactionCouponSummary.getAppliedCount() != null) {
          // Clear applied count in case of incremental mode
          retailTransactionCouponSummary.setAppliedCount(toBigDecimal(0.0));
          SecureLogger.debug(LOGGER, "Reseted coupon with couponNumber {} to
AppliedCount=0.0", couponNumber);
        }
        checkExternalCoupon(context, retailTransactionCouponSummary, couponNumber);
      }
    }
    LOGGER.debug("LEAVE [getCouponNumbers]");
  }
  protected void checkExternalCoupon(Context context, RetailTransactionCouponSummary
retailTransactionCouponSummary,
      String couponNumber) {
    // Check coupons for Privilege Type = EXT
    if ("EXT".equalsIgnoreCase(retailTransactionCouponSummary.getPrivilegeType())) {
      String extCouponNr = "EXT-" + couponNumber;
      // Add to custom extension
      ExternalCouponContext externalCoupons = ExtensionContext.getExtensionContext(context,
          ExternalCouponContext.class);
      if (externalCoupons == null) {
        externalCoupons = new ExternalCouponContext();
        ExtensionContext.addExtensionContext(context, externalCoupons);
      }
      externalCoupons.addExternalCouponNumber(extCouponNr);
      // Use SecureLogger to log external provided values to prevent
      // CWE-117:
      // Improper Output Neutralization for Logs
      SecureLogger.debug(LOGGER, "Add  couponNumber {} to custom context", extCouponNr);
    }
  }

  protected void logAnalyzedData(Context ctx) {
    if (LOGGER.isDebugEnabled()) {
      StringBuilder sb = new StringBuilder();
      sb.append("CustomCouponTransactionAnalyzer summary:");
      Set<String> couponNumbers = ctx.getTransactionAnalyzerContext().getCouponNumberSet();
      if (couponNumbers != null) {
        for (String couponNo : couponNumbers) {
          sb.append(" - Coupon: ");
          sb.append(couponNo);
        }
      }
```

```
      ExternalCouponContext externalCoupons = ExtensionContext.getExtensionContext(ctx,
ExternalCouponContext.class);
      if (externalCoupons != null) {
        for (String couponNo : externalCoupons.getExternalCouponNumbers()) {
          sb.append(" - ExtCoupon: ");
          sb.append(couponNo);
        }
      }
      LOGGER.debug(SecureLogger.sanitizeLogString(sb.toString()));
    }
  }
}
```

10.2.4.2 Custom Coupon Eligibility Loader

A new eligibility loader to simulate loading coupon eligibilities from a different data source:

```
package com.gk_software.pricing_engine.extension.eligibilities;

import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.gk_software.pricing_engine.api.EligibilityLoader;
import com.gk_software.pricing_engine.api.context.Context;
import com.gk_software.pricing_engine.api.context.EligibilityWrapper;
import com.gk_software.pricing_engine.api.context.EligibilityWrapper.EligibilityStatus;
import com.gk_software.pricing_engine.api.utils.ContextUtils;
import com.gk_software.pricing_engine.common.logging.SecureLogger;
import com.gk_software.pricing_engine.common.typecodes.EligibilityTypeCodes;
import com.gk_software.pricing_engine.eligibility_loader.AbstractEligibilityLoader;
import com.gk_software.pricing_engine.extension.ExtensionContext;
import com.gk_software.pricing_engine.extension.ExternalCouponContext;
import
com.gk_software.pricing_engine.model.md.promotion.CouponPromotionConditionEligibilitySO;

/**
 * Coupon eligibility loader.
 *
 * It tries to find {@link CouponPromotionConditionEligibilitySO} for collected
 * coupon data
 * {@link ExternalCouponContext}.
 * If one exists it will be activated.
 */
public class CustomCouponEligibilityLoader extends AbstractEligibilityLoader implements
EligibilityLoader {
```

```
  private static final Logger LOGGER =
LoggerFactory.getLogger(CustomCouponEligibilityLoader.class);

  @Override
  public void execute(Context context) {
    LOGGER.debug("ENTER [execute]");

    // Get custom coupon numbers
    ExternalCouponContext couponContext = ExtensionContext.getExtensionContext(context,
ExternalCouponContext.class);
    Long businessUnitGroupID = context.getBusinessUnitGroupID();

    for (String couponNumber : couponContext.getExternalCouponNumbers()) {

      // Load eligibility from data access layer
      List<CouponPromotionConditionEligibilitySO> loadedEligibilities =
this.dataAccessService.getCouponEligibilities(
          context, businessUnitGroupID, couponNumber,
ContextUtils.getSaleReturnTypeCodesForSales(context));

      // Store loaded Eligibilities in this Context List
      List<EligibilityWrapper<CouponPromotionConditionEligibilitySO>> couponEligibilities =
context
          .getEligibilityLoaderContext().getCouponEligibilitiyList();

      for (CouponPromotionConditionEligibilitySO loadedEligibility : loadedEligibilities) {

        // Create EligibilityWrapper and add the loaded Eligibility to it
        EligibilityWrapper<CouponPromotionConditionEligibilitySO> eligibilityWrapperCoupon =
getObjectFactory()
            .createEligibilityWrapper();

        // Check loaded Eligibility Status Code
        String eligibilityStatusCode = loadedEligibility.getStatusCode();

        // Set status to inactive if not activated in master data
        if (eligibilityStatusCode != null &&
!EligibilityTypeCodes.STATUS_CODE_ACTIVE.equals(eligibilityStatusCode)) {

          SecureLogger.debug(LOGGER, "Coupon eligibility id {} with coupon number {} has not
status 'Active'",
              loadedEligibility.getInternalEligibilityID(),
loadedEligibility.getCouponNumber());
          SecureLogger.trace(LOGGER, "{}", eligibilityWrapperCoupon);

          eligibilityWrapperCoupon.setStatus(EligibilityStatus.INACTIVE);

          continue;
        }

        eligibilityWrapperCoupon.setEligibility(loadedEligibility);
        eligibilityWrapperCoupon.setStatus(EligibilityStatus.ACTIVE);

        // Add EligibilityWrapper to Context List
```

```
        couponEligibilities.add(eligibilityWrapperCoupon);

        // Register the eligbility
        super.registerEligibility(context, eligibilityWrapperCoupon);

        // Activate elgibility
        super.positionEligibilityActivated(context,
loadedEligibility.getInternalEligibilityID(),
            loadedEligibility.getParentEligibilityID());

        LOGGER.debug("--- finished activation of coupon eligibility: {},
consumptionTypeCode: {}",
            loadedEligibility.getInternalEligibilityID(),
loadedEligibility.getConsumptionTypeCode());
      }
    }
    LOGGER.debug("LEAVE [execute]");
  }
}
```

10.2.5 Custom Exclusive Check Handler

A new plugin implementation of the ExclusiveCheckHandler extension point, example of using the ConcurrenceControlVector:

```
package com.gk_software.pricing_engine.pos.extension.concurrency_control_vector;

import java.util.Map;

import org.apache.commons.lang.BooleanUtils;
import org.apache.commons.lang.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.annotation.Order;

import com.gk_software.pricing_engine.api.calculation.ExclusiveCheckHandler;
import com.gk_software.pricing_engine.api.context.Context;
import com.gk_software.pricing_engine.api.result.PositionSummaryResult;
import com.gk_software.pricing_engine.api.result.SummaryResult;
import com.gk_software.pricing_engine.api.rules.PromotionConditionWrapper;
import com.gk_software.pricing_engine.model.md.promotion.PromotionConditionSO;

/**
 *
 * The exclusion vector PromotionPriceDerivationRule.ConcurrenceControlVector
 * consists of 10 digits. Each digit corresponds to a flag for the exclusion of
 * discount.
 *
```

```
 */
@Order(999)
public class ConcurrenceControlVectorExclusiveCheckHandler implements ExclusiveCheckHandler
{

  private static final Logger LOGGER =
LoggerFactory.getLogger(ConcurrenceControlVectorExclusiveCheckHandler.class);

  @Override
  public boolean isExclusiveTransactionCondition(PromotionConditionSO currentCondition,
      PromotionConditionSO condition) {
    final boolean checkExclusionNonExclusive = checkExclusionNonExclusive(currentCondition);

    if (checkExclusionNonExclusive && isExclusiveCondition(condition)) {
      return false;
    }

    final boolean checkExclusionExclusive = checkExclusionExclusive(currentCondition);

    return !(checkExclusionExclusive && BooleanUtils.isFalse(condition.getExclusiveFlag()));
  }

  @Override
  public boolean isExclusiveLineItemCondition(PromotionConditionSO condition) {
    if (condition != null && isExclusiveCondition(condition)) {
      String concurrenceVector = condition.getConcurrenceControlVector();
      return concurrenceVector == null || concurrenceVector.length() > 0 &&
checkExclusionVector03(concurrenceVector);
    }
    return false;
  }

  @Override
  public boolean canApplyConditionExclusiveCheck(Context context, SummaryResult
summaryResult,
      PromotionConditionSO promotionCondition) {
    boolean isLineItemCalculation = summaryResult instanceof PositionSummaryResult;
    return !isLineItemCalculation || !context.isExclusiveTransactionConditionUsed()
        || canConditionBeUsedWithExclusiveTxCondition(promotionCondition);
  }

  @Override
  public boolean canApplyExclusiveCondition(Context context, PromotionConditionSO
testedCondition,
      Long priceDerivationRuleEligibilityID) {
    Map<Long, PromotionConditionWrapper> promotionConditionMap =
context.getPromotionConditionContext()
        .getPromotionConditionMapByEligibilityID();

    PromotionConditionWrapper conditionWrapper =
promotionConditionMap.get(priceDerivationRuleEligibilityID);
    PromotionConditionSO condition = conditionWrapper.getPromotionCondition();

    return !(isExclusiveCondition(condition) && testedCondition.getSequence() >
```

```
condition.getSequence()
        && !this.canBeCombinedWithExclusiveCondition(testedCondition));
  }

  /**
   * Checks if the {@link PromotionConditionSO} is exclusive
   *
   * @param condition
   *          the condition
   * @return True if the condition is exclusive
   */
  protected boolean isExclusiveCondition(PromotionConditionSO condition) {
    return BooleanUtils.isTrue(condition.getExclusiveFlag());
  }

  /**
   * At digit 1
   * <p>
   * value 1 -> A receipt related condition can only be combined with
   * non-exclusive line item related conditions.
   * </p>
   * <p>
   * value 3 -> A receipt related condition cannot be combined with a line item
   * related condition.
   * </p>
   *
   * @param condition
   *          condition to check
   * @return True if condition can be applied
   */
  private boolean checkExclusionNonExclusive(PromotionConditionSO condition) {
    if (condition != null) {
      String concurrenceVector = condition.getConcurrenceControlVector();
      return concurrenceVector != null && concurrenceVector.length() > 1
          && !checkExclusionVector11(concurrenceVector) &&
!checkExclusionVector13(concurrenceVector);
    }
    return false;
  }

  /**
   * At digit 1 value 2 -> A receipt related condition can only be combined with
   * exclusive line item related conditions. value 3 -> A receipt related
   * condition cannot be combined with a line item related condition.
   *
   * @param condition
   *          condition to check
   * @return true is apply
   */

  private boolean checkExclusionExclusive(PromotionConditionSO condition) {
    if (condition != null) {
      String concurrenceVector = condition.getConcurrenceControlVector();
      return concurrenceVector != null && concurrenceVector.length() > 1
```

```
          && !checkExclusionVector12(concurrenceVector) &&
!checkExclusionVector13(concurrenceVector);

    }
    return false;
  }

  /**
   * Checks if the condition can be used with an exclusive transaction related condition
   *
   * @param condition
   *          The {@link PromotionConditionSO}
   * @return True if can be used
   */
  public boolean canConditionBeUsedWithExclusiveTxCondition(final PromotionConditionSO
condition) {
    final String concurrenceControlVector = condition.getConcurrenceControlVector();

    boolean canBeUsed = !(concurrenceControlVector == null
        || concurrenceControlVector.length() > 0 &&
checkExclusionVector01(concurrenceControlVector));

    if (!canBeUsed) {
      LOGGER.debug(
          "Position condition with rule ID {} and eligibility ID {} cannot be used because
of exclusive transaction condition ",
          condition.getInternalRuleID(), condition.getInternalEligibilityID());
    }
    return canBeUsed;
  }

  /**
   * Checks if the condition could be combined with an exclusive condition
   *
   * @param condition
   *          The {@link PromotionConditionSO}
   * @return True if can be combined
   */
  private boolean canBeCombinedWithExclusiveCondition(PromotionConditionSO condition) {
    String concurrenceVector = condition.getConcurrenceControlVector();
    return !(StringUtils.isEmpty(concurrenceVector)
        || concurrenceVector.length() > 3 && checkExclusionVector30(concurrenceVector));
  }

  /**
   * A line item related condition is deleted by an exclusive receipt related condition.
   *
   * @param concurrenceControlVector
   *          The exclusion vector
   * @return True if the exclusion vector's 0 position is 1
   */
  protected boolean checkExclusionVector01(final String concurrenceControlVector) {
    return concurrenceControlVector.charAt(0) == '1';
  }
```

```
  /**
   * A receipt related condition deletes line item discounts.
   *
   * @param concurrenceVector
   *          The exclusion vector
   * @return True if the exclusion vector's 0 position is 3
   */
  protected boolean checkExclusionVector03(String concurrenceVector) {
    return concurrenceVector.charAt(0) == '3';
  }

  /**
   * A receipt related condition can only be combined with non-exclusive line item related
conditions.
   *
   * @param concurrenceVector
   *          The exclusion vector
   * @return True if the exclusion vector's 1 position is 1
   */
  protected boolean checkExclusionVector11(String concurrenceVector) {
    return concurrenceVector.charAt(1) == '1';
  }

  /**
   * // A receipt related condition can only be combined with exclusive line item related
conditions.
   *
   * @param concurrenceVector
   *          The exclusion vector
   * @return True if the exclusion vector's 1 position is 2
   */
  protected boolean checkExclusionVector12(String concurrenceVector) {
    return concurrenceVector.charAt(1) == '2';
  }

  /**
   * A receipt related condition cannot be combined with a line item related condition.
   *
   * @param concurrenceVector
   *          The exclusion vector
   * @return True if the exclusion vector's 1 position is 3
   */
  protected boolean checkExclusionVector13(String concurrenceVector) {
    return concurrenceVector.charAt(1) == '3';
  }

  /**
   * A line item/receipt related condition cannot be combined with a line item/receipt
related exclusive condition.
   *
   * @param concurrenceVector
   *          The exclusion vector
   * @return True if the exclusion vector's 3 position is 01
```

```
   */
  protected boolean checkExclusionVector30(String concurrenceVector) {
    return concurrenceVector.charAt(3) == '0';
  }

  @Override
  public boolean supports(PromotionConditionSO condition) {
    return condition != null;
  }
}
```

10.2.6 Custom Rebate-able Item Filter

A new plugin implementation of the RebatableItemFilterPlugin extension point, example of using the ItemDiscountControlVector:

```
package com.gk_software.pricing_engine.extension.filter.plugin;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.annotation.Order;

import com.gk_software.pricing_engine.api.configuration.Configuration;
import com.gk_software.pricing_engine.api.context.Context;
import com.gk_software.pricing_engine.api.filter.plugin.RebatableItemFilterPlugin;
import com.gk_software.pricing_engine.api.rules.PromotionConditionWrapper;
import com.gk_software.pricing_engine.model.md.promotion.PromotionConditionSO;
import com.gk_software.pricing_engine.model.transaction.SaleReturnLineItem;
import com.gk_software.pricing_engine.model.typecodes.DiscountVectorMethodCodeEnum;

/**
 * Filters items, if they are valid for given promotion based on itemDiscountControlVector
and discountTypeCode.
 * Item is valid for promotion if position in itemDiscountControlVector defined by
discountTypeCode is 1.
 * Example: valid - discountTypeCode=2, itemDiscountControlVector=111
 *      not valid - discountTypeCode=2, itemDiscountControlVector=110
 *
 * @since 2.9.3
 */
@Order(999)
public class DiscountControlVectorFilterPlugin implements RebatableItemFilterPlugin {

  private static final Logger LOGGER =
LoggerFactory.getLogger(DiscountControlVectorFilterPlugin.class);

  @Override
  public boolean supports(PromotionConditionWrapper promotionConditionWrapper) {
```

```
    return promotionConditionWrapper != null &&
        promotionConditionWrapper.getPromotionCondition() != null &&
        promotionConditionWrapper.getPromotionCondition().getItemDiscountControlVector() !=
null &&

!promotionConditionWrapper.getPromotionCondition().getItemDiscountControlVector().isEmpty();
  }

  /**
   * {@link SaleReturnLineItem} is valid for {@link PromotionConditionWrapper}, if
   *
   * {@link Configuration#getDiscountVectorMethod()} is {@link
DiscountVectorMethodCodeEnum#ALL},
   * {@link SaleReturnLineItem#getDiscountTypeCode()} is not null or empty,
   * {@link PromotionConditionSO#getItemDiscountControlVector()} is not null or empty,
   * position in vector defined by discountTypecode is 1.
   *
   * @param context Context.
   * @param srLineItem {@link SaleReturnLineItem} to be checked.
   * @param promotionConditionWrapper Promotion to be applied.
   * @return true, if promotion is possible, else false.
   */
  @Override
  public boolean accept(Context context, SaleReturnLineItem srLineItem,
      PromotionConditionWrapper promotionConditionWrapper) {
    String discountTypeCode = srLineItem != null ? srLineItem.getDiscountTypeCode() : null;
    String itemDiscountControlVector =
promotionConditionWrapper.getPromotionCondition().getItemDiscountControlVector();

    if (discountTypeCode != null && DiscountVectorMethodCodeEnum.ALL.is(
        context.getConfiguration().getDiscountVectorMethod())) {
      int index = discountTypeCode.charAt(0) - '0';
      if (index < 0 || index > 9) {
        LOGGER.error("Not accepted: discountTypecode must be in range 0-9, but is {}",
srLineItem);
        return false;
      }
      else if (index >= itemDiscountControlVector.length() ||
itemDiscountControlVector.charAt(index) != '1') {
        LOGGER.info("Not accepted: pos. {} on {}", discountTypeCode,
itemDiscountControlVector);
        return false;
      }
      else {
        LOGGER.debug("Accepted: pos. {} on {}", discountTypeCode,
itemDiscountControlVector);
        return true;
      }
    }
    LOGGER.trace("Not tested: pos. {} on {} (conf: {})", discountTypeCode,
itemDiscountControlVector,
        context.getConfiguration().getDiscountVectorMethod());
    return true;
```

```
  }
}
```

10.2.7 Mapping of Custom Fields

10.2.7.1 GKModelExtractor implementation sample

Example of mapping any fields to GK XXCustom fields.

```
package com.gk_software.pricing_engine.pos.extension.model_extractor;

import java.util.List;
import java.util.Map;

import org.springframework.core.annotation.Order;

import com.gk_software.gkr.api.txpool.dto.Transaction;
import com.gk_software.pricing_engine.gk.model.plugin.GKModelExtractor;

/**
 * GK model extractor responsible to map any fields to XXCustom[0-15] fields
 *
 * @author GK Software AG
 * @since 3.0.0
 */
@Order(1000)
public class GKModelExtractorImpl implements GKModelExtractor {

  /**
   * Map any fields to XXCustom[0-15] fields
   * @param transactionGK
   *          POS transaction (will set in response)
   */
  @SuppressWarnings("unchecked")
  @Override
  public void mapAnyToCustomFields(
      final com.gk_software.pricing_engine.gk.model.transaction.TransactionGk transactionGK)
{
    if (transactionGK == null) {
      throw new IllegalArgumentException("transaction or engineResponseTransaction can not
be null");
    }

    final Transaction adaptedObject = transactionGK.getAdaptedObject();

    final List<Object> any = transactionGK.getAny();
```

```
    // TODO do we check all fields into any to find a map?
    if (any != null && any.get(0) instanceof Map) {
      extractValues(adaptedObject, any);
    }
  }

  @SuppressWarnings("unchecked")
  private void extractValues(Transaction transaction, List<Object> any) {
    final Map<String, String> extensions = (Map<String, String>) any.get(0);

    if (extensions == null) {
      return;
    }

    for (Map.Entry<String, String> entry : extensions.entrySet()) {
      if (entry.getValue() == null) {
        continue;
      }

      if (entry.getKey().equalsIgnoreCase("xxCustom01")) {
        transaction.setXXCustom01(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom02")) {
        transaction.setXXCustom02(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom03")) {
        transaction.setXXCustom03(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom04")) {
        transaction.setXXCustom04(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom05")) {
        transaction.setXXCustom05(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom06")) {
        transaction.setXXCustom06(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom07")) {
        transaction.setXXCustom07(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom08")) {
        transaction.setXXCustom08(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom09")) {
        transaction.setXXCustom09(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom10")) {
        transaction.setXXCustom10(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom11")) {
        transaction.setXXCustom11(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom12")) {
        transaction.setXXCustom12(entry.getValue());
```

```
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom13")) {
        transaction.setXXCustom13(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom14")) {
        transaction.setXXCustom14(entry.getValue());
      }
      if (entry.getKey().equalsIgnoreCase("xxCustom15")) {
        transaction.setXXCustom15(entry.getValue());
      }
    } // for
  }

  @Override
  public boolean supports(com.gk_software.pricing_engine.gk.model.transaction.TransactionGk
delimiter) {
    return delimiter != null;
  }
}
```

10.2.7.2 GKModelAdapter implementation sample

Example of mapping GK XXCustom fields to any fields.

```
package com.gk_software.pricing_engine.pos.extension.model_adapter;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.core.annotation.Order;

import com.gk_software.gkr.api.txpool.dto.Transaction;
import com.gk_software.pricing_engine.gk.model.plugin.GKModelAdapter;

/**
 * GK model adapter responsible to map GK XXCustom[0-15] fields into any fields.
 *
 * @author GK Software AG
 * @since 3.0.0
 */
@Order(1000)
public class GKModelAdapterImpl implements GKModelAdapter {

  /**
   * Map GK XXCustom[0-15] fields into any fields.
   *
   * @param transactionGK
```

```
   *          the wrapper transaction for internal use in PCE
   */
  @SuppressWarnings("unchecked")
  @Override
  public void mapCustomFieldsToAnyField(
      final com.gk_software.pricing_engine.gk.model.transaction.TransactionGk transactionGK)
{
    if (transactionGK == null) {
      throw new IllegalArgumentException("requestTransaction or transactionGK can not be
null");
    }

    final com.gk_software.gkr.api.txpool.dto.Transaction requestTransaction =
transactionGK.getAdaptedObject();
    final Map<String, String> values = extractValues(requestTransaction);

    final List<Object> any = transactionGK.getAny();
    if (any != null) {
      any.add(values);
    }
  }

  private Map<String, String> extractValues(Transaction requestTransaction) {
    final Map<String, String> values = new HashMap<String, String>();
    final String xxCustom01 = requestTransaction.getXXCustom01();
    if (xxCustom01 != null) {
      values.put("xxCustom01", xxCustom01);
    }
    final String xxCustom02 = requestTransaction.getXXCustom02();
    if (xxCustom02 != null) {
      values.put("xxCustom02", xxCustom02);
    }
    final String xxCustom03 = requestTransaction.getXXCustom03();
    if (xxCustom03 != null) {
      values.put("xxCustom03", xxCustom03);
    }
    final String xxCustom04 = requestTransaction.getXXCustom04();
    if (xxCustom04 != null) {
      values.put("xxCustom04", xxCustom04);
    }
    final String xxCustom05 = requestTransaction.getXXCustom05();
    if (xxCustom05 != null) {
      values.put("xxCustom05", xxCustom05);
    }
    final String xxCustom06 = requestTransaction.getXXCustom06();
    if (xxCustom06 != null) {
      values.put("xxCustom06", xxCustom06);
    }
    final String xxCustom07 = requestTransaction.getXXCustom07();
    if (xxCustom07 != null) {
      values.put("xxCustom07", xxCustom07);
    }
    final String xxCustom08 = requestTransaction.getXXCustom08();
    if (xxCustom08 != null) {
```

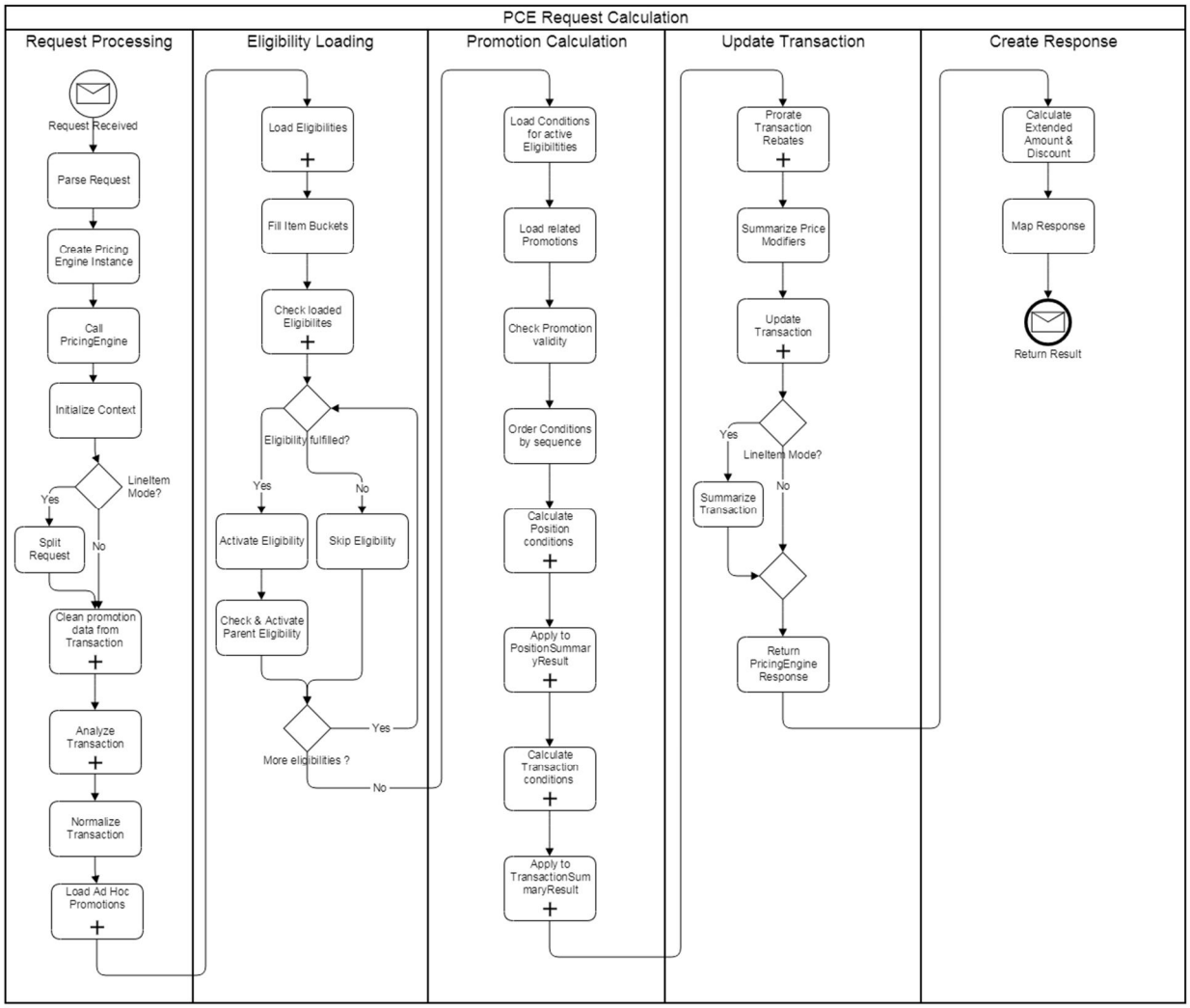
```
      values.put("xxCustom08", xxCustom08);
    }
    final String xxCustom09 = requestTransaction.getXXCustom09();
    if (xxCustom09 != null) {
      values.put("xxCustom09", xxCustom09);
    }
    final String xxCustom10 = requestTransaction.getXXCustom10();
    if (xxCustom10 != null) {
      values.put("xxCustom10", xxCustom10);
    }
    final String xxCustom11 = requestTransaction.getXXCustom11();
    if (xxCustom11 != null) {
      values.put("xxCustom11", xxCustom11);
    }
    final String xxCustom12 = requestTransaction.getXXCustom12();
    if (xxCustom12 != null) {
      values.put("xxCustom12", xxCustom12);
    }
    final String xxCustom13 = requestTransaction.getXXCustom13();
    if (xxCustom13 != null) {
      values.put("xxCustom13", xxCustom13);
    }
    final String xxCustom14 = requestTransaction.getXXCustom14();
    if (xxCustom14 != null) {
      values.put("xxCustom14", xxCustom14);
    }
    final String xxCustom15 = requestTransaction.getXXCustom15();
    if (xxCustom15 != null) {
      values.put("xxCustom15", xxCustom15);
    }
    return values;
  }

  @Override
  public boolean supports(com.gk_software.pricing_engine.gk.model.transaction.TransactionGk
delimiter) {
    return delimiter != null;
  }
}
```

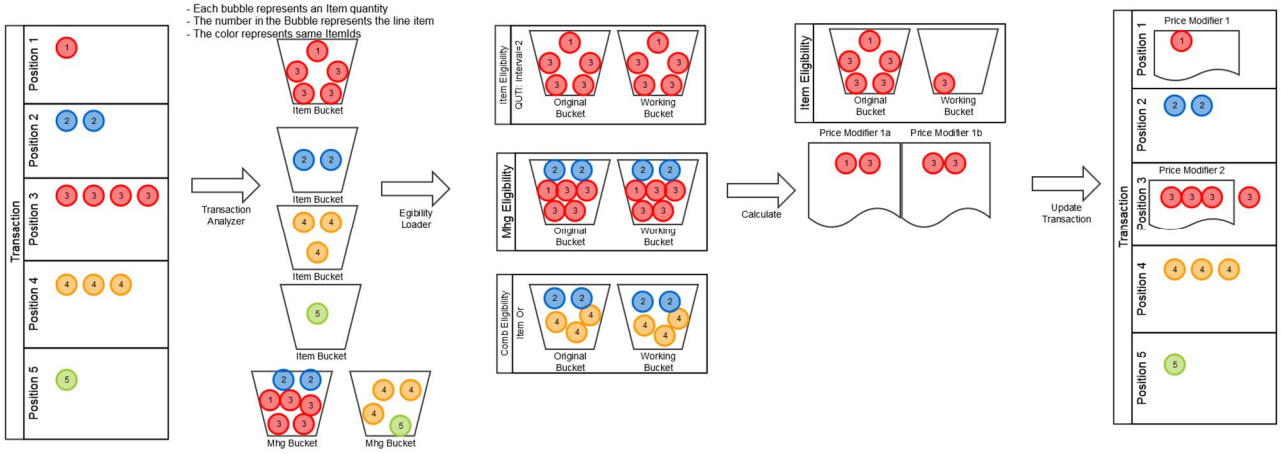
10.3 Appendix C: Diagrams

The sequence diagrams listed in the appendix and used in the documentation reflect the implementation of PCE 2.9.x. As they show the sequence of involved classes and called methods on a higher level only, they are not expected to change with newer releases. That being said, the diagrams will be updated when necessary, to reflect the correct behavior of the PCE.

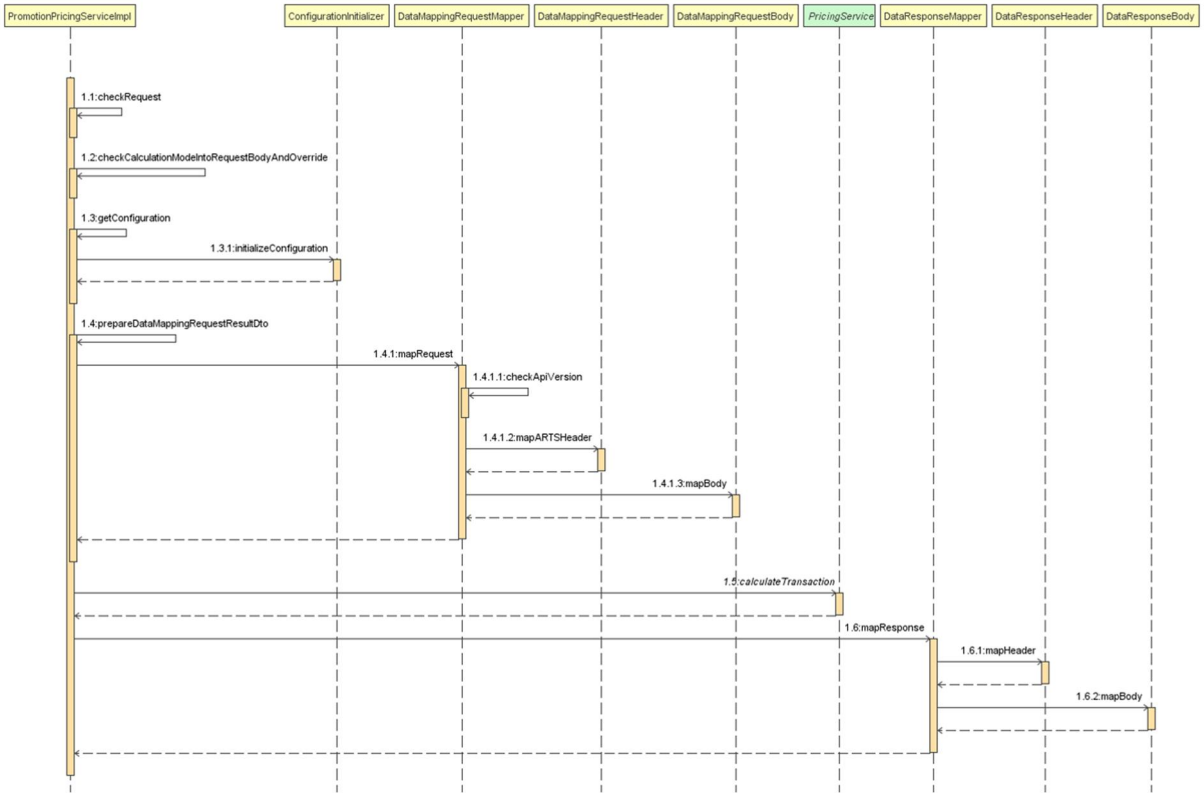
10.3.1 Pricing Engine Processing Flow



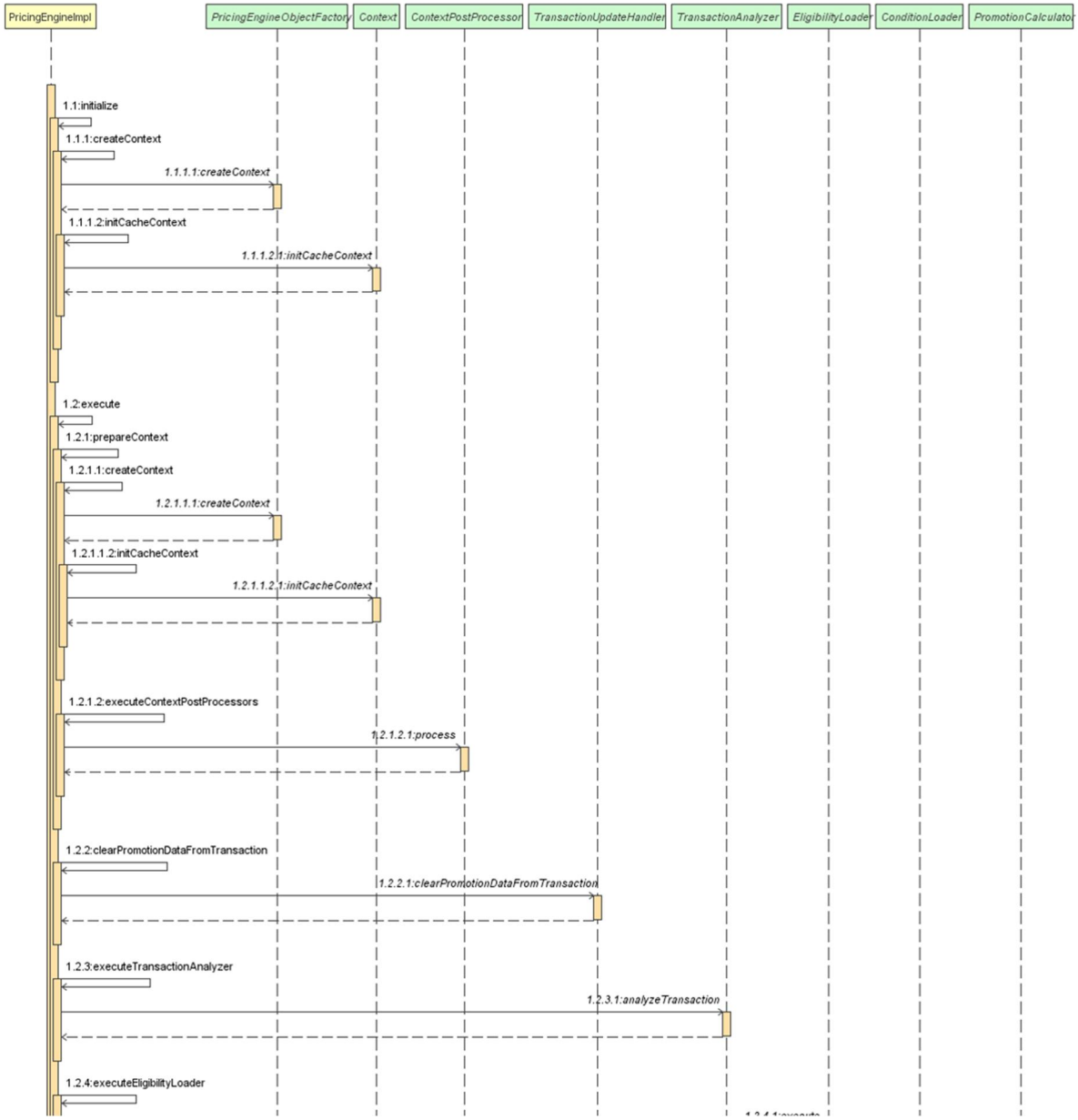
10.3.2 Price Calculation Logic (Buckets) Diagram

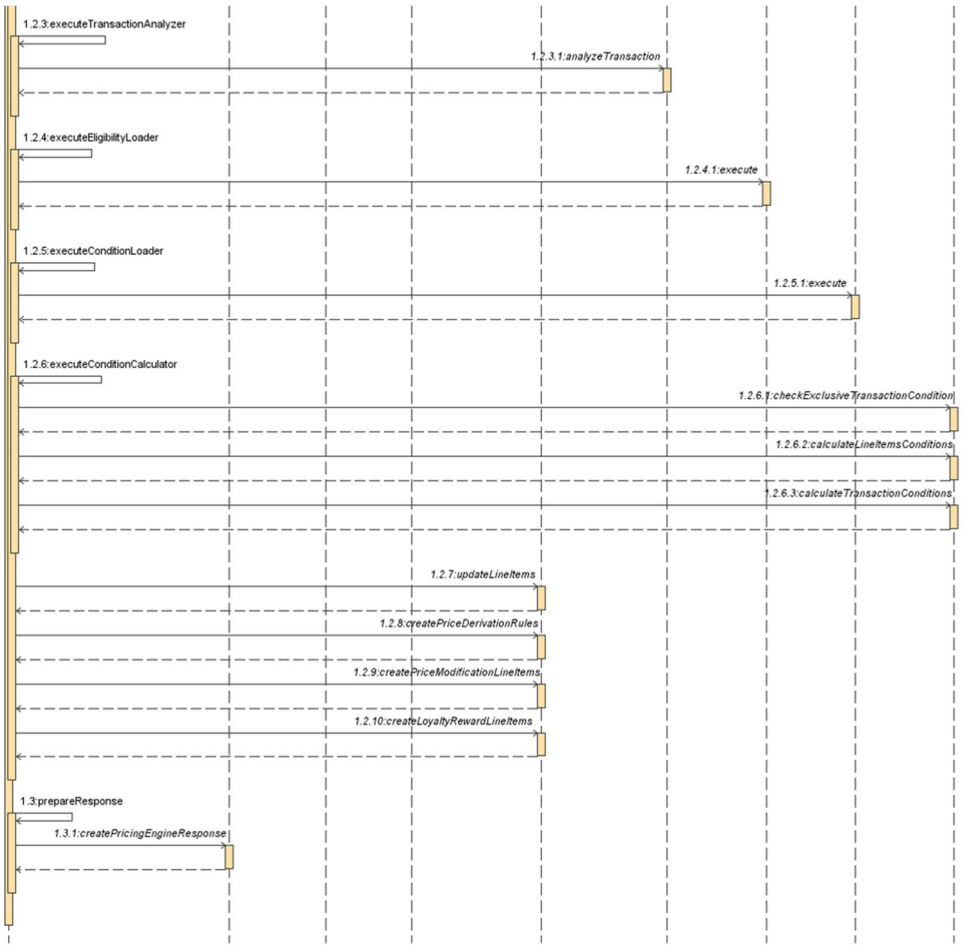


10.3.3 Pricing Engine Request Processing

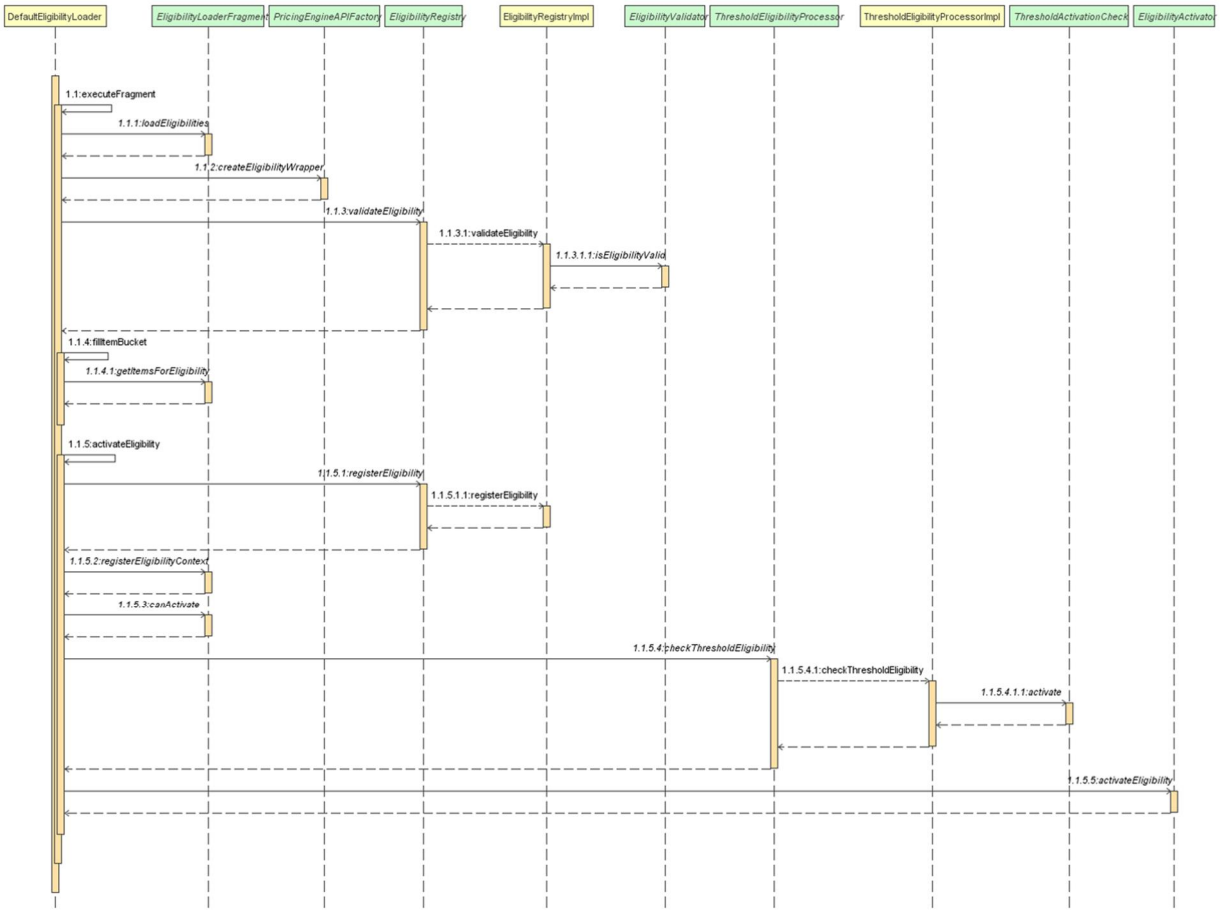


10.3.4 Pricing Engine Calculation Process

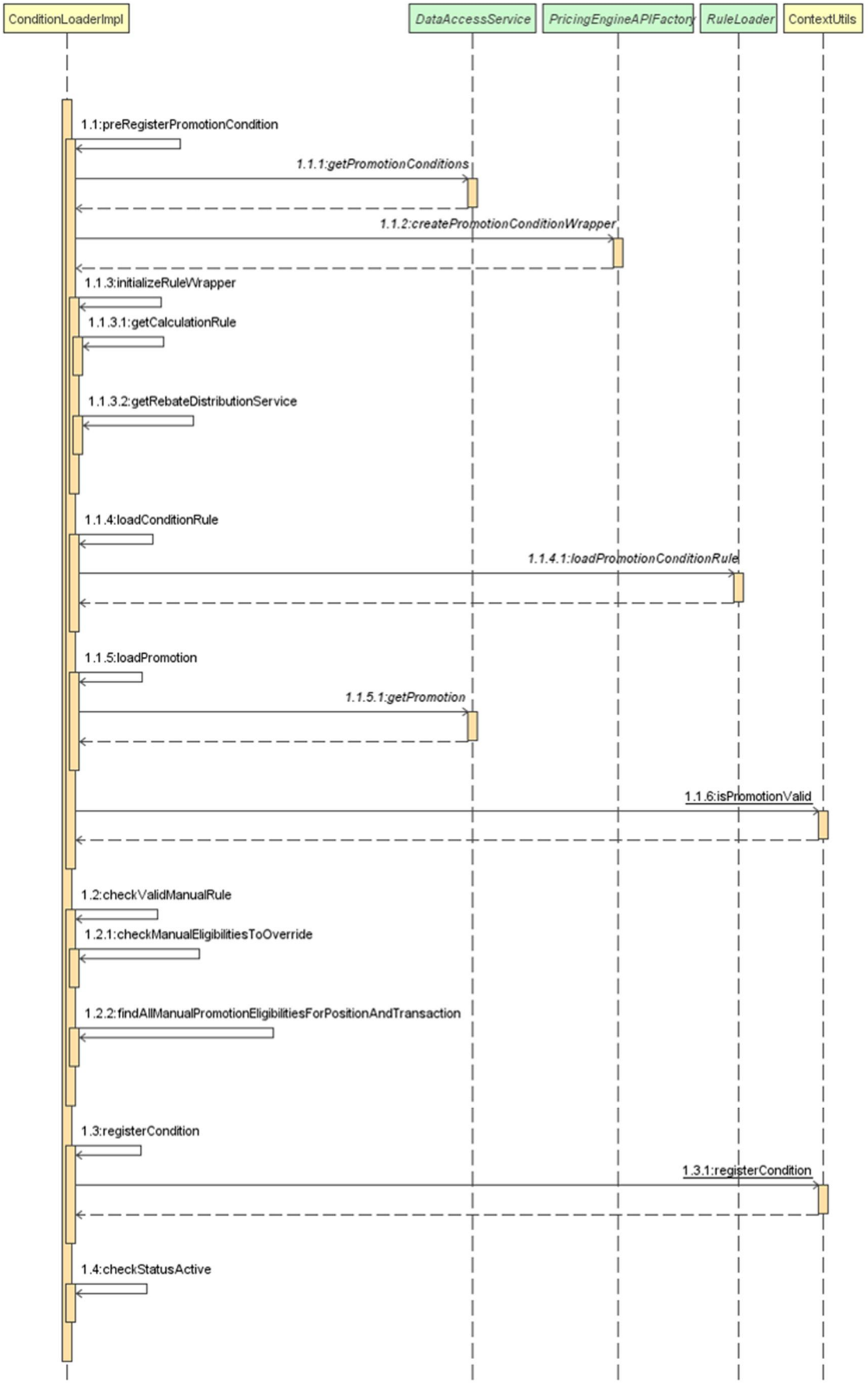




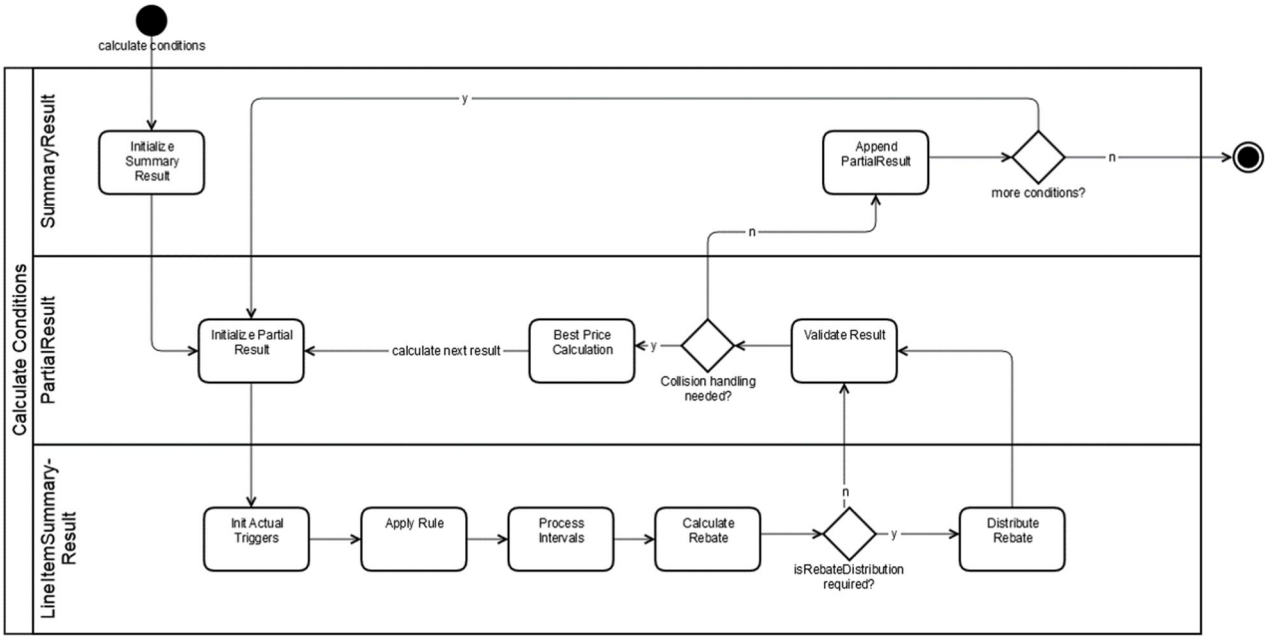
10.3.5 Eligibility Loading Process



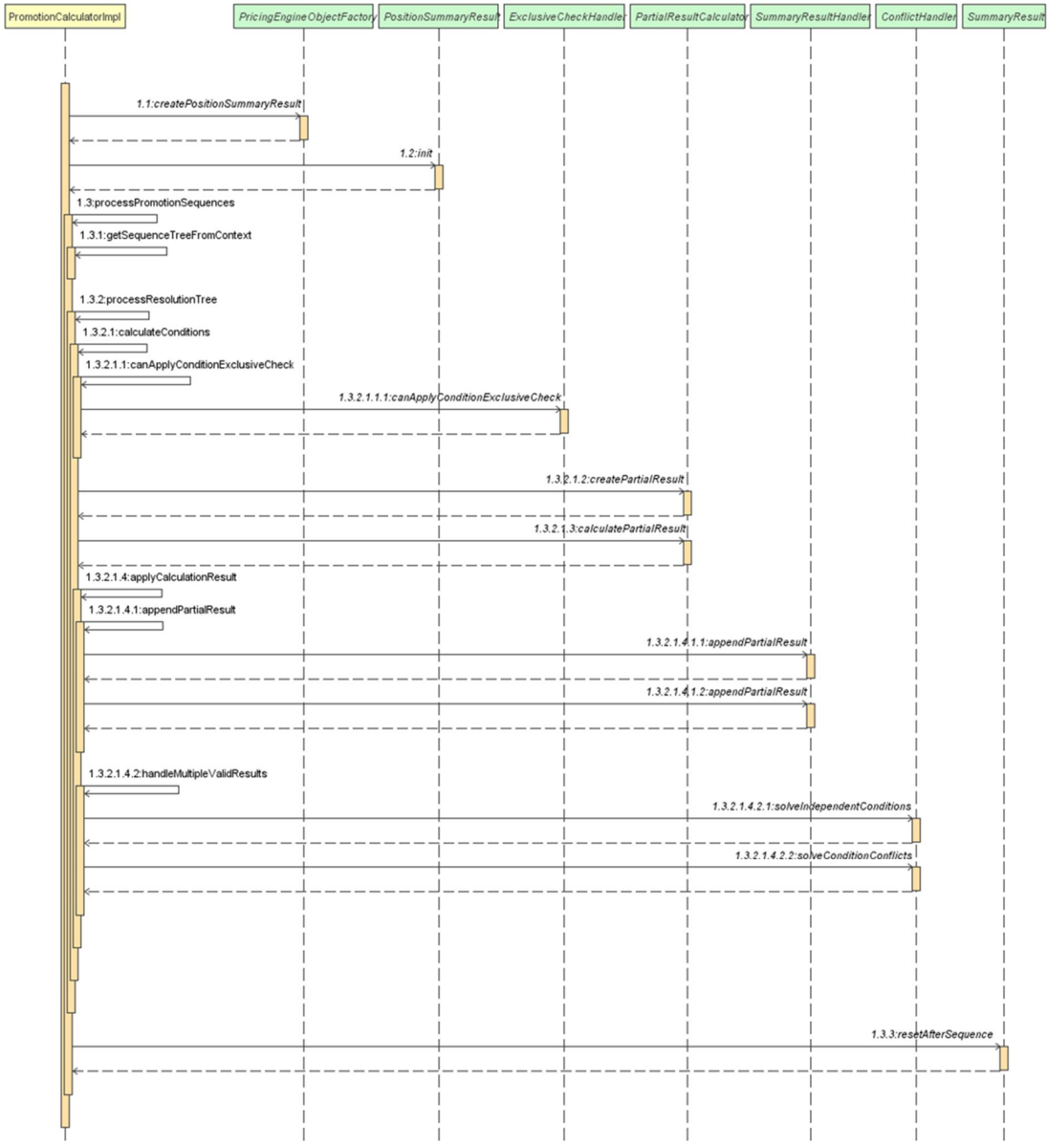
10.3.6 Condition Loading Process



10.3.7 Condition Calculation Process



10.3.8 Promotion Calculation Process



10.3.9 Update Transaction Process


TransactionUpdateHandlerImpl
TransactionCalculationService
1.1:summarizeRetailPriceModifiers
1.1.1:addRetailTransactionPriceModifiers
1.1.1.1:calculateExtraAmount
1.1.1.2:calculateExtraAmount
1.2:summarizeFrequentShopperPointsModifiers
1.2.1:addFrequentShopperPointsModifiers
1.3:updateLineItemsForPositionConditions
1.3.1:updateLineItems
1.3.1.1:addSaleReturnLineItemModifierCoupons
1.3.1.2:addSaleReturnLineItemModifierReferences
1.3.1.3:addSaleReturnLineItemModifierCoupons
1.3.1.4:addSaleReturnLineItemModifierReferences
1.4:updateLineItemsForTransactionConditions
1.4.1:updateLineItems
1.4.1.1:addSaleReturnLineItemModifierCoupons
1.4.1.2:addSaleReturnLineItemModifierReferences
1.4.1.3:addSaleReturnLineItemModifierCoupons
1.4.1.4:addSaleReturnLineItemModifierReferences

www.sap.com/contactsap

Material Number: