PUBLIC
SAP Variant Configuration and Pricing
Document Version: Latest – 2023-10-25

Extension Guide for SAP Variant Configuration and Pricing

THE BEST RUN SAP

Content

1 Extensions

This guide provides an overview of the extension concept of SAP Variant Configuration and Pricing, which allows you to implement variant functions and custom pricing routines in the cloud within the boundaries described in this document.

In variant configuration, the functionality of the standard syntax can be extended by customers with variant functions, this is referred to as *extensions* in this guide. SAP offers a predefined set of variant functions for common cases, which are supported with best performance out-of-the-box by configuration service. For those, please refer to SAP Note 2695561 and referencing SAP Notes.

This does not apply to advanced variant configuration (AVC) in SAP S/4HANA. AVC does not support variant functions, therefore, this extension concept is not applicable. If you are using AVC's pre- and post-processing BAdI, you must enable AVC forwarding as described Forwarding Configuration Requests to Advanced Variant Configuration.

In sales pricing the functionality of the standard pricing customizing can be extended by custom requirements and formulas, also referred to as *user exits* or *extensions* in this guide. SAP offers a predefined set of such pricing user exits, which are supported with best performance out-of-the-box by the pricing service. Further details can be found in Supported Standard Pricing Exits.

It is strongly recommended to use the standard variant functions and pricing exits mentioned above and to reduce the usage of custom extensions to the necessary minimum.

> **i Note**
>
> Extensions only have access to configuration and pricing data provided to them when called by configuration or pricing service. That means that for variant functions only the characteristics of the function's interface definition can be accessed or changed. See Variant Configuration Extension Interface [page 43]. It also means that for custom pricing formulas, only the requested pricing attributes, certain data of the pricing document, the document item, the active price condition, and the price condition to which the formula is assigned can be read. See Pricing Extension Interface [page 46]. Remote Extensions do not have direct access to the tables and data that have been replicated for configuration and pricing services.

We will describe two extension concepts within this document; local extensions and remote extensions. The remote extension concept is complementary to the local extensions. If both, the local and remote extensions are active (hybrid mode) for a tenant, the Variant Configuration and Pricing services will:

1. Verify whether an extension is implemented locally, i.e. code has been uploaded.
2. If a local implementation was found, execute it.
3. If a local implementation was not found, call the REST endpoint maintained for the remote extension implementation.

2 Local Extensions

This section presents the specifics of implementing extensions for the local concept, including the differences compared to the remote extensions.

Variant functions and custom pricing routines that were implemented in ABAP in the back end cannot be run by configuration and pricing services in the cloud. The service can execute uploaded JavaScript code in a secured environment through an embedded script engine though. This code is then executed *locally* within the engine. Therefore, we call it local extensions here in contrast with the remote extensions mentioned in another part of the documentation where a web service is called.

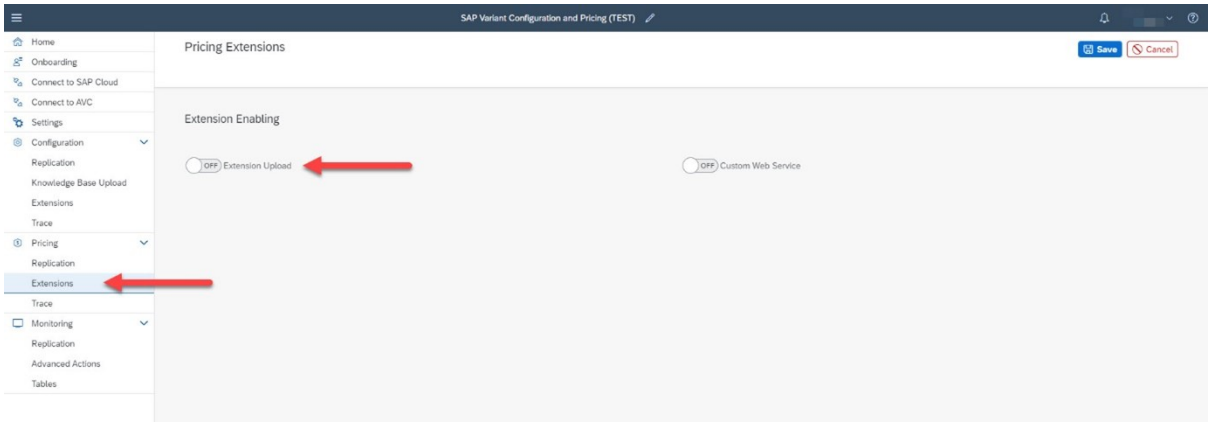
Please note that only variant functions, not pfunctions, are supported for both local and remote extensions. Please refer to Extension API of Pricing Service [page 20] for supported types of pricing extensions for remote and local extensions.

> **i Note**
>
> The customer is responsible for the correctness and the potential performance impact of the uploaded code.

2.1 Administration of Local Extensions

Local extensions can be setup and managed in the administration UI for SAP Variant Configuration and Pricing. Please note that this setup must be done independently for variant configuration and for pricing.

The use of local extensions for a tenant can be enabled/disabled via a toggle switch in the *Extensions* page. Navigate to Configuration or Pricing -> *Extensions* -> *Extension Enabling* and enable *Local Extension*.

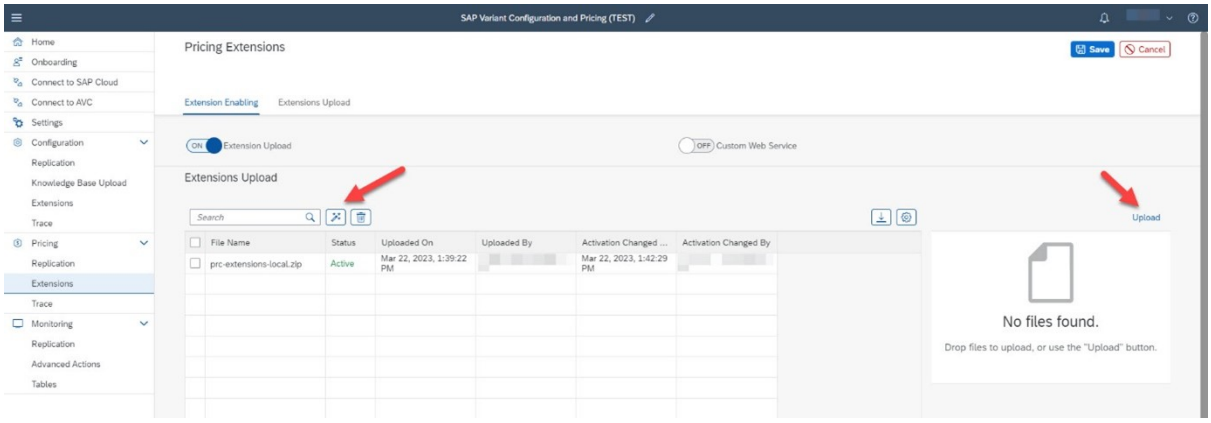


With local extensions enabled, a new section *Extensions Upload* is displayed, it contains the list of uploaded files as well as an upload area.

In the upload area, ZIP-files containing the local extension implementations can be uploaded. Either via drag and drop into the area or by selecting the *Upload* link, which opens a file dialog. Once the ZIP-file has been

uploaded, it must be activated. To do so, select the *Activate uploaded extension* icon in the table header. To delete uploaded files, select the *Delete uploaded extension(s)* icon.

Note that only one ZIP-file can be active at a time.



> **i Note**
>
> The uploaded ZIP-files containing the local extension implementations must have a flat structure, without any folders. The ZIP-file must only contain plain JavaScript files.

2.1.1 Buffering Extension Calls of Local Extensions

For performance reasons, the response of extensions is cached. When an extension is called with the same input data, it is not called again, but the response is read from the cache.

Cache settings

These are the default cache settings:

- Eviction time: 60 minutes. The cache is reset every 60 minutes, all objects are evicted from the cache.
- Eviction counter: if the counter is exceeded, the cache is reset, and all objects are evicted from the cache. These are the counters of each of the services:
  - Pricing: 20.000
  - Configuration: 10.000

> **i Note**
>
> The list of pricing attributes is cached during pricing independently of the buffering mechanism for extensions described here. Therefore, for pricing extensions, calls with action *COLLECT_ATTRIBUTES* are always done only once and cached, independent of the settings described above.

2.2 Implementation of Local Extensions

Local extensions must be implemented in JavaScript and the implementation must be compliant with the *ECMAScript 2021* specification and may not have external dependencies.

> **i Note**
>
> Non-compliant syntax elements, such as NodeJS-specific syntax, are not supported and will lead to an exception during the extension execution in the load phase.

The extensions must be implemented as global functions in a regular JavaScript file and follow a specific naming pattern – this will be further detailed in the section Function Names [page 7].

> **i Note**
>
> Functions must not be implemented inside a JavaScript module.

The API for the local extension interface is the same as for the *remote*, with the input/output in JSON format.

2.2.1 Restrictions

The local extension concept imposes certain restrictions:

- Supported language: only ECMAScript 2021-compliant JavaScript is supported.
- Use of the *console* object is not supported and will lead to an exception.
- Access to Extension Input and Output data as well as APIs provided by SAP, for example the Extension Logger presented in the section Tracing below.

Extensions that cannot fulfill these restrictions must be implemented as Remote Extensions [page 12].

2.2.2 Function Names

Functions implementing an extension must follow a specific naming pattern, so that they can be looked up and executed by the configuration and pricing services.

Variant Configuration:

The function name must exactly match the name of the variant function called in a dependency. The function name must be in upper-case. For example: *ZCPS_DRYER_LEAD_TIME*.

Pricing:

The function name must follow the pattern *<formula_type>_<formula_number>*, where *<formula_type>* is one of *REQ, BAS, VAL, SCL* or *GRP*. The function name must be in upper-case. For example: *BAS_995*.

> **i Note**
>
> The function must implement both *COLLECT_ATTRIBUTES* and *PROCESS_FORMULA* actions.

General Limitations with respect to naming

Function names in local extension implementations may not contain the character *'/'*. Instead, all invocations to functions where the name contains a *'/'* character are mapped to local extension implementation function names, with the *'_'* character instead.

2.2.3 Tracing

Local extensions can be traced by enabling the extension traces in the Administration UI. Navigate to Configuration or Pricing -> *Trace* -> *Extension Trace* and enable the *Trace* toggle.

The input/output of the extension execution is then written to a trace file that can be displayed in the Administration UI. Navigate to Configuration or Pricing -> *Trace* -> *Extension Trace* and check the content of the extension trace table.

> **i Note**
>
> Failed executions are always traced.

2.2.4 Logging

In addition to tracing, logging is also supported.

This is done with the help of the global object `sap`, which provides the method `log()`. This method returns a handle to the logger object, which provides the methods *debug(...)* and *error(...)*. These methods can be called from within the extension code.

Example:

- sap.log().debug('This writes a debug log');
- sap.log().error('This writes an error log');
- var log = sap.log(); log.debug('This writes a debug log');

Logs are written to the same file as the traces and are displayed in the same way.

2.2.5 Database Access

Local Extensions may also read data from database tables via custom select statements. The API for database access can be obtained by calling:

```
var db = sap.db();
```

The *db* object offers methods to construct select statements and execute them. A select statement builder can be obtained by calling:

```
db.select() or db.select("nameOfColumn1", "nameOfColumn2", ...)
```

The returned builder offers the methods:

- "top(int top)" [optional, mandatory max value 100] : limit the number of rows returned
- "columns(String... columnNames)" [optional, default: '*'] : specify the columns to select
- "from(String tableName)" [mandatory] : specify the table to select from
- "where(WhereExpression whereExpression)" [optional] : specify a where clause
- "build()" : construct the select statement

The builder can be supplied with a where clause, termed *WhereExpression*, the construction of which we represent as a tree. *WhereExpressions* can be nested. *WhereExpression* are obtained from the *db* object via the following methods:

- WhereExpression or(WhereExpression... whereExpressions) : disjunction ("or") of >= 2 WhereExpressions
- WhereExpression and(WhereExpression... whereExpressions) : conjunction ("and") of >= 2 WhereExpressions
- WhereExpression not(WhereExpression whereExpression) : negation ("not") of a WhereExpression
- WhereExpression eq(String columnName, Value value) : "equals" comparison of a specifed column and a given Value
- WhereExpression ne(String columnName, Value value) : "not equals" comparison of a specifed column and a given Value
- WhereExpression gt(String columnName, Value value) : "greater than" comparison of a specifed column and a given Value
- WhereExpression gteq(String columnName, Value value) : "greater than or equals" comparison of a specifed column and a given Value
- WhereExpression lt(String columnName, Value value) : "less than" comparison of a specifed column and a given Value
- WhereExpression lteq(String columnName, Value value) : "less than or equals" comparison of a specifed column and a given Value
- WhereExpression in(String columnName, ValueCollection valueCollection) : specifed column has a value "in" the given ValueCollection

Representation of *Value* to use in some *WhereExpression* constructions:

- Value bool(Boolean value) : a boolean value
- Value integer(Integer value) : an integer value
- Value dbl(Double value) : a floating point value
- Value string(String value) : a string value

- Value timestamp(String value) : a timestamp value, given as string
- Value date(String value): a date value, given as string

Representation of *ValueCollection* to use in some *WhereExpression* constructions:

- ValueCollection bools(Boolean... values) : a collection of boolean values
- ValueCollection integers(Integer... values) : a collection of integer values
- ValueCollection dbls(Double... values) : a collection of floating point values
- ValueCollection strings(String... values) : a collection of string values
- ValueCollection timestamps(String... values) : a collection of timestamp values, given as strings
- ValueCollection dates(String... values) : a collection of date values, given as strings

Thus, a select statement can be built like this:

```
var customerId = "id1";
var selectStatement = db.select()
 .columns("COLOUR")
 .from("ZCUSTOMERCOLOUR")
 .where(
     db.eq("CUSTOMER", db.string(customerId))
 ).build();
```

And then executed with via the *execute(Select select)* on the *db* object (timeout 1 second).

```
var dbResult = db.execute(select);
```

The *dbResult* object offers the methods ("rowIndex" and "columnIndex" start at 0):

- int getRowCount()
- int getColumnCount()
- Object get(int rowIndex, int columnIndex)
- String getColumnName(int columnIndex)

For example, retrieve a result like this:

```
if (dbResult.getRowCount() >= 1 && dbResult.getColumnCount() >= 1) {
    var resultRow0Col0 = dbResult.get(0, 0); // row 0, col 0
} else {
    sap.log().error("expected at least one row and one column (in this
example)");
}
```

2.2.6 Error Handling for Local Extensions

Any exception thrown during the execution of a local extension is caught, and the execution is marked as failed.

More information on how configuration and pricing services behave when the execution fails, can be found in API for the Extension of Variant Configuration Service and API for the Extension of Pricing Service and in the section Setting Up Extensions of the Administration Guide for SAP Variant Configuration and Pricing.

2.3 Timeout for Local Extensions

To improve performance and resilience, the configuration and pricing services use a timeout when calling the extension implementation. By default, the timeout is set to 3 seconds. If a timeout error occurs, the execution will be marked as failed. For *sap.db().execute()* there is a timeout set of 1second, after which an exception is thrown and logged to the extension trace.

3 Remote Extensions

Remote extensions of SAP Variant Configuration and Pricing complement the previous local extensions concept. For the variant configuration service, a custom web service, which handles the requests for the extensions can be specified. That service is called by the configuration engine for each variant function, but not for pfunctions. The customer is responsible for the deployment, correctness, and the potential performance impact of the calls on the variant function implementation.

The built-in variant functions start with *SAP_VF* and no custom web services are called for this prefix, see SAP Note 2695561

A custom web service can also be specified for pricing service. That service is called by the pricing engine for each custom user exit. Currently, five user exit types are supported. For details see API for the Extension of Variant Configuration Service and API for the Extension of Pricing Service. The customer is responsible for deployment, correctness, and the potential performance impact of the calls to the user exits.

The implementation of those web services will be provided by the customer via Kyma Functions (separate license needed) or other web technologies. The service endpoint will receive a request from the variant configuration service or the pricing service and will provide the calculated response that is then processed further by the calling SAP service.

> **i Note**
>
> Custom code execution, message serialization and deserialization, and especially network latency will have a noticeable impact on performance. Therefore, the services avoid extension calls with identical input data by buffering. See also Buffering Extension Calls of Remote Extensions [page 13]. Numerous extensions calls and a custom extension code with long execution times, that for example trigger calls to other systems, can lead to a serious degradation of the overall runtime.

3.1 Administration of Remote Extensions

The extension endpoints for the variant configuration and pricing services are configured in the administration UI for SAP Variant Configuration and Pricing. More information can be found in the Administration Guide for SAP Variant Configuration and Pricing.

3.1.1 Authentication

The extension implementations should perform an authorization check to prevent unauthorized calls.

SAP Variant Configuration and Pricing recommends using SAP Destination service to connect configuration and pricing engines to extension implementations via custom web services. Multiple authentication options are

supported by destinations, for example, OAuth2 and basic authentication. For that, you maintain a destination in your tenant / subaccount. The destination includes the URL as well as the authentication/authorization data required to call the extension service.

You maintain the name of the created destination in the administration UI of configuration and pricing services which links the named destination with the subscription tenant. Configuration service and pricing service call the extensions as maintained in the destination. More information about how to maintain and manage destinations can be found in Managing Destinations.

If you configure the destination with OAuth2 or basic authentication, you must also implement OAuth2 or basic authentication check in your extension. Please also note that, if you use a destination with OAuth2, the lifetime of the JWT token must be longer than 5 minutes.

Alternatively, SAP Variant Configuration and Pricing also supports direct extension calls using API keys. For that, you specify in the administration UI, as part of the extension endpoints setup, a request header parameter name. For example: x-api-key or APIKey. Furthermore, you maintain the API key value that shall be sent for that request parameter. Configuration service and pricing service will send that data with each HTTPS request when calling the extensions. The extension implementation must check the parameter value and reject the request with return code *401* for an unexpected API key. See Error Handling of Remote Extensions [page 14].

Depending on the chosen technology or framework to implement the extension, you will get support to generate and check an API key. If you get support, the request header parameter name is given by the framework. If you do not get framework support, you can define the parameter name yourself, generating an UUID as API key and checking it in your implementation.

3.1.2 Buffering Extension Calls of Remote Extensions

For performance reasons, the response of extensions is cached. When an extension is called with the same input data, it is not called again, but the response is read from cache. For pricing extensions, we recommend using *extendedInput* of action *COLLECT_ATTRIBUTES* to specify and minimize the request payload to your needs. This increases the likelihood of calls being buffered. In some scenarios however, the response must not be read from cache. Two mechanisms are available to control whether the response of an extension is cached or not:

- A global buffering switch is available for each service in the administration UI for SAP Variant Configuration and Pricing to activate/deactivate caching for all extensions. This is relevant during development of extensions, for example, when several changes are done and should be tested in a short time. More information can be found in the Prerequisites.
- Some extension implementations, such as an extension that returns a *GUID*, do not always return the same value for the same input, and should therefore not be cached. In this case, the extension implementation should include *Cache-Control: no-store* in its response headers. The Variant Configuration and Pricing services react on this header parameter, and do not cache the response.

Extensions that return with HTTP status code 501 are cached as *not being implemented*. Caching extensions *not being implemented* improve the performance because subsequent calls to the extension are served from the cache and do not add the additional network latency to the overall runtime.

Cache settings:

These are the default cache settings:

- Eviction time: the cache is reset every 60 minutes, all objects are evicted from the cache.
- Eviction counter: if the counter is exceeded, the cache is reset, and all objects are evicted from the cache. These are the counters of each of the services:
  - Pricing: 20.000
  - Configuration: 10.000

> **i Note**
>
> The list of pricing attributes is cached during pricing independently of the buffering mechanism for extensions described here. Therefore, for pricing extensions, calls with action *COLLECT_ATTRIBUTES* are always done only once and cached, independent of the settings described above.

3.2 Implementation of Remote Extensions

Developers of extensions must provide two REST endpoints, which the configuration and pricing services will call when executing extensions. Since a single REST endpoint must be provided for each service, the underlying implementation must either handle all extensions, or delegate to the appropriate implementation based on which extension should be executed. More details on the format of the input data are provided in the following sections.

3.2.1 Error Handling of Remote Extensions

To ensure proper error handling on the application side, the extension implementation should react on, and return the following HTTP error codes:

HTTP Error Code	Error Description
400	Syntax error in the extension input data
401	Authorization error
404	Not found
500	Internal server error
501	The extension with the specified ID has not been implemented

More information on how Variant Configuration and Pricing services behave when the extension implementation returns an error, can be found in Extension API of Variant Configuration Service [page 17] and Extension API of Pricing Service [page 20].

Using the HTTP error code 501 for not implemented extensions is recommended as the extension is then cached, see also Buffering Extension Calls of Remote Extensions [page 13].

3.3 Timeout for Remote Extensions

To enhance performance and resilience, the configuration and pricing services use a timeout when calling the extension implementation. If a timeout error occurs, the configuration and pricing services handle it in a similar way to other HTTP errors. More details can be found in Extension API of Variant Configuration Service [page 17] and Extension API of Pricing Service [page 20]. By default, the timeout is set to 500 ms. The value can be configured independently for the configuration and pricing services in the administration UI.

More information can be found in the section Tracing the Configuration Extension Calls and Tracing the Pricing Extension Calls of the Administration Guide for SAP Variant Configuration and Pricing.

4 Monitoring and Error Handling

The calls to the extension implementations can be monitored in the administration UI for SAP Variant Configuration and Pricing. Errors when calling the extension implementations are always logged. Those errors are calls that return HTTP status code not equal to 200, timeouts, hand-shake problems, etc. More detailed logs can be obtained by activating the traces in the administration UI. All extension call traces are then recorded, even the calls that are served from the cache. Please refer to the Prerequisites for more details on how to activate and view the traces.

An error message is logged if destinations are used for authentication, and the extension cannot be called due to an error getting the destination. A trace with the status code *500* and the message *Failed to execute extension - HTTP client is null due to error getting destination* is prompted. In this case, please verify that the destination has been properly maintained in your tenant/sub-account, and that the destination name has been properly maintained in the administration UI for SAP Variant Configuration and Pricing.

Please note that only the calls to the extension implementations, and any related errors, are traced. Additional logging that may be used to monitor and debug the user extension logic must be taken care of in the extension implementation itself.

More information can be found in the section Tracing the Configuration Extension Calls and Tracing the Pricing Extension Calls of the Administration Guide for SAP Variant Configuration and Pricing.

5 Extension API of SAP Variant Configuration and Pricing

The following two sections describe the extension API of configuration service and pricing service.

5.1 Extension API of Variant Configuration Service

The following sections describe the extension API of variant configuration service in detail and provide examples.

5.1.1 Input Data of Variant Configuration Service

The input data of the Variant Configuration extension interface includes the following information:

- Identifier of the extension type. Note that only *vfun*, for variant function, is currently supported.
- Knowledge base header information, analogous to the data returned by the knowledgebase determination endpoint of the Variant Configuration service.
- A list of variant function arguments:
  - Both input and output, that contain the identifier of the argument: characteristic ID
  - Its data type: string, float or date
  - List of values in string format

5.1.2 Output Data of Variant Configuration Service

The output data of the Variant Configuration extension interface includes the following information:

- Identifier of the extension type. Note that only *vfun*, variant function, is currently supported.
- Result of the variant function: it is true if it was successfully executed. For conditions, this means that the condition is true. For inferences, this means that all output values were successfully calculated.
- List of variant function output arguments with their calculated values.

Note that input/output characteristics of the variant function call are not explicitly identified. In the variant function input a list of so called*fnArgs* which consists of *id, type, value* is provided. For the input arguments of the variant function call all three parameters are provided. For the output arguments the custom implementation has to set the missing values and provide the list of fnArgs in the variantFunctionOutput together with a result true.

Please refer to the OpenAPI documentation found in the Appendix [page 37] for more information about the Variant Configuration extension interface, including object and data types and how they are structured.

5.1.3 Multi-Valued Characteristics

The user extension API supports passing multi-valued characteristics as input/output arguments. However, this is not supported out-of-the-box. If you wish to make use of multi-valued characteristics, you must implement the pilot note 2799818 in the back end where the configuration master data is maintained. This note consists of a change in the variant configuration syntax check, allowing the use of multi-valued characteristics in user extensions.

You can request access to the pilot note via a ticket in the component *LOD-CPS*. When creating the ticket, please choose your system, where you want to implement the note.

Please be aware that even when applying the modification of the SAP Note 2799818, variant configuration will not be able to process variant functions with multi-value characteristics as input or output. You must continue to use the *PFUNCTIONs* in variant configuration and you will have to use the function *SAP_VF_SYSTEM_ENVIRONMENT* from the SAP Note 2790161 to use the corresponding variant functions with multi-value characteristics.

5.1.4 Example Request

In the following example, the custom variant function *ZCPS_VF_VARCONDS* adds values to the characteristic *CPS_VARCOND* depending on the values of the characteristic *CPS_DRYING_ADD_FEATURES.*

```
Content-Type:"application/json"
Accept:"*/*"
APIKey: …
Cache-Control:"no-cache"
Host: …
accept-encoding:"gzip, deflate"
content-length:571
Connection:"keep-alive"
{
    "type": "vfun",
    "vfunInput": {
        "id": "ZCPS_VF_VARCONDS",
        "kbHeaderInfo": {
            "id": 4316,
            "key": {
                "logsys": "R7ECLNT800",
                "kbName": "CPS_DRYING_HOPPER",
                "kbVersion": "1.0"
            },
            "validFromDate": "2019-04-26",
            "changeDate": "2019-04-26",
            "build": 2,
            "structureHash": "9487B6703B211D9853BF522D838BD26B"
        },
        "fnArgs": [{
                "id": "CPS_DRYING_ADD_FEATURES",
                "type": "String",
```

```
                "values": ["SIF", "LIP", "WAS", "FOF"]
            }, {
                "id": "CPS_VARCOND",
                "type": "String",
                "values": []
            }
        ]
    }
}
```

5.1.5 Example Response

```
Content-Type:"application/json; charset=utf-8"
Content-Length:"169"
Connection:"keep-alive"
Access-Control-Allow-Origin:"*"
ETag:"W/"a9-MFERxGISGa/tK+UJbZrtmoBbkUo""
Strict-Transport-Security:"max-age=15724800; includeSubDomains"
{
    "type": "vfun",
    "vfunOutput": {
        "result": true,
        "fnArgs": [
            {
                "id": "CPS_VARCOND",
                "type": "String",
                "values": [
                    "SIF",
                    "LIP",
                    "WAS",
                    "FOF",
                    "SIFLIP_DISCOUNT",
                    "MANY_FEATURE_DISCOUNT"
                ]
            }
        ]
    }
}
```

5.1.6 Executing the Nonoperational Exit

If the extension implementation returns an error, the Variant Configuration service executes a so-called nonoperational function, which does not perform any operation and simply returns *true*. This allows the Variant Configuration service to remain usable if an extension implementation is missing. On the other hand, this approach can result in an incomplete configuration. This nonoperational function is also executed for all custom *pfunctions*.

5.2 Extension API of Pricing Service

The supported extension types are:

- *Requirement - REQ*: Requirement is used during condition finding on pricing procedure at step/counter level, and on condition at access step level. If the extension returns false, access is skipped.
- *Condition Base Formula - BAS*: Condition base formula can be used to change the automatically calculated base value of a condition. The extension is called after the calculation of the condition base value for each pricing condition.
- *Scale Base Formula - SCL*: Scale Base Formula can be used to replace the automatically determined scale base. The extension is called after the calculation of the condition scale base value for a pricing condition. Group scale processing is not supported with the extension concept.
- *Condition Value Formula - VAL*: The Condition Value Formula can be used to replace the automatically determined condition value. The extension is called after the calculation of the condition value for each pricing condition. Group value processing is not supported with the extension concept.
- *Group Key Formula – GRP*: This seldomly used extension influences the grouping of group conditions, which are conditions that are processed together over more than one item. The Group Key Formula is called when the key of a group condition is determined. The method determines, from the different passed object references, a string which is used for the grouping rule of group conditions. Different string values for two conditions will not allow the two conditions to form a group.

5.2.1 Input Data of Pricing Service

The input data of the Pricing extension interface includes the following information:

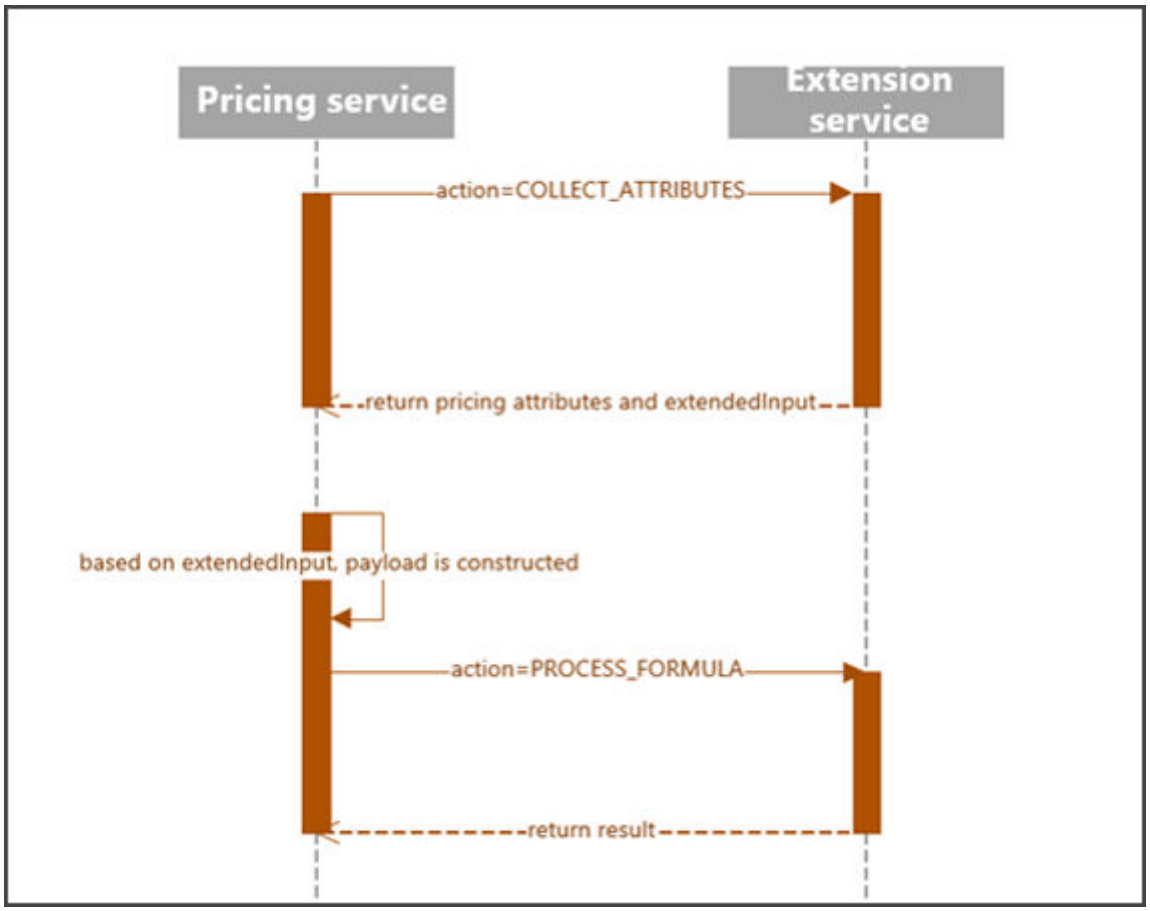
- Formula type of the extension: REQ, BAS, VAL, SCL, GRP.
- Formula number of the extension.
- Actions like *COLLECT_ATTRIBUTES*, *PROCESS_FORMULA*, the default is *PROCESS_FORMULA*.
- The document input of the extension. The document input provides all required information that is required to implement a meaningful extension. The provided input details for the extension are:
  - Document currency
  - Local currency
  - Item details
  - Condition details

5.2.2 Output Data of Pricing Service

The output data of the Pricing extension interface includes the following information:

- *Result*: depending on the user exit type the result is either true, false or a decimal.
  - Boolean: true or false for Requirement (formula type REQ)

    - Decimal: for Condition Base Formula (formula type BAS), Condition Value Formula (formula type VAL), or Scale Base Formula (formula type SCL)
    - String: the overwritten group condition key (formula type GRP)
- A text message that can be looked up in tracing and logging.
- *Item*: returned only for formula type VAL (Condition Value Formula) and BAS (Condition Base Formula) when called with action PROCESS_FORMULA. The pricing item's exclusion indicator and subtotals are overwritten with the passed value.
- *Condition*: returned only for formula type VAL (Condition Value Formula) and BAS (Condition Base Formula) when called with action PROCESS_FORMULA. The pricing condition's statistical flag, inactive flag and condition rate value will be overwritten with the passed values.
- *ExtendedInput*: returned only when the formula is called with the action *COLLECT_ATTRIBUTES*. It is used to tell the pricing engine in more detail which data is required for formula processing. It is recommended to reduce the amount of requested data to the minimum to increase performance and cache hits. It is an optional field and all the details of item and pricing condition are passed when calling the formula with the action *PROCESS_FORMULA*.



Basically, your extension implementation will get as input the calculation results of the current condition type for which the custom routine is executed and the details of the active price condition. The output of the extension is a JSON object where the key *result* contains one calculated value.

Additionally, some document and item details are provided as input. For example, the calculated sub-totals and the pricing attribute values provided by the caller. For the REQ formula, only the exclusion indicator and the

pricing attribute values are provided. By providing the above mentioned *extendedInput*, it is possible to request the condition type *name* as input for the REQ formulas.

By default, extensions do not have access to any other details of the current pricing result, such as other calculated discount or surcharge conditions, or any pricing data stored in the service's cloud database. Accessing other conditions during *PROCESS_FORMULA* must be requested via *extendedInput* during the *COLLECT ATTRIBUTES* call.

The actual calculation is done by extension implementation when called with the action *PROCESS_FORMULA*. When called with the action *COLLECT_ATTRIBUTES*, the extension implementation return a list of required pricing attributes, that must be provided by the calling application. When called with action *COLLECT_ATTRIBUTES*, *documentInput* is passed as a null value.

The response structure *extendedInput* was added to the action *COLLECT_ATTRIBUTES* to not only let the action return the list of required pricing attributes in response to the structure *result*, but also to let it tell the pricing service exactly which data is needed during formula processing with the action *PROCESS_FORMULA*.

Please refer to the OpenAPI documentation in the Appendix [page 37] for more information on the Pricing Extension Interface [page 46], including object and data types and how they are structured.

5.2.3 Sample Requests and Responses

Action COLLECT_ATTRIBUTES:

Example 1: In the following example, with the action *COLLECT_ATTRIBUTES* the custom requirement formula 910 tells the pricing engine that it needs the values for pricing attributes *KOMK-LAND1*, *KOMK-WAERK*, *KOMK-HWAER*, and *KOMP-SKTOF* as input for the action *PROCESS_FORMULA* to be able to process the formula successfully.

Request:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 92
Connection: keep-alive
cache-control: no-cache
{"formulaNumber":910,"formulaType":"REQ","action":"COLLECT_ATTRIBUTES","documentI
nput":null}
```

Response :

```
Content-Type: application/json; charset=utf-8
Content-Length: 77
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": ["KOMK-LAND1","KOMK-WAERK","KOMK-HWAER","KOMP-SKTOF"],
    "message": ""
}
```

Action PROCESS_FORMULA:

Example 1: In the following example, the custom value formula 978 overwrites the condition value 0 EUR, for condition type 0PR0, with value 112 EUR.

Request VAL formula:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 2128
Connection: keep-alive
cache-control: no-cache
{"formulaNumber":978,"formulaType":"VAL","action":"PROCESS_FORMULA","documentInpu
t":{"localCurrency":{"unit":"EUR","numberOfDecimals":2},"documentCurrency":
{"unit":"EUR","numberOfDecimals":2},"itemInput":{"quantity":
{"unit":"EA","internalUnit":"EA","value":1.000},"netValue":0.00,"netPrice":0.00,"
taxValue":0.00,"volume":
{"unit":"M3","internalUnit":"M3","value":1.000},"grossWeight":
{"unit":"KG","internalUnit":"KG","value":1.000},"netWeight":
{"unit":"KG","internalUnit":"KG","value":1.000},"subTotals":
[{"flag":"A","value":0.00},{"flag":"1","value":0.00},
{"flag":"2","value":0.00}],"attributes":[{"name":"KOMK-PLTYP","values":[""]},
{"name":"KOMP-SKTOF","values":["X"]},{"name":"KOMK-HIENR02","values":[""]},
{"name":"KOMK-WERKS","values":[""]},{"name":"KOMK-SPART","values":["00"]},
{"name":"KOMK-WAERK","values":["EUR"]},{"name":"KOMK-HWAER","values":["EUR"]},
{"name":"KOMK-VTWEG","values":["10"]},{"name":"KOMP-PRSFD","values":["X"]},
{"name":"KOMK-KNRZE","values":["GTS002"]},{"name":"KOMK-LAND1","values":["DE"]},
{"name":"KOMK-KONDA","values":["01"]},{"name":"KOMP-PMATN","values":["CPS-
PERF-001"]},{"name":"KOMK-VKORG","values":["1000"]},{"name":"KOMK-
KUNNR","values":["CPS-PERF"]}],"statistical":false,"lastPriceCondition":
{"stepNumber":0,"counter":1,"conditionType":"","calculationType":"C","quantity":
{"unit":"EA","internalUnit":"EA","value":1.000},"conditionBase":1.000,"conditionR
ate":{"unit":"EUR","internalUnit":"EUR","value":0.00},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1.000},"conditionValue":0.00,"conditionC
ontrol":"
","factor":null,"variantConditionFactor":null,"manuallyChanged":false}},"pricingC
ondition":
{"stepNumber":11,"counter":1,"conditionType":"0PR0","calculationType":"C","condit
ionBase":1.000,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":140.00},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1.000},"conditionValue":140.00,"conditio
nClass":"B","structureCondition":"
","purpose":null,"statistical":false,"variantCondition":false,"variantConditionFa
ctor":0,"variantConditionKey":null,"inactiveFlag":"
","recordId":"00016036321","origin":"A","scaleBaseType":"C"}}}
```

Response VAL formula:

```
Content-Type: application/json; charset=utf-8
Content-Length: 61
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": "112.00",
    "message": "",
    "item": null,
    "condition": null
}
```

Example 2: In the following example, the custom requirement formula 910 returns true for the given input, which is mainly the list of pricing attributes.

Request REQ formula:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 669
Connection: keep-alive
cache-control: no-cache
{"formulaNumber":910,"formulaType":"REQ","action":"PROCESS_FORMULA","documentInpu
t":{"localCurrency":null,"documentCurrency":null,"itemInput":
{"quantity":null,"netValue":null,"netPrice":null,"taxValue":null,"volume":null,"g
rossWeight":null,"netWeight":null,"subTotals":null,"attributes":[{"name":"KOMK-
SPART","values":["10"]},{"name":"KOMK-VTWEG","values":["10"]},{"name":"KOMP-
PRSFD","values":["X"]},{"name":"KOMP-PMATN","values":["AK-CAMERA-KIT"]},
{"name":"KOMK-KONDA","values":["01"]},{"name":"KOMK-VKORG","values":["3020"]},
{"name":"KOMK-KUNNR","values":
["2551280"]}],"statistical":false,"lastPriceCondition":null,"exclusionIndicator":
"$"},"pricingCondition":null}}
```

Response REQ formula:

```
Content-Type: application/json; charset=utf-8
Content-Length: 28
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": true,
    "message": ""
}
```

Example 3: In the following example, the custom value formula 980 overwrites the item's subtotals and sets the exclusion indicator for next formulas that will be processed for this item.

Request VAL formula:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 1718
Connection: keep-alive
cache-control: no-cache
{"formulaNumber":980,"formulaType":"VAL","action":"PROCESS_FORMULA","documentInpu
t":{"localCurrency":{"numberOfDecimals":2,"unit":"EUR"},"documentCurrency":
{"numberOfDecimals":2,"unit":"EUR"},"itemInput":{"quantity":
{"unit":"EA","internalUnit":"EA","value":2},"netValue":1000,"netPrice":500,"taxVa
lue":0,"volume":{"unit":"M3","internalUnit":"M3","value":0},"grossWeight":
{"unit":"KG","internalUnit":"KG","value":0},"netWeight":
{"unit":"KG","internalUnit":"KG","value":0},"subTotals":
[{"flag":"1","value":"1000.00"},{"flag":"2","value":"0.00"},
{"flag":"3","value":"0.00"},{"flag":"4","value":"0.00"},
{"flag":"5","value":"0.00"}],"attributes":[{"name":"KOMK-VTWEG","values":["30"]},
{"name":"KOMP-PMATN","values":["AK_CAMERA_KIT"]},{"name":"KOMK-VKORG","values":
["3020"]},{"name":"KOMK-KUNNR","values":
["0000000255"]}],"statistical":false,"lastPriceCondition":{"quantity":
```

```
{"unit":"EA","internalUnit":"EA","value":2},"stepNumber":10,"counter":1,"conditio
nType":"YPR0","calculationType":"C","conditionBase":2,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":500},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1},"conditionValue":1000,"conditionContr
ol":"A","factor":0,"variantConditionFactor":null,"manuallyChanged":false},"exclus
ionIndicator":"$"},"pricingCondition":
{"stepNumber":30,"counter":1,"conditionType":"K005","calculationType":"C","condit
ionBase":2,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":-50},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1},"conditionValue":-100,"conditionClass
":"A","structureCondition":"
","purpose":null,"statistical":false,"variantCondition":false,"variantConditionFa
ctor":0,"variantConditionKey":null,"inactiveFlag":" ","origin":"C"}}}
```


Response VAL formula:


```
Content-Type: application/json; charset=utf-8
Content-Length: 246
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": 1000.00,
    "message": null,
    "item": {
        "exclusionIndicator": "$",
        "subtotals": [
            {
                "flag": "1",
                "value": "100.00"
            },
            {
                "flag": "2",
                "value": "200.00"
            },
            {
                "flag": "3",
                "value": "300.00"
            },
            {
                "flag": "4",
                "value": "400.00"
            },
            {
                "flag": "5",
                "value": "1000.00"
            }
        ]
    },
    "condition": null
}
```


Example 4: In the following example, the custom base formula 915 overwrites the condition base of condition type PR00 with zero and sets the condition to inactive.

Request BAS formula:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 1745
Connection: keep-alive
```

```
cache-control: no-cache
{"formulaNumber":915,"formulaType":"BAS","action":"PROCESS_FORMULA","documentInpu
t":{"localCurrency":{"numberOfDecimals":2,"unit":"EUR"},"documentCurrency":
{"numberOfDecimals":2,"unit":"EUR"},"itemInput":{"quantity":
{"unit":"EA","internalUnit":"EA","value":2},"netValue":1000,"netPrice":500,"taxVa
lue":0,"volume":{"unit":"M3","internalUnit":"M3","value":0},"grossWeight":
{"unit":"KG","internalUnit":"KG","value":0},"netWeight":
{"unit":"KG","internalUnit":"KG","value":0},"subTotals":
[{"flag":"1","value":"1000.00"},{"flag":"2","value":"0.00"},
{"flag":"3","value":"0.00"},{"flag":"4","value":"0.00"},
{"flag":"5","value":"0.00"}],"attributes":[{"name":"KOMK-VTWEG","values":["30"]},
{"name":"KOMP-PMATN","values":["AK_CAMERA_KIT"]},{"name":"KOMK-VKORG","values":
["3020"]},{"name":"KOMK-KUNNR","values":
["0000000255"]}],"statistical":false,"lastPriceCondition":{"quantity":
{"unit":"EA","internalUnit":"EA","value":2},"stepNumber":10,"counter":1,"conditio
nType":"YPR0","calculationType":"C","conditionBase":2,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":500},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1},"conditionValue":1000,"conditionContr
ol":"A","factor":0,"variantConditionFactor":null,"manuallyChanged":false},"exclus
ionIndicator":"$"},"pricingCondition":
{"stepNumber":20,"counter":1,"conditionType":"PR00","calculationType":"C","condit
ionBase":2,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":500},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1},"conditionValue":0,"conditionClass":"
B","structureCondition":"
","purpose":"BASE","statistical":false,"variantCondition":false,"variantCondition
Factor":0,"variantConditionKey":null,"inactiveFlag":"
","recordId":"0001603683","origin":"A"}}}
```


Response BAS formula:


```
Content-Type: application/json; charset=utf-8
Content-Length: 91
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": 0,
    "message": null,
    "item": null,
    "condition": {
        "statistical": null,
        "inactiveFlag": "Z"
    }
}
```


Example 5

In the following example, the custom scale formula 915 overwrites the scale base of condition type PR00 with 10.000 which is determined from the gross weight of the item.

Request SCL formula:


```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 1741
Connection: keep-alive
cache-control: no-cache
{"formulaNumber":915,"formulaType":"SCL","action":"PROCESS_FORMULA","documentInpu
t":{"localCurrency":{"numberOfDecimals":2,"unit":"EUR"},"documentCurrency":
{"numberOfDecimals":2,"unit":"EUR"},"itemInput":{"quantity":
```

```
{"unit":"EA","internalUnit":"EA","value":2},"netValue":1000,"netPrice":500,"taxVa
lue":0,"volume":{"unit":"M3","internalUnit":"M3","value":0},"grossWeight":
{"unit":"KG","internalUnit":"KG","value":10.000},"netWeight":
{"unit":"KG","internalUnit":"KG","value":0},"subTotals":
[{"flag":"1","value":"1000.00"},{"flag":"2","value":"0.00"},
{"flag":"3","value":"0.00"},{"flag":"4","value":"0.00"},
{"flag":"5","value":"0.00"}],"attributes":[{"name":"KOMK-VTWEG","values":["30"]},
{"name":"KOMP-PMATN","values":["AK_CAMERA_KIT"]},{"name":"KOMK-VKORG","values":
["3020"]},{"name":"KOMK-KUNNR","values":
["0000000255"]}],"statistical":false,"lastPriceCondition":{"quantity":
{"unit":"EA","internalUnit":"EA","value":2},"stepNumber":10,"counter":1,"conditio
nType":"YPR0","calculationType":"C","conditionBase":2,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":500},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1},"conditionValue":1000,"conditionContr
ol":"A","factor":0,"variantConditionFactor":null,"manuallyChanged":false},"exclus
ionIndicator":"$"},"pricingCondition":
{"stepNumber":20,"counter":1,"conditionType":"PR00","calculationType":"C","condit
ionBase":2,"conditionRate":
{"unit":"EUR","internalUnit":"EUR","value":500},"conditionUnit":
{"unit":"EA","internalUnit":"EA","value":1},"conditionValue":0,"conditionClass":"
B","structureCondition":"
","purpose":"BASE","statistical":false,"variantCondition":false,"variantCondition
Factor":0,"variantConditionKey":null,"inactiveFlag":"
","origin":"A","scaleBaseType":"B"}}}
```


Sample COLLECT_ATTRIBUTES responses and resulting PROCESS_FORMULA requests: Below you find a few examples which show how the request payload for action *PROCESS_FORMULA* is constructed based on the response of *COLLECT_ATTRIBUTES* using response structure *extendedInput*.

Example 1: In this example, custom value formula 992 requests the condition value of the condition to which the formula is assigned and suppresses all other item and condition data.

Response VAL|992|COLLECT_ATTRIBUTES:

```
Content-Type: application/json; charset=utf-8
Content-Length: 231
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": [],
    "message": null,
    "item": null,
    "condition": null,
    "extendedInput": {
        "documentInput": {
            "itemInput": {
                "projection": [],
                "conditions": null
            },
            "pricingCondition": {
                "projection":
["conditionValue","recordId","origin","scaleBaseType"]
            }
        }
    }
}
```

1. itemInput->projection field is [] i.e. empty list, that means formula does not need any of the item fields as input.
2. itemInput->conditions is null that means formula doesn't need any other conditions of the same item.

3. pricingCondition->projection is ["conditionValue","recordId","origin","scaleBaseType"] that means formula requires conditionValue, recordId, origin and scaleBaseType fields of pricingCondition for which formula is executed.

Based on the above *extendedInput* response, below is the payload generated when custom value formula 992 is called with action *PROCESS_FORMULA*.

Request VAL|992|PROCESS_FORMULA:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 311
Connection: keep-alive
cache-control: no-cache
{
    "formulaNumber": 992,
    "formulaType": "VAL",
    "action": "PROCESS_FORMULA",
    "documentInput": {
        "localCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "documentCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "itemInput": {},
        "pricingCondition": {
            "conditionValue": 0.75,
            "recordId": "0462449197",
            "origin": "A",
            "scaleBaseType": "B"
        }
    }
}
```

Example 2: In this example, custom base formula 990 requests the variant condition key of the condition to which the formula is assigned and some fields of other conditions VA00, ZVA4 and ZVA5 of the same item.

Response BAS|990|COLLECT_ATTRIBUTES:

```
Content-Type: application/json; charset=utf-8
Content-Length: 437
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": [],
    "message": null,
    "item": null,
    "condition": null,
    "extendedInput": {
        "documentInput": {
            "items": null,
            "itemInput": {
                "projection": ["conditions"],
                "conditions": {
                    "filter": {
                        "conditionType": ["VA00", "ZVA4", "ZVA5"]
```

```
                    },
                    "projection": ["conditionValue", "variantCondition",
"variantConditionKey", "inactiveFlag", "conditionType", "counter", "stepNumber"]
                }
            },
            "pricingCondition": {
                "projection": ["variantConditionKey", "conditionType",
"counter", "stepNumber"]
            }
        }
    }
}
```

1. itemInput->projection field is *["conditions"]*, that means formula requires information about other conditions of the same item.
2. itemInput->conditions->filter->conditionType is [*"VA00"*, *"ZVA4"*, *"ZVA5"*], that means formula requires other conditions of the same item with condition types VA00, ZVA4 and ZVA5.
3. itemInput->conditions->projection is [*"conditionValue"*, *"variantCondition"*, *"variantConditionKey"*, *"inactiveFlag"*, *"conditionType"*, *"counter"*, *"stepNumber"*] that means that the formula requires the mentioned condition fields.
4. pricingCondition->projection is [*"variantConditionKey"*, *"conditionType"*, *"counter"*, *"stepNumber"*] that means that the formula requires the mentioned fields of pricingCondition for which the formula is executed.

Based on the above *extendedInput* response, below is the payload generated when custom base formula 990 is called with action PROCESS_FORMULA. The calculated pricing conditions that have the same step number as the pricingCondition for which the formula is executed, are also passed.

Request BAS|990|PROCESS_FORMULA

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 773
Connection: keep-alive
cache-control: no-cache
{
    "formulaNumber": 990,
    "formulaType": "BAS",
    "action": "PROCESS_FORMULA",
    "documentInput": {
        "localCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "documentCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "itemInput": {
            "conditions": [{
                "stepNumber": 20,
                "counter": 1,
                "conditionType": "VA00",
                "conditionValue": 5.00,
                "variantCondition": true,
                "variantConditionKey": "2ndPATY",
                "inactiveFlag": " "
            }, {
                "stepNumber": 20,
```

```
                "counter": 2,
                "conditionType": "VA00",
                "conditionValue": 5.00,
                "variantCondition": true,
                "variantConditionKey": "BACON",
                "inactiveFlag": " "
            }, {
                "stepNumber": 30,
                "counter": 1,
                "conditionType": "ZVA4",
                "conditionValue": -0.50,
                "variantCondition": true,
                "variantConditionKey": "2ndPATY",
                "inactiveFlag": " "
            }]
        },
        "pricingCondition": {
            "stepNumber": 30,
            "counter": 2,
            "conditionType": "ZVA4",
            "variantConditionKey": "BACON"
        }
    }
}
```


Example 3: In this example, custom scale base formula 910 requests just the quantity details of the item and suppresses all pricing condition details.

Response SCL|910|COLLECT_ATTRIBUTES:


```
Content-Type: application/json; charset=utf-8
Content-Length: 189
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": [],
    "message": null,
    "item": null,
    "condition": null,
    "extendedInput": {
        "documentInput": {
            "itemInput": {
                "projection": ["quantity"],
                "conditions": null
            },
            "pricingCondition": {
                "projection": []
            }
        }
    }
}
```


1. itemInput->projection field is *["quantity"]*, that means formula requires quantity information of the item.
2. itemInput->conditions is *null* that means formula doesn't need any other conditions of the same item.
3. pricingCondition->projection is *[] i.e empty list* that means formula doesn't need any of the pricingCondition fields as input.

Based on the above extendedInput response, below is the payload generated when custom scale base formula 910 is called with action PROCESS_FORMULA.

Request SCL|910|PROCESS_FORMULA:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 285
Connection: keep-alive
cache-control: no-cache
{
    "formulaNumber": 910,
    "formulaType": "SCL",
    "action": "PROCESS_FORMULA",
    "documentInput": {
        "localCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "documentCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "itemInput": {
            "quantity": {
                "unit": "ST",
                "internalUnit": "ST",
                "value": 1.000
            }
        },
        "pricingCondition": {}
    }
}
```

Example 4: In this example, custom requirement formula 989 requests the required item attribute KOMK-KONDA and condition type information of the pricing condition to which formula is assigned.

Response REQ|989|COLLECT_ATTRIBUTES:

```
Content-Type: application/json; charset=utf-8
Content-Length: 218
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": ["KOMK-KONDA"],
    "message": null,
    "item": null,
    "condition": null,
    "extendedInput": {
        "documentInput": {
            "itemInput": {
                "projection": ["attributes"],
                "conditions": null
            },
            "pricingCondition": {
                "projection": ["conditionType"]
            }
        }
    }
}
```

1. itemInput->projection field is *["attributes"]*, that means formula requires the requested pricing attributes of the item. As result is *["KOMK-KONDA"]* only *KOMK-KONDA* pricing attribute is sent as the input.
2. itemInput->conditions is *null* that means formula does not require any other conditions of the same item as input.
3. pricingCondition->projection is *["conditionType"]* that means formula requires *conditionType* field of pricingCondition for which formula is executed.

Based on the above *extendedInput* response, below is the payload generated when custom requirement formula 989 is called with action PROCESS_FORMULA.

Request REQ|989|PROCESS_FORMULA:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 243
Connection: keep-alive
cache-control: no-cache
{
    "formulaNumber": 989,
    "formulaType": "REQ",
    "action": "PROCESS_FORMULA",
    "documentInput": {
        "localCurrency": null,
        "documentCurrency": null,
        "itemInput": {
            "attributes": [{
                "name": "KOMK-KONDA",
                "values": ["01"]
            }]
        },
        "pricingCondition": {
            "conditionType": "Z0K2"
        }
    }
}
```

Example 5: In this example, custom value formula 991 requests the subTotals and conditions of the item and no information from the pricing condition to which formula is assigned.

Response VAL|991|COLLECT_ATTRIBUTES:

```
Content-Type: application/json; charset=utf-8
Content-Length: 297
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": [],
    "message": null,
    "item": null,
    "condition": null,
    "extendedInput": {
        "documentInput": {
            "itemInput": {
                "projection": ["subTotals", "conditions"],
                "conditions": {
                    "filter": {
                        "conditionType": [null]
                    },
```

```
                    "projection": ["conditionType", "conditionValue",
"inactiveFlag"]
                }
            },
            "pricingCondition": {
                "projection": []
            }
        }
    }
}
```

1. itemInput->projection field is *["subTotals", "conditions"]*, that means formula requires subtotal flags and information about other conditions of the same item.
2. itemInput->conditions->filter->conditionType is *[null]*, that means formula requires only subtotal steps.
3. itemInput->conditions->projection is *["conditionType", "conditionValue", "inactiveFlag"]* that means formula require the mentioned condition fields. The *conditionType* field is not part of payload as it is null.
4. pricingCondition->projection is *[] i.e. empty list*, that means formula doesn't need any information of pricingCondition for which formula is executed.

Based on the above *extendedInput* response, below is the payload generated when custom value formula 991 is called with action PROCESS_FORMULA.

Request VAL|991|PROCESS_FORMULA:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 474
Connection: keep-alive
cache-control: no-cache
{
    "formulaNumber": 991,
    "formulaType": "VAL",
    "action": "PROCESS_FORMULA",
    "documentInput": {
        "localCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "documentCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "itemInput": {
            "subTotals": [{
                "flag": "1",
                "value": "0.75"
            }, {
                "flag": "2",
                "value": "7.95"
            }, {
                "flag": "3",
                "value": "-3.18"
            }],
            "conditions": [{
                "conditionValue": 0.75,
                "inactiveFlag": " "
            }, {
                "conditionValue": 7.95,
                "inactiveFlag": " "
            }, {
```

```
                "conditionValue": -3.18,
                "inactiveFlag": " "
            }]
        },
        "pricingCondition": {}
    }
}
```

Example 6: In this example, custom value formula 990 requests the conditionType and inactiveFlag of other condition Z0K2 of the same item and all the details for the pricing condition to which formula is assigned.

Response VAL|990|COLLECT_ATTRIBUTES:

```
Content-Type: application/json; charset=utf-8
Content-Length: 257
Connection: close
Access-Control-Allow-Origin: *
ETag: W/"20-FSSzTaUmOXUkw20qgOFxsNCExcg"
Strict-Transport-Security: max-age=15724800; includeSubDomains
{
    "result": [],
    "message": null,
    "item": null,
    "condition": null,
    "extendedInput": {
        "documentInput": {
            "itemInput": {
                "projection": ["conditions"],
                "conditions": {
                    "filter": {
                        "conditionType": ["Z0K2"]
                    },
                    "projection": ["conditionType", "inactiveFlag"]
                }
            },
            "pricingCondition": null
        }
    }
}
```

1. itemInput->projection field is *["conditions"]*, that means formula requires information about other conditions of the same item.
2. itemInput->conditions->filter->conditionType is *["Z0K2"]*, that means formula requires other conditions with condition type Z0K2 of the same item.
3. itemInput->conditions->projection is *["conditionType", "inactiveFlag"]* that means formula require the mentioned condition fields.
4. pricingCondition is *null* that means formula requires all the field of pricingCondition for which formula is executed.

Based on the above *extendedInput* response, below is the payload generated when custom value formula 990 is called with action PROCESS_FORMULA.

Request VAL|990|PROCESS_FORMULA:

```
Host: …
Content-Type: application/json
Accept: */*
API_KEY: …
Cache-Control: no-cache
accept-encoding: gzip, deflate
content-length: 763
Connection: keep-alive
cache-control: no-cache
```

```
{
    "formulaNumber": 990,
    "formulaType": "VAL",
    "action": "PROCESS_FORMULA",
    "documentInput": {
        "localCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "documentCurrency": {
            "numberOfDecimals": 2,
            "unit": "EUR"
        },
        "itemInput": {
            "conditions": [{
                "conditionType": "Z0K2",
                "inactiveFlag": " "
            }]
        },
        "pricingCondition": {
            "stepNumber": 80,
            "counter": 1,
            "conditionType": "K005",
            "calculationType": "C",
            "conditionBase": 1.000,
            "conditionRate": {
                "unit": "EUR",
                "internalUnit": "EUR",
                "value": -0.50
            },
            "conditionUnit": {
                "unit": "ST",
                "internalUnit": "ST",
                "value": 1.000
            },
            "conditionValue": -0.50,
            "conditionClass": "A",
            "structureCondition": " ",
            "purpose": null,
            "statistical": false,
            "variantCondition": false,
            "variantConditionFactor": 0,
            "variantConditionKey": null,
            "inactiveFlag": " ",
"recordId": "1057354822",
"origin": "A",
"scaleBaseType": "C"
        }
    }
}
```


5.2.4 Error Handling for the Pricing Service

The extension implementation for the pricing service reacts on HTTP error codes, timeouts, and other errors as described in the general error section. Errors in the extension calls are treated as if the extension is not implemented. The respective condition is marked as inactive, *inactiveFlag = X*, in the pricing result.

Any invalid response, for example not supported inactive flag, invalid subtotal flag will mark the condition as inactive with inactiveFlag = X.

Requirements REQ need some special consideration: if the requirement formula is not implemented, it is treated as if returning false. As the condition record is not determined there is no condition that can be marked as being inactive.

All errors and invalid responses are logged and can be viewed in the administration UI. If projection for *itemInput*, *conditions*, and *pricingCondition* contains a field which is not available, such fields are just ignored and no error/warning message is logged.

6 Appendix

The following sections include an example of the implementation and of the extension interface for each of the services.

6.1 Example Implementations

The following are example implementations of local and remote extensions.

6.1.1 Local Extension

Variant Function

File: ZCPS_VF_VARCONDS.js

```
function ZCPS_VF_VARCONDS(input) {
    var log = sap.log();
    var jsonInput = JSON.parse(input);
    var cps_drying_add_features, cps_varcond;

    try {
        cps_drying_add_features = jsonInput.vfunInput.fnArgs.find(({ id }) => id
=== 'CPS_DRYING_ADD_FEATURES');
        cps_varcond = jsonInput.vfunInput.fnArgs.find(({ id }) => id ===
'CPS_VARCOND');
    } catch (error) {
        // An exception can occur if no fnArgs has been provided
        log.error('Error: incomplete input data - no fnArgs provided');
        throw new Error('Incomplete input');
    }

    if (typeof cps_drying_add_features === 'undefined' ||
        typeof cps_varcond === 'undefined') {
        // Exception handling: unexpected fnArgs input
        log.error('Error: incomplete input data - missing (one of)
CPS_DRYING_ADD_FEATURES/CPS_VARCOND');
        throw new Error('Incomplete input');
    }

    // Copy the values of CPS_DRYING_ADD_FEATURES to CPS_VARCOND
    const features = cps_drying_add_features.values;
    var varconds = [...features];

    // Special case: if 'SIF' and 'LIP', add 'SIFLIP_DISCOUNT'
```

```
    if (features.includes('SIF') && features.includes('LIP')) {
        varconds.push('SIFLIP_DISCOUNT');
    }

    // Special case: if > 3 features, add 'MANY_FEATURE_DISCOUNT'
    if (features.length > 3) {
        varconds.push('MANY_FEATURE_DISCOUNT');
    }

    // Prepare and return
    cps_varcond.values = varconds;

    log.debug('Calculated VARCONDS' + cps_varcond.values);

    return JSON.stringify({ type: 'vfun', vfunOutput: { result: true, fnArgs:
[ cps_varcond ] } });
}
```

Variant Function with Database Access

File: Z_CUSTOMER_COLOUR.js

```
Z_CUSTOMER_COLOUR = function(input) {
  // obtain access to sap APIs
  var log = sap.log(); var db = sap.db();
  // get CPS_CUSTOMER and CPS_BACKREST_COLOUR from json input
  var jsonInput = JSON.parse(input);
  var cps_customer = jsonInput.vfunInput.fnArgs.find( ({ id }) => id ===
'CPS_CUSTOMER');
  var customerId = cps_customer.values[0];
  var cps_backrest_colour = jsonInput.vfunInput.fnArgs.find( ({ id }) => id ===
'CPS_BACKREST_COLOUR');
  // construct a select statement for the ZCUSTOMERCOLOUR table
  // it has the columns: [CLIENT, CUSTOMER, COLOUR]
  var selectStatement = db.select()
    .columns("COLOUR")
    .from("ZCUSTOMERCOLOUR")
    .where(
      db.eq("CUSTOMER", db.string(customerId))
    ).build();
  // execute it
  var dbResult = db.execute(selectStatement);
  // entry found for customer - return true
  if (dbResult.getRowCount() == 1) {
    var colour = dbResult.get(0, 0); // get row 0, column 0 - row and column
index begins at zero
    log.debug("found COLOUR: " + colour + " for CUSTOMER: " + customerId);
    if (typeof cps_backrest_colour === 'undefined') {
      return JSON.stringify({ type: 'vfun', vfunOutput: { result: true, fnArgs:
[ ] } });
    } else {
      cps_backrest_colour.values = [colour];
      return JSON.stringify({ type: 'vfun', vfunOutput: { result: true, fnArgs:
[ cps_backrest_colour ] } });
    }
  }
  // no entry found for customer - return false
  log.debug("no COLOUR found for CUSTOMER: " + customerId);
  return JSON.stringify({ type: 'vfun', vfunOutput: { result: false, fnArgs:
[] } });
}
```

REQ formula

File: req986.js

```
function REQ_986(pricingInput) {

    var json = JSON.parse(pricingInput);
    var response;
    if (json.action === 'COLLECT_ATTRIBUTES') {
        response = req_986_collect_attributes();
    }
    if (json.action === 'PROCESS_FORMULA') {
        response = req_986_process_formula(json);
    }
    return JSON.stringify(response);
}
function req_986_process_formula(json) {

    var requirementResult = false;
    var requirementMessage = "";
    var response;
    try {
        var land1 = getAttributeValue(json.documentInput.itemInput.attributes,
"KOMK-LAND1");
        var localCurrency =
getAttributeValue(json.documentInput.itemInput.attributes, "KOMK-WAERK");
        var documentCurrency =
getAttributeValue(json.documentInput.itemInput.attributes, "KOMK-HWAER");
        var cashDiscount =
getAttributeValue(json.documentInput.itemInput.attributes, "KOMP-SKTOF");
        response = { result: requirementResult, message: '' };
        if((land1 !== '') && (localCurrency == documentCurrency)) {
            if(cashDiscount === "X"){
                requirementResult = true;
                response = { result: requirementResult, message:
requirementMessage };
                return response;
            }
        }
        return response;
    }catch(e){
        console.log(e);
        return response;
    }
}
function req_986_collect_attributes(){
    var requirementMessage = "";
    var attributes = ["KOMK-LAND1","KOMK-WAERK","KOMK-HWAER","KOMP-SKTOF"];
    var response = { result: attributes, message: requirementMessage };
    return response;
}
```

The above example implementation uses the utility function “getAttributeValue”, which can be defined in a separate JavaScript file:

File: Utility.js

```
function getAttributeValue(attributes, name) {

    if(attributes === undefined){
        return '';
    }
    for (let i = 0; i < attributes.length ; i++){
        if((attributes[i].name) == name) {
            if(attributes[i].values !== undefined) {
```

```
                return attributes[i].values[0];
            }
            return '';
        }
    }
}
```

6.1.2 Remote Extensions

Variant Function

The following is an example of the implementation of a Variant Configuration extension in NodeJS. The example has been implemented via SAP BTP, Kyma Runtime. Please note that this is a simplified example with no logging or tracing, and no guarantee of functional or syntactical correctness.

The function *main* serves as the entry point for the extension implementations. It handles basic error handling, and forwards to the proper extension implementation based on the ID present in the input data. The function *cpsvarconds* is the actual implementation of our example variant function *ZCPS_VF_VARCONDS*. The function sets values for characteristic *CPS_VARCOND* based on the values of *CPS_DRYING_ADD_FEATURES*:

```
module.exports = {
    main: function (event, context) {

        if (event.data.type !== 'vfun') {
            // Error handling for wrong function type
            event.extensions.response.status(400);
            return 'Error: only functions of type \'vfun\' are supported';
        }

        var vfunInput = event.data.vfunInput;

        if (typeof vfunInput === 'undefined') {
            // Error handling for missing variant function input
            event.extensions.response.status(400);
            return 'Error: variant function input not provided';
        }
        // Response of the extension implementation call
        var functionResponse;
        switch (vfunInput.id) {
            case "ZCPS_VF_VARCONDS":
                functionResponse = cpsvarconds(event.data);
                event.extensions.response.status(functionResponse.statusCode);
                return functionResponse.data;
            default:
                if ( typeof vfunInput.id === 'undefined') {
                    event.extensions.response.status(400);
                    return 'Error: variant function ID not specified';
```

```
                } else {
                    event.extensions.response.status(404);
                    return 'Error: variant function ID ' + vfunInput.id + ' not
found';
                }
        }
    }
}
function cpsvarconds(data) {
      var cps_drying_add_features, cps_varcond;
      try {
          cps_drying_add_features = data.vfunInput.fnArgs.find(({ id }) => id
=== 'CPS_DRYING_ADD_FEATURES');
          cps_varcond = data.vfunInput.fnArgs.find(({ id }) => id ===
'CPS_VARCOND');
      } catch (error) {
          // An exception can occur if no fnArgs has been provided
          return { statusCode: 400, data: "Error: incomplete input data - no
fnArgs provided" };
      }

      if (typeof cps_drying_add_features === 'undefined' ||
          typeof cps_varcond === 'undefined') {
          return { statusCode: 400, data: "Error: incomplete input data -
missing (one of) CPS_DRYING_ADD_FEATURES/CPS_VARCOND" };
      }
      const features = cps_drying_add_features.values;
      var varconds = [...features];

      // Special case: if 'SIF' and 'LIP', add 'SIFLIP_DISCOUNT'
      if (features.includes('SIF') && features.includes('LIP')) {
          varconds.push('SIFLIP_DISCOUNT');
      }

      // Special case: if > 3 features, add 'MANY_FEATURES_DISCOUNT'
      if (features.length > 3) {
          varconds.push(' MANY_FEATURES_DISCOUNT');
      }

      // Prepare and return
      cps_varcond.values = varconds;
      return { statusCode: 200, data: { type: 'vfun', vfunOutput: { result:
true, fnArgs: [ cps_varcond ] } } };
}
```

REQ formula

The following is an example of the implementation of a REQ formula extension in NodeJS. The example has been implemented via SAP BTP, Kyma Runtime. Please note that this is a simplified example with no logging or tracing, and no guarantee of functional or syntactical correctness.

The function *main* serves as the entry point for the extension implementations. It handles basic error handling, and forwards to the proper extension implementation based on the ID present in the input data.

```
module.exports = {
    main: function (event) {
        var errorMessage;
        var functionResponse;
        switch(event.data.formulaType){
            case "REQ":
                functionResponse = req(event.data);
                event.extensions.response.status(functionResponse.statusCode);
```

```
                return functionResponse.data;
                // If formula type is not implemented
            default:
                event.extensions.response.status(501);
                errorMessage = 'Error: User Exit type ' + event.data.formulaType
+ ' not found';
                return { result: '', message: errorMessage };
        }
    }
 }
 function req(data){
    var functionResponse;
    var result;
    var errorMessage;
        switch(data.formulaNumber) {
            case 986:
                functionResponse = req986(data);
                return { statusCode: 200, data: functionResponse };

               // If formula number is not implemented
            default:
                errorMessage = 'Error: UserExit number ' + data.formulaNumber +
' of type ' + data.formulaType + ' not implemented' ;
                result = { result: '', message: errorMessage };
                response =  { statusCode: 501, data: result };
        }
        return response;
 }
 function req986(data){
        var response;
        switch(data.action){
            case 'COLLECT_ATTRIBUTES':
                var requirementMessage = "";
                var attributes = ["KOMK-LAND1","KOMK-WAERK","KOMK-HWAER","KOMP-
SKTOF"];
                response = { result: attributes, message: requirementMessage };
                return response;
            case 'PROCESS_FORMULA':
                var requirementResult = false;
                var requirementMessage = "";
                try {
                    var land1 =
getAttributeValue(data.documentInput.itemInput.attributes, "KOMK-LAND1");
                    var localCurrency =
getAttributeValue(data.documentInput.itemInput.attributes, "KOMK-WAERK");
                    var documentCurrency =
getAttributeValue(data.documentInput.itemInput.attributes, "KOMK-HWAER");
                    var cashDiscount =
getAttributeValue(data.documentInput.itemInput.attributes, "KOMP-SKTOF");
                    response = { result: requirementResult, message: '' };
                    if((land1 !== '') && (localCurrency == documentCurrency)) {
                        if(cashDiscount === "X"){
                            requirementResult = true;
                            response = { result: requirementResult, message:
requirementMessage };
                            return response;
                        }
                    }
                    return response;
                } catch(e){
                    return response;
                }
        }
        return response;
  }
// Helper function to get item attributes value
 function  getAttributeValue (attributes, name){
    if(attributes === undefined){
```

```
            return '';
        }

       for (i = 0; i < attributes.length ; i++){
           if((attributes[i].name) == name)
           {
                if(attributes[i].values !== undefined){
                     return attributes[i].values[0];
                }
                return '';
           }
       }
}
```

In the above code snippet, function *req986* handles both *process formula* and *collect attributes* for the requirement formula 986.

6.2 Extension Interface

This section presents the user extension interfaces for the Variant Configuration and Pricing services in the OpenAPI format. The documentation can be imported and viewed with any OpenAPI-compatible editor or API designer, such as the SAP API Designer on SAP Business Technology Platform.

6.2.1 Variant Configuration Extension Interface

Copy into the Swagger editor for better readability. The examples refer to the *CPS_BURGER* which is also used in the *api.sap.com/api/ProductConfigurationService/resource* examples.

```
openapi: 3.0.1
info:
  title: Variant Configuration service
  description: "Use our variant configuration APIs to build applications that
enable your sales teams, customers, and channel partners to configure and price
your products and services accurately and efficiently.\r\n\r\n"
  version: v2
servers:
  - url: https://localhost
    description: Generated server url
tags:
  - description: Variant function call stub
    name: Variant function call
paths:
  /api/v2/cfguserextension:
    post:
      operationId: createConfiguration
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ext.ConfigurationUserExitInput'
        required: true
      responses:
        default:
```

```
          content:
            '*/*':
              schema:
                $ref: '#/components/schemas/ext.ConfigurationUserExitOutput'
          description: default response
      summary: Call extension
      tags:
        - Variant function call
components:
  schemas:
    ext.ConfigurationUserExitInput:
      type: object
      properties:
        type:
          type: string
          description: Identifier of the extension type. Currently only variant
functions (type = 'vfun') are supported.
          example: vfun
        vfunInput:
          $ref: '#/components/schemas/ext.VariantFunctionInput'
      description: Represents the input data structure (request body) of a
generic Variant Configuration user exit.
    ext.ConfigurationUserExitOutput:
      type: object
      properties:
        type:
          type: string
          description: Identifier of the user exit type
        vfunOutput:
          $ref: '#/components/schemas/ext.VariantFunctionOutput'
      description: Represents the output data structure (response) of a generic
CFG user exit.
    ext.KBHeaderInfo:
      type: object
      properties:
        bomApplication:
          type: string
          description: 'The application that was used during knowledge-base
generation to determine the relevant bill of materials by its appropriate BOM
Usage and Alternative BOM. E.g. SD01 for sales BOMs. Note: bomApplication is
only part of the data model in most recent back-end releases with implemented
BAdI KBSCHEMA_CURT.'
          example: SD01
        build:
          type: integer
          description: The knowledge-base build number.
          format: int32
          example: 9
        changeDate:
          type: string
          description: Date of the last knowledge-base's change in ISO8601. This
is set to knowledge-base creation date for new versions.
          example: '2018-10-04'
        id:
          type: integer
          description: Identifier of the knowledge base
          format: int32
          example: 80
        key:
          $ref: '#/components/schemas/ext.KnowledgebaseKey'
        plant:
          type: string
          description: 'The plant that was used during knowledge-base generation
to determine the relevant bill of materials. Note: Plant is only part of the
data model in most recent back-end releases with implemented BAdI KBSCHEMA_CURT.'
          example: '1000'
        structureHash:
          type: string
```

```
          description: Hash value of static product structure data contained in
the knowledge base. It changes if products, classes, characteristics, values,
bill of materials are updated.
          example: CF897878C9EBDFBB379EC107602BB490
        type:
          type: string
          description: "The type of the knowledge base indicates both the origin
and the use of the knowledge-base runtime version. The following values are
possible: variantConfiguration, solutionConfiguration.\r\nConfigurations that
refer to knowledge bases of type variantConfiguration are processed with
the Variant Configuration service.\r\nConfigurations that refer to knowledge
bases of type solutionConfiguration must be processed with SAP Solution Sales
Configuration."
        validFromDate:
          type: string
          description: Start date of the knowledge-base's validity in ISO8601
          example: '2018-05-02'
      description: Represents the header data of a knowledge base.
    ext.KnowledgebaseKey:
      type: object
      properties:
        kbName:
          type: string
          description: Language neutral identifier of the knowledgebase.
          example: CPS_BURGER
        kbVersion:
          type: string
          description: Version of the knowledgebase.
          example: '0.1'
        logsys:
          type: string
          description: Logical system of origin of the knowledgebase.
          example: RR4CLNT910
      description: Represents the key structure for external identification of a
knowledgebase.
    ext.VariantFunctionArgument:
      type: object
      properties:
        id:
          type: string
          description: Identifier of the parameter
          example: CPS_SIZE
        type:
          type: string
          description: Data type of the parameter, one of string, float or date
          example: string
        values:
          type: array
          description: List of stringified bindings of the argument in the
internal (language independent) format.
          example: '''[L]'''
          items:
            type: string
            description: List of stringified bindings of the argument in the
internal (language independent) format.
            example: '''[L]'''
      description: Represents an input or output argument of the function call
and its value binding(s).
      example:
        - id: CPS_FRY_SAUCE
          type: String
          values:
            - K
    ext.VariantFunctionInput:
      type: object
      properties:
        fnArgs:
          type: array
```

```
          description: List of function arguments. The Input arguments are
distinguished from the Output arguments by their value bindings. Typically,
a single value is provided for each input argument. Output argument's value
bindings are left empty in the request.
          example:
            - id: CPS_GARNISH_INV
              type: String
              values:
                - F
            - id: CPS_SIZE
              type: String
              values:
                - L
            - id: CPS_FRY_SAUCE
              type: String
              values: []
          items:
            $ref: '#/components/schemas/ext.VariantFunctionArgument'
        id:
          type: string
          description: Identifier of Variant function (language neutral name)
        kbHeaderInfo:
          $ref: '#/components/schemas/ext.KBHeaderInfo'
      description: Represents the input data structure (request) of a variant
function call.
    ext.VariantFunctionOutput:
      type: object
      properties:
        fnArgs:
          type: array
          description: List of variant function output arguments. The Output
argument’s value bindings are set in the variant function implementation.
Typically, a single value binding is provided for each output argument. If the
variant function is called from a procedure for a multivalue characteristic, the
output value binding can be a comma separated list of values.
          example:
            - id: CPS_FRY_SAUCE
              type: String
              values:
                - K
          items:
            $ref: '#/components/schemas/ext.VariantFunctionArgument'
        result:
          type: boolean
          description: Result of the variant function, true or false.
          example: true
      description: Represents the output data structure (response) of a variant
function call.
```

6.2.2 Pricing Extension Interface

Copy into the Swagger editor for better readability.

```
openapi: 3.0.1
info:
  description: Use our pricing APIs in your applications to calculate prices
based on data maintained in SAP ERP or S/4HANA.
  title: Pricing service
  version: v1
servers:
  - url: http://localhost:8080
    description: Generated server url
```

```
tags:
  - description: API as called for custom formula implementations.
    name: Custom formula controller
paths:
  /api/rv1/customformula:
    post:
      description: 'Pricing service calls the customer web service for all
custom pricing routines used in a pricing procedure. The Pricing service
provides the input data and expects the output data including http-codes as
described here. '
      operationId: callCustomFormula
      parameters:
        - description: Endpoint where the custom formula implementation exits.
          in: query
          name: url
          required: true
          schema:
            type: string
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ext.input'
        required: true
      responses:
        '200':
          content:
            '*/*':
              schema:
                $ref: '#/components/schemas/ext.PricingOutput'
          description: Custom formula executed successfully.
        '401':
          content:
            '*/*':
              schema:
                type: string
          description: Unauthorized. Authentication for the user failed.
        '403':
          content:
            '*/*':
              schema:
                type: string
          description: Forbidden. The user is not authorized to call custom
formula implementation.
        '404':
          content:
            '*/*':
              schema:
                type: string
          description: Web service URL for custom formula implementation not
found.
        '500':
          content:
            '*/*':
              schema:
                type: string
          description: A server-side exception occurred during function
processing.
        '501':
          content:
            '*/*':
              schema:
                type: string
          description: Custom formula not implemented.
        '503':
          content:
            '*/*':
              schema:
```

```
                type: string
          description: 'The service is not available, please try again later.
Possible reasons: the tenant provisioning has not been completed yet, or the
service is down because of maintenance activities, or the contract ended.'
      summary: Documentation of the API for the web services that implement
custom pricing routines.
      tags:
        - Custom formula controller
components:
  schemas:
    ext.Attribute:
      required:
        - values
      type: object
      properties:
        name:
          type: string
          description: Attribute name in SAP internal format
        values:
          type: array
          description: Attribute values in SAP internal format
          items:
            type: string
            description: Attribute values in SAP internal format
      description: Pricing attributes data for custom formula
    ext.ConditionOutput:
      type: object
      properties:
        conditionRate:
          $ref: '#/components/schemas/ext.ConditionRate'
        inactiveFlag:
          type: string
          description: |-
            Status of the condition.The value of the inactive flag in the
pricing condition will be overwritten with the value passed.
             The inactive flag which can be used are: <table><tr><td>Inactive
Status</td><td>Description</td></tr><tr><td>X</td><td>Inactive via formula or
Inactive due to error</td></tr><tr><td>Z</td><td>Invisible</td></tr></table>
        statistical:
          type: boolean
          description: Whether the condition is a statistical condition.
Statistical conditions are not considered in net price calculations.The value of
the statistical flag in the pricing condition will be overwritten with the value
passed.
      description: Pricing condition output data for custom formula. Processed
only for formula types VAL and BAS when called with action PROCESS_FORMULA.
    ext.ConditionRate:
      type: object
      properties:
        value:
          type: number
          description: Condition rate value
      description: Amount or percentage depending on the condition type. The
condition rate of the pricing condition will be overwritten with the value
passed.
    ext.ConditionsFilter:
      type: object
      properties:
        conditionType:
          type: array
          description: "Contains the list of condition type names required for
the formula.In pricing, the pricing elements are depicted using condition types.
Pricing elements can be prices, surcharges, or discounts. Add 'null' in case
formula requires subtotal steps, too.\r\n\nFor example: [ \"PR00\", \"ZVA1\",
null ]"
          items:
            type: string
```

```
            description: "Contains the list of condition type names required for
the formula.In pricing, the pricing elements are depicted using condition types.
Pricing elements can be prices, surcharges, or discounts. Add 'null' in case
formula requires subtotal steps, too.\r\n\nFor example: [ \"PR00\", \"ZVA1\",
null ]"
      description: Request additional item conditions as required by custom user
exitformula.
    ext.CurrencyUnit:
      type: object
      properties:
        numberOfDecimals:
          type: integer
          description: Number of decimal places for currency unit (e.g. 2)
          format: int32
        unit:
          type: string
          description: ISO code for currency unit (e.g. USD)
      description: Currency unit
    ext.DocumentInput:
      type: object
      properties:
        documentCurrency:
          $ref: '#/components/schemas/ext.CurrencyUnit'
        itemInput:
          $ref: '#/components/schemas/ext.ItemInput'
        localCurrency:
          $ref: '#/components/schemas/ext.CurrencyUnit'
        pricingCondition:
          $ref: '#/components/schemas/ext.PricingCondition'
      description: Pricing document input data for custom formula
    ext.ExtendedCondition:
      type: object
      properties:
        filter:
          $ref: '#/components/schemas/ext.ConditionsFilter'
        projection:
          type: array
          description: "List that holds the condition fields in which
the formula is interested. In case 'projection' is an empty array, none
of the condition fields will be passed and in case of 'null' value
or not provided, all the available condition fields will be passed
when calling the formula with action PROCESS_FORMULA.\r\nBelow are the
listed fields which are allowed as projection for a pricing condition.
Projection name matches with the field name and is case-sensitive.\r\n<table
border= \"solid\"><tr><td>stepNumber</td></tr><tr><td>counter</
td></tr><tr><td>conditionType</td></tr><tr><td>calculationType</
td></tr><tr><td>conditionBase</td></tr><tr><td>conditionRate</td></
tr><tr><td>conditionUnit</td></tr><tr><td>conditionValue</td></
tr><tr><td>conditionClass</td></tr><tr><td>structureCondition</
td></tr><tr><td>purpose</td></tr><tr><td>statistical</td></
tr><tr><td>variantCondition</td></tr><tr><td>variantConditionFactor</
td></tr><tr><td>variantConditionKey</td></tr><tr><td>inactiveFlag</td></
tr><tr><td>durationFactor</td></tr><tr><td>recordId</td></tr><tr><td>origin</
td></tr><tr><td>scaleBaseType</td></tr></table>"
          items:
            type: string
            description: "List that holds the condition fields in which
the formula is interested. In case 'projection' is an empty array, none
of the condition fields will be passed and in case of 'null' value
or not provided, all the available condition fields will be passed
when calling the formula with action PROCESS_FORMULA.\r\nBelow are the
listed fields which are allowed as projection for a pricing condition.
Projection name matches with the field name and is case-sensitive.\r\n<table
border= \"solid\"><tr><td>stepNumber</td></tr><tr><td>counter</
td></tr><tr><td>conditionType</td></tr><tr><td>calculationType</
td></tr><tr><td>conditionBase</td></tr><tr><td>conditionRate</td></
tr><tr><td>conditionUnit</td></tr><tr><td>conditionValue</td></
tr><tr><td>conditionClass</td></tr><tr><td>structureCondition</
```

td></tr><tr><td>purpose</td></tr><tr><td>statistical</td></
tr><tr><td>variantCondition</td></tr><tr><td>variantConditionFactor</
td></tr><tr><td>variantConditionKey</td></tr><tr><td>inactiveFlag</td></
tr><tr><td>durationFactor</td></tr><tr><td>recordId</td></tr><tr><td>origin</
td></tr><tr><td>scaleBaseType</td></tr></table>"
      description: 'Allows filtering of the conditions and projection on
required condition fields which user exit extension requires. '
    ext.ExtendedDocument:
      type: object
      properties:
        itemInput:
          $ref: '#/components/schemas/ext.ExtendedItem'
        pricingCondition:
          $ref: '#/components/schemas/ext.ExtendedPricingConditionInput'
      description: Allows to specify the needed item and condition data via so-
called projections.
    ext.ExtendedInput:
      type: object
      properties:
        documentInput:
          $ref: '#/components/schemas/ext.ExtendedDocument'
      description: Allows custom formula to tell the pricing engine which item
fields, conditions, and fields of a condition are required during formula
execution. This information can only be provided when formula is called with
action COLLECT_ATTRIBUTES. It is recommended to reduce the amount of requested
data to the absolute minimum.
    ext.ExtendedItem:
      type: object
      properties:
        conditions:
          $ref: '#/components/schemas/ext.ExtendedCondition'
        projection:
          type: array
          description: "List that holds the item fields in which the
formula is interested. In case 'projection' is an empty array, none of
the item fields will be passed and in case of 'null' value or not
provided, all the available item fields will be passed when calling the
formula with action PROCESS_FORMULA.\r\n\nBelow are the listed fields which
are allowed as projection for an item. Projection name matches with the
field name and is case-sensitive.<table border= \"solid\"><tr><td>quantity</
tr><tr><td>netValue</td></tr><tr><td>netPrice</td></tr><tr><td>taxValue</td></
tr><tr><td>volume</td></tr><tr><td>grossWeight</td></tr><tr><td>netWeight</td></
tr><tr><td>subTotals</td></tr><tr><td>attributes</td></tr><tr><td>statistical</
td></tr><tr><td>lastPriceCondition</td></tr><tr><td>exclusionIndicator</td></
tr><tr><td>conditions</td></tr></table>\nSpecify 'attributes' here as
projection, to ensure that only the subset of attributes requested
during the COLLECT_ATTRIBUTES call will be sent as input during
PROCESS_FORMULA call.\r\n\nThe additional item conditions can only be
passed to the formula if 'conditions'->'filter' and 'conditions'->'projection'
were specified. Only those conditions of the same item will be considered
that match the filter criteria and have a step number less than or equal
to the current pricing step for which the custom formula is being executed."
          items:
            type: string
            description: "List that holds the item fields in which the
formula is interested. In case 'projection' is an empty array, none of
the item fields will be passed and in case of 'null' value or not
provided, all the available item fields will be passed when calling the
formula with action PROCESS_FORMULA.\r\n\nBelow are the listed fields which
are allowed as projection for an item. Projection name matches with the
field name and is case-sensitive.<table border= \"solid\"><tr><td>quantity</
tr><tr><td>netValue</td></tr><tr><td>netPrice</td></tr><tr><td>taxValue</td></
tr><tr><td>volume</td></tr><tr><td>grossWeight</td></tr><tr><td>netWeight</td></
tr><tr><td>subTotals</td></tr><tr><td>attributes</td></tr><tr><td>statistical</
td></tr><tr><td>lastPriceCondition</td></tr><tr><td>exclusionIndicator</td></
tr><tr><td>conditions</td></tr></table>\nSpecify 'attributes' here as
projection, to ensure that only the subset of attributes requested
during the COLLECT_ATTRIBUTES call will be sent as input during

```
PROCESS_FORMULA call.\r\n\nThe additional item conditions can only be
passed to the formula if 'conditions'->'filter' and 'conditions'->'projection'
were specified. Only those conditions of the same item will be considered
that match the filter criteria and have a step number less than or equal
to the current pricing step for which the custom formula is being executed."
      description: 'Allows to specify which item fields are really needed by
the custom formula via a so-called projection. Furthermore, it allows to specify
via a filter which additional condition types are needed as input and via a
further projection what exact data is required from conditions.Note: Please be
aware that requesting more data here has negative impact on performance as for
example the likelihood of cache hits decreases. '
    ext.ExtendedPricingCondition:
      type: object
      properties:
        calculationType:
          type: string
          description: This field specifies how the pricing service calculates
the condition value. The pricing service can calculate a condition value
in different ways, for example in its absolute amount, as a percentage
etc. <table  border= "solid"><tr><td>Calculation Type</td><td>Description</
td></tr><tr><td>A</td><td>Percentage</td></tr><tr><td>B</td><td>Fixed Amount</
td></tr><tr><td>C</td><td>Quantity</td></tr><tr><td>D</td><td>Gross weight</
td></tr><tr><td>E</td><td>Net weight</td></tr><tr><td>F</td><td>Volume</td></
tr><tr><td>G</td><td>Formula</td></tr><tr><td>H</td><td>Percentage (in
hundreds)</td></tr><tr><td>I</td><td>Percentage (travel expenses)</td></
tr><tr><td>J</td><td>Per mile</td></tr><tr><td>K</td><td>Per mile (in
thousands)</td></tr><tr><td>L</td><td>Points</td></tr><tr><td>M</td><td>Quantity
- monthly price</td></tr><tr><td>N</td><td>Quantity - yearly price</td></
tr><tr><td>O</td><td>Quantity - daily price</td></tr><tr><td>P</td><td>Quantity
- weekly price</td></tr><tr><td>Q</td><td>Commodity Formula</td></tr><tr><td>U</
td><td>Percentage FIN</td></tr><tr><td>W</td><td>Percentage (with 6 decimal
places)</td></tr></table>
          deprecated: true
        conditionBase:
          type: number
          description: 'Condition base. The unit of condition base depends
on the calculation type of the condition. Refer to below table: <table
border= "solid"><tr><td>Calculation Type</td><td>Condition Base: Unit</td></
tr><tr><td>A, B, H, U, I, W</td><td> Document Currency </td></tr><tr><td>G, Q</
td><td>Unit of measure of last pricing condition base</td></tr><tr><td>Other
types</td><td>Unit of measure of the condition record</td></tr></table>'
        conditionClass:
          type: string
          description: Preliminary structuring of condition types
e.g. in surcharges and discounts or prices. Allows standardized
processing of individual condition classes within the system. <table
border= "solid"><tr><td>Condition Class</td><td>Description</td></tr><tr><td>A</
td><td>Discount or surcharge</td></tr><tr><td>B</td><td>Prices</td></
tr><tr><td>C</td><td>Expense reimbursement</td></tr><tr><td>D</td><td>Taxes</
td></tr><tr><td>E</td><td>Extra pay</td></tr><tr><td>F</td><td>Fees for
differential</td></tr><tr><td>G</td><td>Tax classification</td></tr><tr><td>H</
td><td>Determining sales deal</td></tr><tr><td>Q</td><td>Totals record for fees
only</td></tr><tr><td>W</td><td>Wage withholding tax</td></tr></table>
          deprecated: true
        conditionRate:
          $ref: '#/components/schemas/ext.UnitValue'
        conditionType:
          type: string
          description: In pricing, the pricing elements are depicted using
condition types. Pricing elements can be prices, surcharges, or discounts.
        conditionUnit:
          $ref: '#/components/schemas/ext.UnitValue'
        conditionValue:
          type: number
          description: The value resulting from pricing, for a condition, total,
or sub-total in a document
        counter:
          type: integer
```

```
          description: counter
          format: int32
          deprecated: true
        durationFactor:
          type: number
          description: 'The duration factor corresponds to the condition''s
calculation type: M-months, N-years, O-days, P-weeks. Duration factor is applied
to the product of condition rate and quantity to calculate the condition value
for the given contract duration.'
        inactiveFlag:
          type: string
          description: |-
            Status of the condition.
             For example: Condition status will be made inactive if there is any
error during currency conversion
             The reasons for being inactive are: <table
border= "solid"><tr><td>Inactive Status</td><td>Description</td></tr><tr><td>A</
td><td>Condition exclusion item</td></tr><tr><td>K</td><td>Inactive due to
calculation basis</td></tr><tr><td>M</td><td>Inactive due to manual entry</td></
tr><tr><tr><td>W</td><td>The document item is statistical</td></tr><tr><td>X</
td><td>Inactive via formula or Inactive due to error</td></tr><tr><td>Y</
td><td>Inactive because of subsequent price</td></tr><td>Z</td><td>Invisible</
td></tr><tr><td>\' \'(space)</q></td><td>Condition is active</td></tr></table>
          deprecated: true
        origin:
          type: string
          description: |-
            Origin of the condition
             Below are the listed origins with description: <br> <table
border= "solid"><tr><th>Origin</th><th>Description</th></tr><tr><td>" "(space)</
td><td>Subtotal step at item level</td></tr><tr><td>A</td><td>Automatic
pricing</td></tr><tr><td>B</td><td>Duplicated from main item</td></tr><tr><td>C</
td><td>Manually entered/changed</td></tr><tr><td>D</td><td>Header condition</
td></tr><tr><td>E</td><td>Item total</td></tr><tr><td>F</td><td>Condition
supplement</td></tr><tr><td>G</td><td>Original header condition</td></
tr><tr><td>H</td><td>Correction rebate</td></tr><tr><td>I</td><td>Cost
correction</td></tr><tr><td>J</td><td>Transaction tax engine</td></tr></table>
        purpose:
          type: string
          description: Name assigned in customizing to a condition record in
field conditionFunction. Used to aggregate conditions with same purpose, e.g.
all types of variant conditions.
        recordId:
          type: string
          description: Number that uniquely identifies a condition record.
        scaleBaseType:
          type: string
          description: Determines how the system interprets a pricing scale
in a condition. For example, the scale can be based on quantity, weight,
or volume. <table  border= "solid"><tr><td>Scale base type</td><td>Description</
td></tr><tr><td>" "(space) or "" (blank)</td><td>No scales</td></tr><tr><td>B</
td><td>Value scale</td></tr><tr><td>C</td><td>Quantity scale</td></tr><tr><td>D</
td><td>Gross weight scale</td></tr><tr><td>E</td><td>net weight scale</td></
tr><tr><td>F</td><td>Volume scale</td></tr><tr><td>G</td><td>Scale based on a
formula</td></tr><tr><td>L</td><td>Point scale</td></tr><tr><td>M</td><td>Time
period scale - Month</td></tr><tr><td>N</td><td>Time period scale - Year</td></
tr><tr><td>O</td><td>Time period scale - Day</td></tr><tr><td>P</td><td>Time
period scale - Week</td></tr><tr><td>R</td><td>Distance</td></tr><tr><td>S</
td><td>Number of shipping units</td></tr></table>
        statistical:
          type: boolean
          description: Whether the condition is a statistical condition.
Statistical conditions are not considered in net price calculations.
        stepNumber:
          type: integer
          description: Number that determines the sequence of the condition
types within a procedure.
          format: int32
```

```
          deprecated: true
        structureCondition:
          type: string
          description: Structure Condition. <table  border=
"solid"><tr><td>Structure Condition</td><td>Description</td></tr><tr><td>A</
td><td>Condition to be duplicated. Condition determination happens at the
main item and same condition is duplicated for sub items.</td></tr><tr><td>B</
td><td>Cumulation condition. Cumulate the sub item price and display it at
the main item level.</td></tr><tr><td>' ' (space)</td><td> Not a structure
condition. </td></tr></table>
          deprecated: true
        variantCondition:
          type: boolean
          description: Whether the condition is a variant condition. Variant
conditions are available for configurable products.
          deprecated: true
        variantConditionFactor:
          type: number
          description: A multiplier originating from Variant Configuration
service that applies to the condition value.
        variantConditionKey:
          type: string
          description: Technical identifier for the variant condition
      description: Pricing condition input data for custom formula
    ext.ExtendedPricingConditionInput:
      type: object
      properties:
        projection:
          type: array
          description: "List that holds the pricing condition fields in
which the formula is interested. In case 'projection' is an empty array,
none of the pricing condition fields will be passed and in case of 'null'
value or not provided, all the available pricing condition fields will
be passed when calling the formula with action PROCESS_FORMULA.\r\n\n\nBelow
are the listed fields which are allowed as projection for a pricing
condition. Projection name matches with the field name and is case-
sensitive.<table border= \"solid\"><tr><td>stepNumber</td></tr><tr><td>counter</
td></tr><tr><td>conditionType</td></tr><tr><td>calculationType</
td></tr><tr><td>conditionBase</td></tr><tr><td>conditionRate</td></
tr><tr><td>conditionUnit</td></tr><tr><td>conditionValue</td></
tr><tr><td>conditionClass</td></tr><tr><td>structureCondition</
td></tr><tr><td>purpose</td></tr><tr><td>statistical</td></
tr><tr><td>variantCondition</td></tr><tr><td>variantConditionFactor</
td></tr><tr><td>variantConditionKey</td></tr><tr><td>inactiveFlag</td></
tr><tr><td>durationFactor</td></tr><tr><td>recordId</td></tr><tr><td>origin</
td></tr><tr><td>scaleBaseType</td></tr></table>"
          items:
            type: string
            description: "List that holds the pricing condition fields in
which the formula is interested. In case 'projection' is an empty array,
none of the pricing condition fields will be passed and in case of 'null'
value or not provided, all the available pricing condition fields will
be passed when calling the formula with action PROCESS_FORMULA.\r\n\n\nBelow
are the listed fields which are allowed as projection for a pricing
condition. Projection name matches with the field name and is case-
sensitive.<table border= \"solid\"><tr><td>stepNumber</td></tr><tr><td>counter</
td></tr><tr><td>conditionType</td></tr><tr><td>calculationType</
td></tr><tr><td>conditionBase</td></tr><tr><td>conditionRate</td></
tr><tr><td>conditionUnit</td></tr><tr><td>conditionValue</td></
tr><tr><td>conditionClass</td></tr><tr><td>structureCondition</
td></tr><tr><td>purpose</td></tr><tr><td>statistical</td></
tr><tr><td>variantCondition</td></tr><tr><td>variantConditionFactor</
td></tr><tr><td>variantConditionKey</td></tr><tr><td>inactiveFlag</td></
tr><tr><td>durationFactor</td></tr><tr><td>recordId</td></tr><tr><td>origin</
td></tr><tr><td>scaleBaseType</td></tr></table>"
      description: Pricing condition to which the formula was assigned. Allows
to specify which pricing condition fields are really needed by the custom
formula via a so-called projection.
```

```
    ext.ItemInput:
      type: object
      properties:
        attributes:
          type: array
          description: "List of pricing attributes as name-value pairs.
Supported attribute names for a pricing procedure are provided by API /api/v1/
pricingprocedure/{pricingProcedureId} and these attributes can also be used
in custom formulas. Additional pricing attributes needed by a custom
formula can be added by implementing action COLLECT_ATTRIBUTES. \r\nFor
example: [{ \"name\": \"KOMK-HWAER\", \"values\": [ \"EUR\" ] }, { \"name\":
\"KOMK-KUNNR\", \"values\": [ \"T-L63A01\" ] }]\r\nHere, KOMK-HWAER is Local
currency and KOMK-KUNNR is sold-toparty."
          items:
            $ref: '#/components/schemas/ext.Attribute'
        conditions:
          type: array
          description: List of item conditions. List is 'null' by default.
If custom formula requires details of other conditions of the same item
than the one to which the formula was assigned or the active price
condition, then those condition details can be provided by the pricing
engine here. For that, the formula implementation must tell the engine via
action COLLECT_ATTRIBUTES in response object 'extendedInput'->'documentInput'-
>'itemInput'->'conditions'->'filter' which condition types are relevant and via
'extendedInput'->'documentInput'->'itemInput'->'conditions'->'projection' which
condition input data is relevant for the formula processing.
          items:
            $ref: '#/components/schemas/ext.ExtendedPricingCondition'
        exclusionIndicator:
          type: string
          description: |-
            Condition exclusion indicator flag that was set for the item in a
preceding extension call. Depending on the value set for the indicator by a
preceding extension call, subsequent decisions can be made.
            For example: For a particular discount condition you can specify a
rule that applies the discount only when the exclusion indicator is set.
          example: $
        grossWeight:
          $ref: '#/components/schemas/ext.UnitValue'
        lastPriceCondition:
          $ref: '#/components/schemas/ext.LastPriceCondition'
        netPrice:
          type: number
          description: |-
            Net price of pricing item.
             For example: If the price of product is 30.00 EUR per 2 PCE,
then this field holds the 30.00 which is derived in document currency (in this
example it is EUR)
        netValue:
          type: number
          description: The net value of pricing item, after any discounts and
surcharges are considered. Sales taxes are not included. The value is expressed
in the document currency.
        netWeight:
          $ref: '#/components/schemas/ext.UnitValue'
        quantity:
          $ref: '#/components/schemas/ext.UnitValue'
        statistical:
          type: boolean
          description: Whether item is statistical or not. Statistial items are
not considered in document level calculation.
        subTotals:
          type: array
          description: |-
            Subtotals of pricing item.
            For example: [{ "flag": "K", "value": 120.00 ] }, { "flag": "1",
"value": 201.32 }] .
          items:
```

```
            $ref: '#/components/schemas/ext.Subtotal'
        taxValue:
          type: number
          description: Tax amount of pricing item (e.g. 256.46). The value is
expressed in the document currency.
        volume:
          $ref: '#/components/schemas/ext.UnitValue'
      description: "Current item's input data for custom formula. For REQ
formula only 'attributes' and 'exclusionIndicator' is sent.\r\nIt is recommended
to reduce the fields sent here to a custom formula for action PROCESS_FORMULA by
specifying a projection in response object via 'extendedInput'->'documentInput'-
>'itemInput'->'projection' during COLLECT_ATTRIBUTES call."
    ext.ItemOutput:
      type: object
      properties:
        exclusionIndicator:
          type: string
          description: Condition exclusion indicator flag can be set for
the item which may be taken into account by subsequent extension calls.
Subsequent extension calls may derive own logic based on the value of the
exclusionIndicator or toggle it to a different value. The exclusion indicator
must be a single character.
        subtotals:
          type: array
          description: |-
            Controls whether and in which fields condition values or subtotals
are temporarily stored (for example, customer discount or cost). If the same
field is used for saving different condition values, the individual values are
added. These condition values or subtotals are used as a basis for further
calculation.
            The value of the subtotal flag in the pricing item will be
overwritten with the value passed.
            For example: [{ "flag": "K", "value": 120.00 ] }, { "flag": "1",
"value": 201.32 }].
             The Subtotal flag which can be used are:
               <table><tr><td>Flag</td><td>Description</td></tr><tr><td>"
"(space)</td><td>No separate sub-totals</td></tr><tr><td>1</td><td>Carry over
value to KOMP-KZWI1 (subtotal 1)</td></tr><tr><td>2</td><td>Carry over
value to KOMP-KZWI2 (subtotal 2)</td></tr><tr><td>3</td><td>Carry over value
to KOMP-KZWI3 (subtotal 3)</td></tr><tr><td>4</td><td>Carry over value to
KOMP-KZWI4 (subtotal 4)</td></tr><tr><td>5</td><td>Carry over value to KOMP-
KZWI5 (subtotal 5)</td></tr><tr><td>6</td><td>Carry over value to KOMP-KZWI6
(subtotal 6)</td></tr><tr><td>7</td><td>Carry over value to KOMP_BONBA (rebate
basis 1)</td></tr><tr><td>8</td><td>Copy values according to KOMP-PREVA
(preference value)</td></tr><tr><td>9</td><td>Copy values to KOMP-BRTWR (gross
value)</td></tr><tr><td>A</td><td>Carry over price to KOMP-CMPRE (credit
price)</td></tr><tr><td>B</td><td>Carry over value to KOMP-WAVWR (cost)</td></
tr><tr><td>C</td><td>Carry over value to KOMP-GKWRT (statistical value)</
td></tr><tr><td>D</td><td>Copy value to XWORKD</td></tr><tr><td>E</td><td>Copy
value to XWORKE</td></tr><tr><td>F</td><td>Copy value to XWORKF</td></
tr><tr><td>G</td><td>Copy value to XWORKG</td></tr><tr><td>H</td><td>Copy value
to XWORKH</td></tr><tr><td>I</td><td>Copy value to XWORKI</td></tr><tr><td>J</
td><td>Copy value to XWORKJ</td></tr><tr><td>K</td><td>Copy value to XWORKK</
td></tr><tr><td>L</td><td>Copy value to XWORKL</td></tr><tr><td>M</td><td>Copy
value to XWORKM</td></tr><tr><td>Q</td><td>Reserved (IS-OIL)</td></tr><tr><td>S</
td><td>Copy values to KOMP-EFFWR (effective value)</td></tr><tr><td>Y</
td><td>Reserved (IS-OIL)</td></tr><tr><td>Z</td><td>Reserved (IS-OIL)</td></tr></
table>
          items:
            $ref: '#/components/schemas/ext.Subtotal'
      description: Pricing item output data for custom formula. Processed only
for formula types VAL and BAS when called with action PROCESS_FORMULA.
    ext.LastPriceCondition:
      type: object
      properties:
        calculationType:
          type: string
```

```
          description: Specifies how the system calculates prices for
last price condition. For example, the system can calculate a
price as a fixed amount or as a percentage based on quantity,
volume, or weight. <table><tr><td>Calculation Type</td><td>Description</
td></tr><tr><td>A</td><td>Percentage</td></tr><tr><td>B</td><td>Fixed Amount</
td></tr><tr><td>C</td><td>Quantity</td></tr><tr><td>D</td><td>Gross weight</
td></tr><tr><td>E</td><td>Net weight</td></tr><tr><td>F</td><td>Volume</td></
tr><tr><td>G</td><td>Formula</td></tr><tr><td>H</td><td>Percentage (in
hundreds)</td></tr><tr><td>I</td><td>Percentage (travel expenses)</td></
tr><tr><td>J</td><td>Per mile</td></tr><tr><td>K</td><td>Per mile (in
thousands)</td></tr><tr><td>L</td><td>Points</td></tr><tr><td>M</td><td>Quantity
- monthly price</td></tr><tr><td>N</td><td>Quantity - yearly price</td></
tr><tr><td>O</td><td>Quantity - daily price</td></tr><tr><td>P</td><td>Quantity
- weekly price</td></tr><tr><td>Q</td><td>Commodity Formula</td></tr><tr><td>U</
td><td>Percentage FIN</td></tr><tr><td>W</td><td>Percentage (with 6 decimal
places)</td></tr></table>
        conditionBase:
          type: number
          description: 'Base value of last price condition. The unit of
condition base depends on the calculation type of the condition. Refer to
below table: <table><tr><td>Calculation Type</td><td>Condition Base: Unit</td></
tr><tr><td>A, B, H, U, I, W</td><td> Document Currency </td></tr><tr><td>G, Q</
td><td>Unit of measure of last pricing condition base</td></tr><tr><td>Other
types</td><td>Unit of measure of the condition record</td></tr></table>'
        conditionControl:
          type: string
          description: Condition control of last
price condition. <table><tr><td>Condition Control</td><td>Description</
td></tr><tr><td>A</td><td>Adjust for quantity variance</td></tr><tr><td>C</
td><td>Changed manually</td></tr><tr><td>D</td><td>Fixed</td></tr><tr><td>E</
td><td>Value and base fixed</td></tr><tr><td>F</td><td>Value fixed for billed
items</td></tr><tr><td>G</td><td>Base fixed</td></tr><tr><td>H</td><td>Value
fixed for cost price</td></tr></table>
        conditionRate:
          $ref: '#/components/schemas/ext.UnitValue'
        conditionType:
          type: string
          description: Condition type name of last price condition. (e.g. PR00)
        conditionUnit:
          $ref: '#/components/schemas/ext.UnitValue'
        conditionValue:
          type: number
          description: Value of last price condition. The value is expressed in
the document currency.
        counter:
          type: integer
          description: Counter of last price condition (e.g. 1)
          format: int32
        factor:
          type: number
          description: 'The factor corresponds to the price condition''s having
calculation type: M-months, N-years, O-days, P-weeks. Factor is applied to the
product of condition rate and quantity to calculate the condition value for the
given contract duration.'
        manuallyChanged:
          type: boolean
          description: Whether the condition was manually changed or not (e.g.
false)
        quantity:
          $ref: '#/components/schemas/ext.UnitValue'
        stepNumber:
          type: integer
          description: Step number of last price condition (e.g. 80)
          format: int32
        variantConditionFactor:
          type: number
          description: A multiplier originating from Variant Configuration
service that applies to the condition value.
```

```
      description: Last calculated price condition as input data for the
custom formula. Price conditions are condition types with condition class 'B'
in customizing. Input data contains details of the active price condition,
processed before the current condition with the assigned custom routine.
    ext.PricingCondition:
      type: object
      properties:
        calculationType:
          type: string
          description: "This field specifies how the Pricing service calculates
the condition value. The Pricing service can calculate a condition value
in different ways, for example in its absolute amount, as a percentage,
etc.\r\n <table  border= \"solid\"><tr><td>Calculation Type</td><td>Description</
td></tr><tr><td>A</td><td>Percentage</td></tr><tr><td>B</td><td>Fixed Amount</
td></tr><tr><td>C</td><td>Quantity</td></tr><tr><td>D</td><td>Gross weight</
td></tr><tr><td>E</td><td>Net weight</td></tr><tr><td>F</td><td>Volume</td></
tr><tr><td>G</td><td>Formula</td></tr><tr><td>H</td><td>Percentage (in
hundreds)</td></tr><tr><td>I</td><td>Percentage (travel expenses)</td></
tr><tr><td>J</td><td>Per mile</td></tr><tr><td>K</td><td>Per mile (in
thousands)</td></tr><tr><td>L</td><td>Points</td></tr><tr><td>M</td><td>Quantity
- monthly price</td></tr><tr><td>N</td><td>Quantity - yearly price</td></
tr><tr><td>O</td><td>Quantity - daily price</td></tr><tr><td>P</td><td>Quantity
- weekly price</td></tr><tr><td>Q</td><td>Commodity Formula</td></tr><tr><td>U</
td><td>Percentage FIN</td></tr><tr><td>W</td><td>Percentage (with 6 decimal
places)</td></tr></table>"
        conditionBase:
          type: number
          description: 'Condition base. The unit of condition base depends
on the calculation type of the condition. Refer to below table: <table
border= "solid"><tr><td>Calculation Type</td><td>Condition Base: Unit</td></
tr><tr><td>A, B, H, U, I, W</td><td> Document Currency </td></tr><tr><td>G, Q</
td><td>Unit of measure of last pricing condition base</td></tr><tr><td>Other
types</td><td>Unit of measure of the condition record</td></tr></table>'
        conditionClass:
          type: string
          description: Preliminary structuring of condition types
e.g. in surcharges and discounts or prices. Allows standardized
processing of individual condition classes within the system. <table
border= "solid"><tr><td>Condition Class</td><td>Description</td></tr><tr><td>A</
td><td>Discount or surcharge</td></tr><tr><td>B</td><td>Prices</td></
tr><tr><td>C</td><td>Expense reimbursement</td></tr><tr><td>D</td><td>Taxes</
td></tr><tr><td>E</td><td>Extra pay</td></tr><tr><td>F</td><td>Fees for
differential</td></tr><tr><td>G</td><td>Tax classification</td></tr><tr><td>H</
td><td>Determining sales deal</td></tr><tr><td>Q</td><td>Totals record for fees
only</td></tr><tr><td>W</td><td>Wage withholding tax</td></tr></table>
        conditionRate:
          $ref: '#/components/schemas/ext.UnitValue'
        conditionType:
          type: string
          description: In pricing, the pricing elements are depicted using
condition types. Pricing elements can be prices, surcharges, or discounts.
        conditionUnit:
          $ref: '#/components/schemas/ext.UnitValue'
        conditionValue:
          type: number
          description: The value resulting from pricing, for a condition, total,
or sub-total in a document
        counter:
          type: integer
          description: counter
          format: int32
        durationFactor:
          type: number
          description: 'The duration factor corresponds to the condition''s
calculation type: M-months, N-years, O-days, P-weeks. Duration factor is applied
to the product of condition rate and quantity to calculate the condition value
for the given contract duration.'
        inactiveFlag:
```

```
          type: string
          description: "Status of the condition.\r\nFor example:
Condition status will be made inactive if there is any
error during currency conversion.\r\nThe reasons for being
inactive are: <table  border= \"solid\"><tr><td>Inactive Status</
td><td>Description</td></tr><tr><td>A</td><td>Condition exclusion item</td></
tr><tr><td>K</td><td>Inactive due to calculation basis</td></tr><tr><td>M</
td><td>Inactive due to manual entry</td></tr><tr><tr><td>W</td><td>The
document item is statistical</td></tr><tr><td>X</td><td>Inactive via formula
or Inactive due to error</td></tr><tr><td>Y</td><td>Inactive because
of subsequent price</td></tr><td>Z</td><td>Invisible</td></tr><tr><td>\\' \
\'(space)</q></td><td>Condition is active</td></tr></table>"
        origin:
          type: string
          description: |-
            Origin of the condition
             Below are the listed origins with description: <br> <table
border= "solid"><tr><th>Origin</th><th>Description</th></tr><tr><td>" "(space)</
td><td>Subtotal step at item level</td></tr><tr><td>A</td><td>Automatic
pricing</td></tr><tr><td>B</td><td>Duplicated from main item</td></tr><tr><td>C</
td><td>Manually entered/changed</td></tr><tr><td>D</td><td>Header condition</
td></tr><tr><td>E</td><td>Item total</td></tr><tr><td>F</td><td>Condition
supplement</td></tr><tr><td>G</td><td>Original header condition</td></
tr><tr><td>H</td><td>Correction rebate</td></tr><tr><td>I</td><td>Cost
correction</td></tr><tr><td>J</td><td>Transaction tax engine</td></tr></table>
        purpose:
          type: string
          description: Name assigned in customizing to a condition record in
field conditionFunction. Used to aggregate conditions with same purpose, e.g.
all types of variant conditions.
        recordId:
          type: string
          description: Number that uniquely identifies a condition record.
        scaleBaseType:
          type: string
          description: Determines how the system interprets a pricing scale
in a condition. For example, the scale can be based on quantity, weight,
or volume. <table  border= "solid"><tr><td>Scale base type</td><td>Description</
td></tr><tr><td>" "(space) or "" (blank)</td><td>No scales</td></tr><tr><td>B</
td><td>Value scale</td></tr><tr><td>C</td><td>Quantity scale</td></tr><tr><td>D</
td><td>Gross weight scale</td></tr><tr><td>E</td><td>net weight scale</td></
tr><tr><td>F</td><td>Volume scale</td></tr><tr><td>G</td><td>Scale based on a
formula</td></tr><tr><td>L</td><td>Point scale</td></tr><tr><td>M</td><td>Time
period scale - Month</td></tr><tr><td>N</td><td>Time period scale - Year</td></
tr><tr><td>O</td><td>Time period scale - Day</td></tr><tr><td>P</td><td>Time
period scale - Week</td></tr><tr><td>R</td><td>Distance</td></tr><tr><td>S</
td><td>Number of shipping units</td></tr></table>
        statistical:
          type: boolean
          description: Whether the condition is a statistical condition.
Statistical conditions are not considered in net price calculations.
        stepNumber:
          type: integer
          description: Number that determines the sequence of the condition
types within a procedure.
          format: int32
        structureCondition:
          type: string
          description: Structure Condition. <table  border=
"solid"><tr><td>Structure Condition</td><td>Description</td></tr><tr><td>A</
td><td>Condition to be duplicated. Condition determination happens at the
main item and same condition is duplicated for sub items.</td></tr><tr><td>B</
td><td>Cumulation condition. Cumulate the sub item price and display it at
the main item level.</td></tr><tr><td>' ' (space)</td><td> Not a structure
condition. </td></tr></table>
        variantCondition:
          type: boolean
```

```
          description: Whether the condition is a variant condition. Variant
conditions are available for configurable products.
        variantConditionFactor:
          type: number
          description: A multiplier originating from Variant Configuration
service that applies to the condition value.
        variantConditionKey:
          type: string
          description: Technical identifier for the variant condition
      description: "<p>Pricing condition input data for custom
formula.\r\nContains the calculated data of the condition to which the formula
has been assigned. \r\nIt is recommended to reduce the fields sent to custom
formula for action PROCESS_FORMULA by specifying a projection in the response
object via 'extendedInput'->'documentInput'->'pricingCondition'->'projection'
provided during COLLECT_ATTRIBUTES call.\r\nFor REQ formula it is 'null'
by default. By the projection on 'pricingCondition' in COLLECT_ATTRIBUTES's
response via 'extendedInput'->'documentInput'->'pricingCondition'->'projection'
it is possible to just send 'conditionType' here for REQ formula.</p>"
    ext.PricingOutput:
      type: object
      properties:
        condition:
          $ref: '#/components/schemas/ext.ConditionOutput'
        extendedInput:
          $ref: '#/components/schemas/ext.ExtendedInput'
        item:
          $ref: '#/components/schemas/ext.ItemOutput'
        message:
          type: string
          description: Message from custom formula which will be logged
        result:
          type: object
          description: Result from the custom formula
implementation. Expected result format vary based on action
and formula type. For more details refer to below table
<table><tr><td>Action</td><td>Formula Type(s)</td><td>Expected Result Format</
td></tr><tr><td>PROCESS_FORMULA</td><td>VAL, BAS, SCL</td><td>Numeric (e.g.
100.36, 1, "478")</td></tr><tr><td>PROCESS_FORMULA</td><td>REQ</td><td>Boolean
(e.g. true, false) </td></tr><tr><td>PROCESS_FORMULA</td><td>GRP</td><td>String
(e.g. "001", "ABC") </td></tr><tr><td>COLLECT_ATTRIBUTES</td><td> VAL, REQ, SCL,
BAS, GRP</td><td>Array of String (e.g. ["KOMK-LAND1", "KOMP-SKTOF"])</td></tr></
table>
      description: Output of custom formula. The formula's calculated results
are mainly returned here by the 'results' object with or without additional
logging information in 'message' object. Some formula types can overwrite
certain fields via the 'condition' and 'item' response objects. 'extendedInput'
can be used to tell the pricing engine in more detail which data is needed for
formula processing.
    ext.Subtotal:
      type: object
      properties:
        flag:
          type: string
          description: Subtotal Flag
        value:
          type: number
          description: Subtotal Value
      description: Pricing subtotals data for custom formula
    ext.UnitValue:
      type: object
      properties:
        internalUnit:
          type: string
          description: SAP internal code representation for unit (e.g. PC, KG,
USD)
        unit:
          type: string
          description: ISO code for unit (e.g. PCE, KGM, USD)
```

```
        value:
          type: number
          description: Value
      description: Unit of measurement
    ext.input:
      type: object
      properties:
        action:
          type: string
          description: Identifier for the action to be executed
by custom formula implementation. <table  border= "solid"><tr><td>Action</
td><td>Description</td></tr><tr><td>COLLECT_ATTRIBUTES</td><td> Return the list
of pricing attributes used in custom formula implementation.Can tell the pricing
engine which additional data shall be provided for formula processing.</td></
tr><tr><td>PROCESS_FORMULA</td><td> Process the logic implemented in custom
formula and returns the result according to formula type.</td></tr></table>
          example: PROCESS_FORMULA
          enum:
            - COLLECT_ATTRIBUTES
            - PROCESS_FORMULA
        documentInput:
          $ref: '#/components/schemas/ext.DocumentInput'
        formulaNumber:
          type: integer
          description: 'Identifier of the formula number (e.g. 632). Since
there are standard delivered formulas which uses pre-defined number, below is
the acceptable number range for custom formula: <table><tr><td>Formula Type</
td><td>Allowed Formula: Range</td></tr><tr><td>VAL</td><td> Greater than 599
</td></tr><tr><td>BAS</td><td> Greater than 599 </td></tr><tr><td>REQ</td><td>
Greater than 599  </td></tr><tr><td>SCL</td><td> Greater than 599 </td></
tr><tr><td>GRP</td><td> Greater than 59 </td></tr></table>'
          format: int32
          example: 978
        formulaType:
          type: string
          description: 'Identifier of the formula type (e.g. VAL,
REQ). Refer the below table for supported formula types: <table
border= "solid"><tr><td>Formula Type</td><td>Description</td></tr><tr><td>VAL</
td><td>Condition value formula</td></tr><tr><td>BAS</td><td>Condition base
formula</td></tr><tr><td>REQ</td><td>Requirement formula </td></tr><tr><td>SCL</
td><td>Scale base formula</td></tr><tr><td>GRP</td><td>Group key formula</td></
tr></table>'
          example: VAL
      description: Input structure for custom formula, which consists of formula
type, formula number, action, item details, attributes and condition details.
```

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP