# ARIS PROCESS PERFORMANCE MANAGER
# PPM CUSTOMIZING

VERSION 10.5

October 2019

This document applies to ARIS Process Performance Manager Version 10.5 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2000 - 2019 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

# Contents

Contents ..... I

1 Text conventions ..... 1

2 General ..... 2

3 Overview ..... 3

3.1 Configuration components ..... 3
3.2 Command line programs ..... 4
3.3 Methodological procedure ..... 5
3.4 Configuration component hierarchy ..... 5

4 Interface languages ..... 7

4.1 User interface languages ..... 7
4.2 Interface language for display of configuration elements ..... 8
4.2.1 Using multi-byte character sets for configuration elements ..... 9

5 Internal names ..... 10

6 Attribute types and attribute type groups ..... 11

6.1 Data types ..... 11
6.1.1 Internal data types ..... 11
6.1.2 User-defined data types ..... 13
6.1.2.1 User-defined data types in multi-byte character sets ..... 14
6.2 Definition of attribute types and attribute type groups ..... 15
6.2.1 Definition of attribute types ..... 15
6.2.2 Definition of attribute type groups ..... 15
6.2.3 Configuration of attribute types and attribute type groups ..... 16
6.2.3.1 Attribute type and attribute type group definition in multi-byte character sets ..... 18

7 Process merge ..... 19

7.1 Process hierarchies ..... 19
7.2 Key rules ..... 20
7.2.1 Process key rules ..... 21
7.2.2 Hierarchy key rules ..... 22
7.2.3 Shared fragment key rules ..... 23
7.2.4 Merge key rules ..... 24
7.2.4.1 Key-based merge ..... 27
7.2.4.2 Merge based on sort order ..... 27
7.2.4.3 Combining merge methods ..... 29
7.2.5 Object key rules ..... 30
7.2.6 Output behavior of messages ..... 31
7.2.7 Configuration file ..... 31
7.3 Process fragment merge ..... 33
7.3.1 Merge mode "Replace" ..... 33
7.3.2 Merge mode "Update" ..... 34
7.4 Merge events ..... 36
7.4.1 Parallel paths with multi-valued keys ..... 37
7.4.2 Merge mode ..... 38

7.5 Attribute copy rules ... 39
7.6 Anonymizing ... 40

8 Process typification ... 43
8.1 Create typification rules ... 43
8.1.1 Measure configuration ... 43
8.1.2 Process tree configuration ... 44
8.1.2.1 Prioritization ... 45
8.1.3 Definition of attribute calculations ... 45
8.1.3.1 Calculation classes ... 49
8.1.3.1.1 Log output for calculation classes ... 49
8.1.3.1.2 Time measures ... 50
8.1.3.1.3 Function measures ... 50
8.1.3.1.4 Process measures ... 55
8.1.3.1.5 Frequency measures ... 55
8.1.3.1.5.1 Function measures ... 55
8.1.3.1.5.2 Process measures ... 57
8.1.3.1.5.3 Process cost rates ... 59
8.1.3.1.5.4 More process measures ... 60
8.1.3.1.5.5 Environmentally relevant calculations ... 60
8.1.3.1.6 Relation measures ... 68
8.1.3.1.7 Process conformance ... 70
8.1.3.1.7.1 Conformance rate measure ... 70
8.1.3.1.7.2 Conformance issue relation ... 70
8.1.3.1.8 Convert time spans in milliseconds ... 71
8.1.3.1.9 Mark as large EPC ... 71
8.1.3.2 Operands ... 72
8.1.3.2.1 Set of values (XML element attribute) ... 72
8.1.3.2.2 Values (XML element filteredattribute) ... 74
8.1.3.2.3 Constants (XML element constant) ... 76
8.1.3.2.4 Determining attribute values ... 77
8.1.3.2.4.1 Attribute values without object reference ... 78
8.1.3.2.4.2 Attribute values with object reference ... 78
8.1.3.3 Conditional attribute type access ... 79
8.1.3.4 Operators ... 80
8.1.3.4.1 Mathematic operators ... 82
8.1.3.4.2 Operators resulting in a set of values ... 91
8.1.3.4.3 Operators producing a value ... 94
8.1.3.4.4 Logical operators ... 100
8.1.3.4.5 Conditional operator ... 112
8.1.3.4.6 String operators ... 113
8.1.3.4.7 Time operators ... 116
8.1.3.4.8 Conditional attribute type calculation ... 121
8.1.3.5 Nesting of operators ... 122
8.1.3.6 Calculation functions ... 123
8.1.3.7 Change the attribute type ... 125
8.1.3.8 Summary ... 125
8.1.3.9 Example attribute calculations ... 126
8.1.3.10 Special features of attribute calculation ... 130
8.1.3.10.1 AT_INTERNAL_NO_CUBE_ENTRY function attribute ... 130
8.1.4 Typification rules in CTK ... 131

8.2 Typification by attribute calculation ..... 131

9 Definition of measures, dimensions, attribute calculations, and relations ..... 133
9.1 Terminology ..... 133
9.1.1 Measures ..... 133
9.1.1.1 Process instance-dependent measures ..... 134
9.1.1.2 Process instance-independent measures (PIKIs) ..... 134
9.1.2 Dimensions ..... 135
9.2 Definition of measures ..... 135
9.2.1 Definition of standard measures ..... 136
9.2.1.1 Formatting measure values ..... 139
9.2.1.2 Definition of process cost measures ..... 140
9.2.2 Measure definition in multi-byte character sets ..... 141
9.2.3 Definition of cardinality measures ..... 142
9.2.4 Definition of process instance-independent measures ..... 143
9.2.4.1 Usage (type) of a data series ..... 149
9.2.4.2 Dimension reference ..... 149
9.2.4.3 Definition of process instance-independent measures in multi-byte character sets ..... 151
9.2.4.4 Configuration import ..... 151
9.2.4.5 Data series migration ..... 153
9.2.4.6 Additional information: User-defined measures based on process instance-independent measures ..... 153
9.2.5 Definition of measure groups ..... 154
9.2.5.1 Visible measure groups ..... 157
9.2.5.2 Group of invisible measures ..... 158
9.3 Definition of dimensions ..... 159
9.3.1 Definition of dimension groups ..... 160
9.3.2 Text dimensions ..... 162
9.3.2.1 General XML structure ..... 162
9.3.2.1.1 One-level dimension ..... 162
9.3.2.1.2 Two-level dimension ..... 163
9.3.2.1.3 N-level dimension ..... 163
9.3.2.2 Configuration ..... 167
9.3.2.2.1 One-level dimensions ..... 167
9.3.2.2.2 Two-level dimensions ..... 169
9.3.2.2.3 N-level dimensions ..... 171
9.3.2.3 Import dimension values ..... 173
9.3.3 Floating point dimensions ..... 173
9.3.4 Time dimensions ..... 174
9.3.4.1 Time dimensions for the Early alert system ..... 176
9.3.4.2 Special feature for calculation of critical time attributes ..... 176
9.3.5 Time range dimensions ..... 179
9.3.6 Time of day dimensions ..... 181
9.3.7 Search dimensions ..... 182
9.3.8 Variant dimension ..... 184
9.3.8.1 Attribute configuration ..... 184
9.3.8.2 Measure configuration - dimension type ..... 184
9.3.8.3 Process tree configuration ..... 185
9.3.8.4 Usage of variant attributes during import ..... 186
9.3.9 Shared function dimension ..... 186
9.3.10 Using organizational units as dimensions ..... 188

9.4 Definition of data access dimensions ..... 188
9.4.1 Using data access dimensions ..... 190
9.5 Process tree definition ..... 191
9.5.1 Registration of measures and dimensions at the PPM system ..... 194
9.5.1.1 Register measure ..... 194
9.5.1.1.1 Register relation measure ..... 195
9.5.1.1.2 Register measures and dimensions of process instance-independent data series ..... 196
9.5.1.1.2.1 Special case: Register referenced dimensions ..... 196
9.5.1.2 Register dimension ..... 197
9.5.1.2.1 Register reference dimension ..... 198
9.5.1.2.2 Register relation dimension ..... 199
9.5.2 Automatic process tree expansion ..... 199
9.5.3 Manual process tree expansion ..... 200
9.5.4 Definition of process tree in multi-byte character sets ..... 200
9.6 Relations ..... 202
9.6.1 Definition of relations ..... 202
9.6.1.1 Reference dimensions ..... 203
9.6.2 Definition of relation calculations ..... 204
9.6.3 Definition of relation measures ..... 209
9.6.4 Definition of relation and organizational dimensions ..... 211

10 Change aggregation behavior ..... 213
10.1 Configure the internal aggregation attribute ..... 213
10.2 Assign aggregation values ..... 214

11 System connections ..... 217
11.1 SAP executables ..... 217
11.1.1 Software requirements ..... 217
11.1.2 Privileges in the SAP system ..... 217
11.1.3 Transaction call ..... 217
11.1.4 Configuration ..... 217
11.1.4.1 Configuration examples ..... 218
11.1.4.2 Explanations regarding the DTD ..... 235

12 Legal information ..... 240
12.1 Documentation scope ..... 240
12.2 Data protection ..... 241

# 1 Text conventions

Menu items, file names, etc. are indicated in texts as follows:

- Menu items, key combinations, dialogs, file names, entries, etc. are displayed in **bold**.
- User-defined entries are shown as **<bold text in angle brackets>**.
- Example texts that are too long to fit on a single line, such as a long directory path, are wrapped to the next line by using ↵ at the end of the line.
- File extracts are shown in this font format:

  `This paragraph contains a file extract.`
- Warnings have a colored background:

  **Warning**

  This paragraph contains a warning.

# 2 General

This manual describes the configuration of ARIS Process Performance Manager (PPM). It provides the PPM system administrator with basic knowledge and configuration know-how to support him in configuration for different usage scenarios and analysis tasks.

The user guide is aimed at PPM Customizing Toolkit users who are application configuration experts.

As an **application configuration expert** you are responsible for customizing all ETL processes (**E**xtracting source system data, **T**ransforming the data, **L**oading the data into the target database), which includes process nesting, process typification, as well as measure and dimension calculation.

Please note that this guide is not intended to replace user or configuration training. It is a source of reference containing information that supplements the information provided in the manuals and online help.

# 3 Overview

Before you can import data into PPM, you need to break down the processes in the source system to be analyzed and, on that basis, create a configuration for the PPM system. This process is referred to as customizing. Once customizing has been completed, a set of specific XML configuration files is available, which can be used to initialize the PPM system.

The import adapters for importing process instance fragments into the PPM database are described in the **PPM Data Import** manual.

## 3.1 Configuration components

Configuration components are divided into the following categories:

### INTERFACE LANGUAGES

PPM differentiates between two categories of interface languages: the user interface language and the language for displaying database contents that have been imported using the configuration files.

### DATA TYPES

In PPM, a distinction is made between internal and user-defined data types. Internal data types cannot be changed. An XML file can be used to import any number of new data types into the PPM system.

### SPECIFIC PPM ATTRIBUTES

Attributes are the information carriers of the PPM system. Attribute values allow data from the source system to reach the PPM system. Calculation results are also saved as attributes. PPM is supplied with a comprehensive set of default attributes, which can be supplemented with user-defined attributes.

### PROCESS KEY RULES

Process key rules determine how process keys are calculated.

Process keys identify associated process instance fragments. Associated process instance fragments are written into a process instance unlinked using the process keys. Process keys are created when importing the process instance fragments.

### MERGE KEY RULES

Merge key rules determine the attributes used by PPM to create merge keys when importing the process instance fragments. When merging events (event merge), the merge events with an identical merge key are used to link the process instance fragments.

### OBJECT KEY RULES

Object key rules determine how object keys are calculated.

Object keys identify identical objects. This ensures that these objects are overwritten if the data is imported again and rules out unintentional multiple occurrences of identical objects within a process instance.

### ATTRIBUTE COPY RULES

Attribute copy rules define the object attributes that are copied to the merged process instance after the merge process. These attributes are required to calculate process measures and to create dimensions.

### TYPIFICATION RULES

With typification rules, the imported process instances are allocated to a particular process type.

### MEASURE DEFINITION

The measure definition is made up of the definition of the measures (type and calculation rule) and the definition of a process tree.
PPM already contains the calculation rule for many standard measures and a default process tree. If you require further measures, you can define supplementary indicators and their calculation.

## 3.2 Command line programs

After creation of the configuration files, the source system data is imported into the PPM system as follows:

| Process | PPM command | Documentation |
|---|---|---|
| Initialize PPM database | runinitdb -init -client <client> | **PPM Operation Guide** |
| Import process instance fragments (XML import adapter or other import adapter) | runxmlimport -client <client> -user ... | **PPM data import** |
| Create process instances from imported process instance fragments, typify process instances and calculate measures | runppmimport -client <client> ... | **PPM Operation Guide** |

After the process instances have been imported and calculated, they can be analyzed and evaluated in the PPM system front-end.

## 3.3 Methodological procedure

Before you start to create the PPM configuration files, you must define the source system activity flows or processes you are analyzing and the variables you want to use. The information required to calculate and process the process instances is identified in the source system (values for process merging, typification and measure calculation).

The procedure for extracting data from the source system and storing the process instance fragments in the PPM system is defined by configuring the relevant PPM import adapter. Importing XML data into the PPM system is described in detail in the technical reference **PPM Data Import**.

If all of the required source system information is known, an assignment of source system information carriers to PPM attributes is created. Depending on the desired analysis results data types and attributes for the measures are determined, along with their calculation rules and attributes for the creation of dimensions.

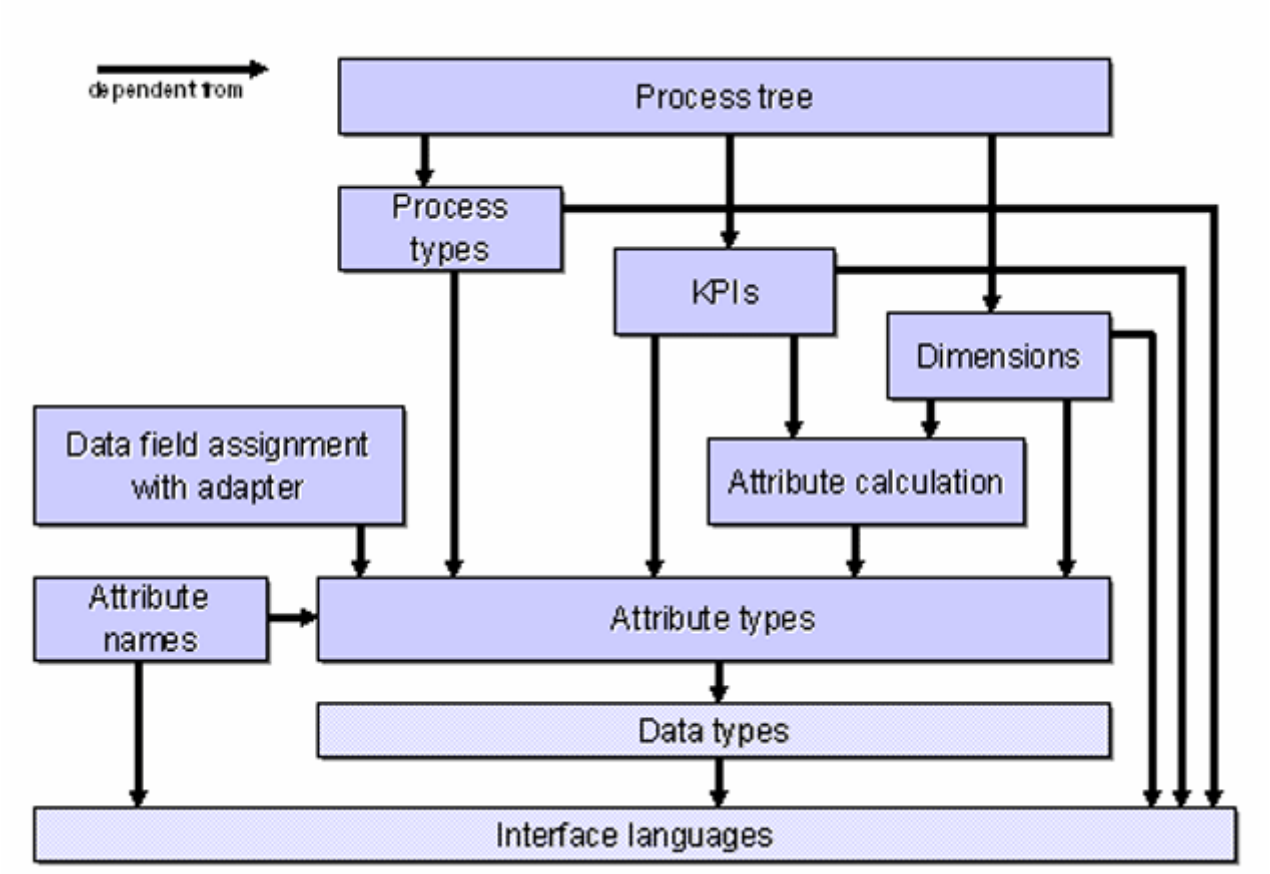
## 3.4 Configuration component hierarchy

The table below shows the assignment of XML configuration files to the PPM components.

| PPM component | XML file (in the xml directory of the default client templates) |
|---|---|
| Process tree | *_processtree.xml |
| Measure calculation | *_keyindicator.xml<br>*_kigroup.xml |
| Creation of keys and merge | *_keyrules.xml<br>*_merger.xml |
| Fragment definition and mapping | No default files. Files must be created using PPM Customizing Toolkit. |
| Attributes | *_attributetypes.xml<br>*_attributenames_<language>.xml (one file for each language) |
| Data types | *_datatypes.xml |
| Interface languages | *_locales.xml |

Then default client templates of the PPM system are stored under <PPM installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ctk\examples\custom\.

The graphic below illustrates the dependencies between the different PPM configuration components.




When initializing the PPM database, ensure that the configuration steps are performed in the correct sequence:

1. Languages
2. Data types
3. Attributes
4. Process merge and typification
5. Measure calculation
6. Process tree

# 4 Interface languages

PPM differentiates between two categories of interface languages: the user interface language, which is used for menu items, dialog boxes, etc., and the language for displaying configuration elements (measure names, dimensions, etc.).

Data that has been imported by importing process instance fragments into the PPM system is always displayed in the source system language, regardless of the selected interface language.

## 4.1 User interface languages

The user sets the user interface language when logging into the system.

PPM supports English and German interface texts for menu items, dialog boxes, etc. The interface texts are contained in the code for the PPM software.

**Warning**

In order to be able to use PPM in multiple languages, an international version of Java Runtime Environment (JRE) must be installed.

The languages available for selection in the Login dialog can be set separately for each PPM client in the file ***_locales.xml**. The file is structured as follows:

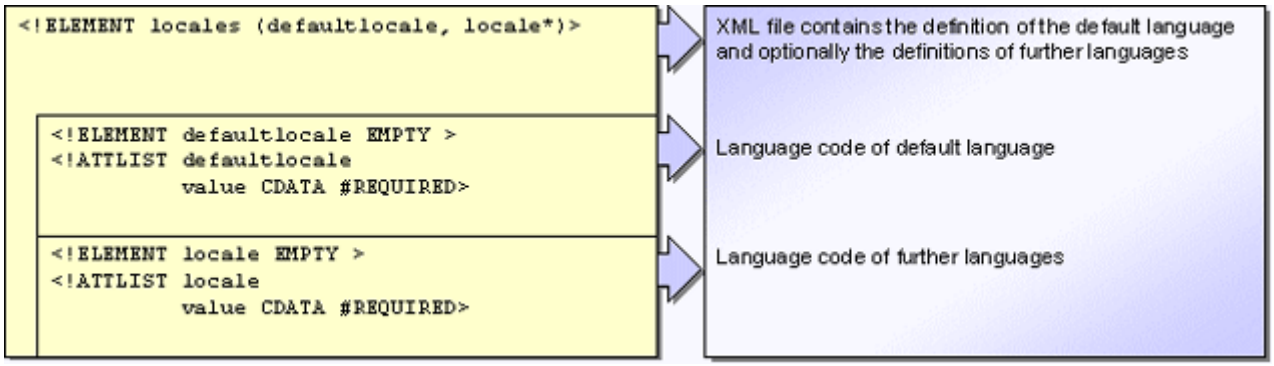
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE locales SYSTEM "Locales.dtd">
<locales>
  <defaultlocale value="en" />
  <locale value="de" />
  <locale value="fr" />
</locales>
```

If the language for the specified language code cannot be determined, for example, an unknown language code is entered in a login URL, the default language is used:

`http://<Web server>/ppm/html/index.html?language=1234`

You can only specify one of the languages available for the user interface as the default language.

The XML configuration file ***_locales.xml** is defined by the following DTD:



## 4.2 Interface language for display of configuration elements

All PPM interface elements that are based on configuration files (name of measures, measure groups and dimensions) can be specified in any language. The naming of elements in the different languages is defined by the **description** XML element, which is used in all XML configuration files. In addition, a language-specific description can be added for each of these interface elements, which is displayed as a tooltip in the front-end.

DTD for XML element **description** (file **_description.dtd**):

```
<!ELEMENT description (#PCDATA)>
<!ATTLIST description
          language NMTOKEN #REQUIRED
          name     CDATA   #REQUIRED>
```

Descriptions are language-specific They consist of a language code and the associated description text. The language code used must be specified in the file **Locales.xml**.

The language-specific descriptive text is specified in the **Name** attribute which is displayed in the PPM user interface.

### Example

Extract from the XML measure configuration:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM 'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <kidef name="PNUM" type="PROCESS" attrname="AT_KI_PNUM"
        calculated="TRUE" distribution="FALSE"
        standarddeviation="FALSE" retrievertype=
        "NUM_KEYINDICATOR" sharedfunctionki="FALSE"
        functionspanki="FALSE" dimreferring="LOOSE"
        importmode="OPTIONAL">
    <description language="de" name="Prozessanzahl">
      Anzahl der tatsächlich durchlaufenen Prozesse
    </description>
    <description language="en" name="Number of processes">
      Number of processes actually passed through
    </description>
  </kidef>
  ...
</keyindicatorconfig>
```

The content of the text specified in the **#PCDATA** tag of the relevant **description** element is displayed as a language-specific tooltip in the PPM interface.

**Warning**

To configure the PPM system successfully, you must enter at least the description in the default language for every configuration element.

Depending on the PPM login language used, the dialog boxes in the PPM front-end show the language-specific name of the configuration elements, for example in the process attribute dialog box the names of attribute type groups and attribute types. The language-specific names are defined in XML configuration files.

### 4.2.1 Using multi-byte character sets for configuration elements

PPM supports the display of source system data and certain configuration elements using local character sets that are not included in the ANSI character set and are coded with a multi-byte character set (MBCS). Examples are Japanese Kanji and Greek characters.

All XML files imported into the PPM system and not based on the character set for a Western European language, must reference the **UTF-8** character set as encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<...>
...
</...>
```

In principle, tooltips and all language-specific names displayed in the PPM user interface can be displayed using a multi-byte character set. Specifically, this applies to the following elements:

- Attribute types
- Attribute type groups
- Measures
- Measure groups
- Dimensions
- Process instance-independent measures (PIKIs)
- Process types
- Process type groups

Furthermore, the language-specific names of user-defined data types can be entered using a multi-byte character set.

For attribute types and attribute type groups, the internal names (**key** XML attribute) can also be configured with a multi-byte character set.

# 5 Internal names

In the PPM configuration files, configuration elements are referenced using a unique, language-independent internal name. This table shows the XML attributes that define the internal name.

| PPM configuration element | XML attribute |
|---|---|
| Attributes | key |
| Data types | name |
| Dimensions | name |
| Measures | name |
| Calculation functions<br>(for example, typification rules) | name |

**Warning**

Internal names are used to reference configuration elements. The internal name of an object in a process instance is specified in the **AT_OBJNAME_INTERN** object attribute.

Internal names begin with a letter and consist of capital letters with no special characters (A-Z), figures (0-9), and the _ (underscore) character.

In practice, the following guidelines for creating internal names have proved useful:

| PPM configuration element | Prefix |
|---|---|
| Attribute names | AT_ |
| Attribute group names | AG_ |
| Measure groups | KI_GROUP_ |
| Dimension groups | DIM_GROUP_ |
| Measure attributes | AT_KI_ |
| Measure names | KI_ |
| Dimension names | D_ |
| Typification rules | TYP_ |

# 6 Attribute types and attribute type groups

Attributes are the data repository of the PPM system. Attributes with a corresponding data type must be defined for instance data extracted, all measures and all dimensions. Attributes can be summarized into attribute groups.

## 6.1 Data types

In PPM, a distinction is made between internal and user-defined data types.

### 6.1.1 Internal data types

The PPM system provides the following internal data types. These cannot be changed.

| Data type | Example (Description) | Units or scaling levels | Units (Description) |
|---|---|---|---|
| **BOOLEAN** | "true" | - | - |
| **TEXT** | "Example text"<br>String | - | - |
| **TEXTPAIR** | "Text 1\Text 2"<br>2 strings separated by a backslash | **LEVEL1SCALE**<br>**LEVEL2SCALE** | Rough level<br>Detailed level |
| **LONG** | "-231456789"<br>Integers | - | - |
| **DOUBLE** | "3.1428"<br>Floating point numbers separated by a decimal point | - | - |
| **DAY** | "24.03.2003"<br>Date in the format dd.MM.yyyy | **DAYSCALE**<br>**WEEKSCALE**<br>**MONTHSCALE**<br>**QUARTERSCALE**<br>**YEARSCALE** | No unit, but levels of accuracy:<br>correct to day<br>correct to week<br>correct to month<br>correct to quarter<br>correct to year |

| Data type | Example (Description) | Units or scaling levels | Units (Description) |
|---|---|---|---|
| **TIME** | "01.01.2002 08:15:23"<br>Date and time in the format dd.MM.yyyy hh:mm:ss<br>The data type is identical to the data types **DATE** and **TIMESTAMP** | **SECONDSCALE**<br>**MINUTESCALE**<br>**HOURSCALE**<br>**DAYSCALE**<br>**WEEKSCALE**<br>**MONTHSCALE**<br>**QUARTERSCALE**<br>**YEARSCALE** | No unit, but levels of accuracy:<br>correct to second<br>correct to minute<br>correct to hour<br>correct to day<br>correct to week<br>correct to month<br>correct to quarter<br>correct to year |
| **TIMEOFDAY** | "12:41:56"<br>Time of the day in format hh:mm:ss | **SECOND_OF_DAY_SCALE**<br>**MINUTE_OF_DAY_SCALE**<br>**HOUR_OF_DAY_SCALE** | No unit, but levels of accuracy:<br>correct to second<br>correct to minute<br>correct to hour |
| **TIMESPAN** | "23 SECOND"<br>Time span | **SECOND**<br>**MINUTE**<br>**HOUR**<br>**DAY**<br>**WEEK**<br>**MONTH**<br>**YEAR** | Second<br>Minute<br>Hour<br>Day<br>Week<br>Month<br>Year |
| **FACTORY TIMESPAN** | "23 FACTORY_HOUR"<br>Time span based on the factory calendar. Only the pure working time is taken into account. | **FACTORY_SECOND**<br>**FACTORY_MINUTE**<br>**FACTORY_HOUR**<br>**FACTORY_DAY**<br>**FACTORY_WEEK**<br>**FACTORY_MONTH**<br>**FACTORY_YEAR** | Person-second<br>Person-minute<br>Person-hour<br>Person-day<br>Person-week<br>Person-month<br>Person-year |
| **FREQUENCY** | "86400 PER_DAY"<br>Number per unit of time | **PER_SECOND**<br>**PER_MINUTE**<br>**PER_HOUR**<br>**PER_DAY**<br>**PER_WEEK**<br>**PER_MONTH**<br>**PER_YEAR** | per second<br>per minute<br>per hour<br>per day<br>per week<br>per month<br>per year |

| Data type | Example (Description) | Units or scaling levels | Units (Description) |
|---|---|---|---|
| **PERCENTAGE** | "63 PERCENT"<br>Percentage | **PERCENT**<br>**VALUE_ONLY** | Percent<br>no unit (factor display) |

For the time-based data types, the base unit is always seconds. The **PERCENTAGE** data type does not have a base unit.

**Warning**

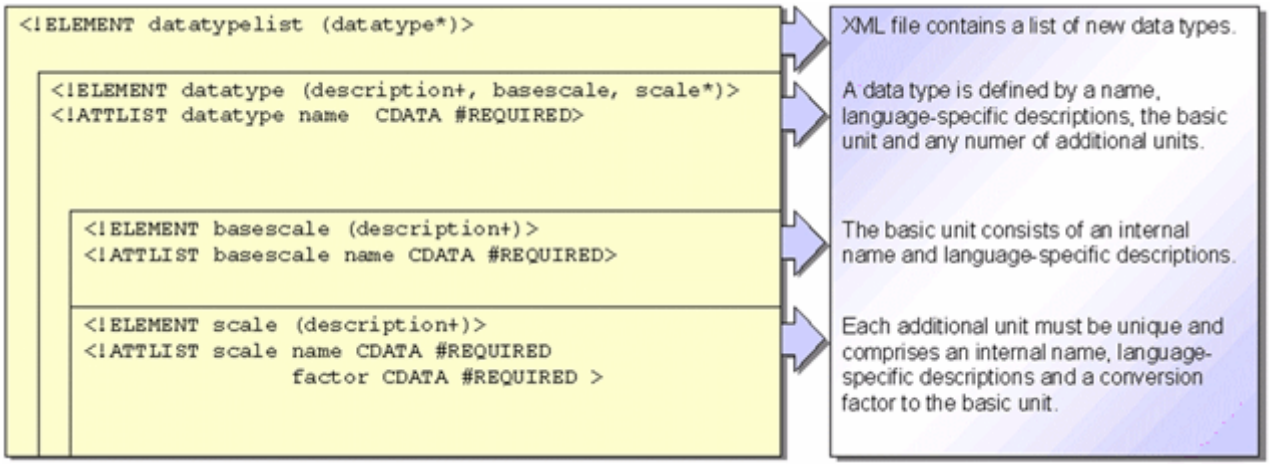
The **TEXTPAIR** data type is used internally by PPM to process binary query results (for example, process type group\process type). This data type is not suitable for direct data exchange, as the **backslash** (****) separator is replaced by the **slash** (***/***) character when importing an XML file.

### 6.1.2 User-defined data types

An XML file can be used to import any number of new data types into the PPM system. The following example file creates a new **Costs** data type (internal name **COST**) with a base unit of **Euros (EUR)** and the additional unit of **Dollars (USD)** with a corresponding conversion factor.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datatypelist SYSTEM 'userdefdatatypes.dtd'>
<datatypelist>
  ...
  <!-- Data type: Costs -->
  <datatype name="COST">
      <description language="de" name ="Kosten"/>
      <description language="en" name ="Costs"/>
      <basescale name="EUR">
          <description language="de" name="EUR"/>
          <description language="en" name="EUR"/>
      </basescale>
      <scale name="USD" factor="0.9">
          <description language="de" name="US Dollar"/>
          <description language="en" name="US Dollars"/>
      </scale>
  </datatype>
  ...
</datatypelist>
```

Document type definition of the XML file for the definition of new PPM data types (file **userdefdatatypes.dtd**):




### IMPORT AND EXPORT OF USER-DEFINED DATA TYPES

You perform the import and export of user-defined data types by executing the command **runppmconfig** with the option **-datatypes** on the PPM server computer (see **PPM Operation Guide**). When importing, the internal name (**datatype name** XML tag) of the data type to be imported is checked. If a data type with the same name already exists in the PPM system, this data type is not imported and a corresponding message is output.

Once imported, user-defined data types cannot subsequently be deleted from the PPM system. It is possible to overwrite their definition only by specifying the **-overwrite** option in the **runppmconfig** command line program.

## 6.1.2.1 User-defined data types in multi-byte character sets

The following extract from the XML configuration file for data types shows an example of the definition options for a user-defined data type when using a multi-byte character set:

Example with base scaling and one other scaling:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE datatypelist SYSTEM "userdefdatatypes.dtd">
<datatypelist>
...
<datatype name="COST">
<description language="en" name="Costs"/>
<description language="en" name="Costs" />
<description language="el" name="έξοδα"/>
   <basescale name="EUR">
      <description language="de" name="EUR"/>
      <description language="en" name="EUR"/>
      <description language="el" name="ΕΥΡ"/>
   </basescale>
   <scale factor="0.001" name="TEUR">
   <description language="de" name="TEUR" />
   <description language="en" name="EUR Thousands" />
   <description language="el" name="Χ.ΕΥΡ"/>
   </scale>
```

```
</datatype>
...
</datatypelist>
```

You can also carry out the user-specific configuration of the file **DataTypes.xml** using PPM Customizing Toolkit.

## 6.2 Definition of attribute types and attribute type groups

All attribute types and attribute type groups known to the PPM system are defined in the XML configuration files ***_AttributeNames_<language>.xml** and ***_AttributeTypes.xml**.

Specify **id="auto"** in the attribute type definition if you want attribute type identifiers or attribute type group identifiers to be automatically generated during the import.

### 6.2.1 Definition of attribute types

Attributes are defined by specifying a unique identifier (**id** XML attribute), a unique internal name (**key** XML attribute) and a data type (**type** XML attribute). Attributes can optionally be assigned to an attribute group (**group** XML attribute).
The identifiers up to **500** are internally reserved for default attributes. These cannot be used for configuration.

### 6.2.2 Definition of attribute type groups

Attribute groups are defined by specifying a unique identifier (**id** XML attribute) and a unique internal name (**key** XML attribute).
The optional specification of the internal name of the superordinate attribute type group (**group** XML attribute) allows attribute type groups to be arranged in a tree structure.

The internal name (**key** XML attribute) is stated in the subsequent **name**.

Use the **AG_INVISIBLE** attribute group pre-assigned by the system for attributes you do not want to be displayed in the PPM user interface in the **Object attributes** or **Process attributes** dialog box in the EPC view .
By default, this attribute group is not defined.

## 6.2.3 Configuration of attribute types and attribute type groups

The XML configuration files ***_AttributeNames_<language>.xml** and ***_AttributeTypes.xml** are defined by the following document type definitions:

### ATTRIBUTETYPES.DTD

```
<!ELEMENT attributetypes (groupdefinition
                          |attributedefinition*)>

  <!ELEMENT groupdefinition EMPTY>
  <!ATTLIST groupdefinition
            id    CDATA #REQUIRED
            key   CDATA #REQUIRED
            group CDATA #IMPLIED>

  <!ELEMENT attributedefinition EMPTY>
  <!ATTLIST attributedefinition
            id CDATA ""
            key   CDATA #REQUIRED
            type CDATA # REQUIRED
            group CDATA #IMPLIED>
```

XML file contains a list of attribute group definitions and attribute definitions.

**Attribute group definition** Comprises an identifier, a name and optionally the name of the superior group.

**Attribute definition**
Contains an identifier, a name, a data type and otpionally the name of the associated group.

### ATTRIBUTENAMES.DTD

The language-specific attribute names are assigned to the attribute definition using the attribute names (**key** XML attribute).

```
<!ELEMENT attributenames (group |attribute)*>
<!ATTLIST attributenames
          language CDATA #REQUIRED>

  <!ELEMENT group EMPTY>
  <!ATTLIST group
            key   CDATA #REQUIRED
            name CDATA  #REQUIRED>

  <!ELEMENT attribute EMPTY>
  <!ATTLIST attribute
            key  CDATA #REQUIRED
            name CDATA #REQUIRED>
```

XML file contains a list of language-specific attribute group names and attribute names. The language code used must be specified in the file **Locales.xml**.

The language-specific attribute group name is assigned to the attribute group by specifiying the name.

The language-specific attribute name is assigned to the attribute by specifiying the name.

The files ***_attributetypes.xml** and ***_attributenames.xml** are used to define a PPM attribute and an attribute group.

### XML FILE *_ATTRIBUTETYPES.XML

The file contains the following information:

Attribute type group:

- Unique attribute type group identifier (optional)
- Unique attribute type group name (optional)
- Attribute type group name for higher level group (optional)

Attribute type:

- Unique identifier (number above 501)
- Unique name
- Data type
- Attribute type group (optional)

The following file extract shows the definition of a default attribute type and a user-defined attribute type:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE attributetypes SYSTEM "attributetypes.dtd">
<attributetypes>
   ...
   <groupdefinition id="2" key="AG_MERGER"
                       group="AG_INTERNAL"/>
   ...
   <groupdefinition id="5" key="AG_COSTING"/>
   ...
   <attributedefinition key="AT_EPK_KEY"
              type="TEXT" group="AG_MERGER" />
   ...
   <attributedefinition id="1000" key="AT_LS"
           type="TIMESPAN" group="AG_COSTING"/>
   ...
</attributetypes>
```

## XML FILE *_ATTRIBUTENAMES.XML

The file contains the following information:

- Language-specific attribute type names
- Language-specific attribute type group names

You must create a separate attribute type name and attribute type group configuration file for each PPM interface language you want to use.

The following extract from the file ***_attributenames_de.xml** contains the attribute type names in German:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE attributenames SYSTEM "attributenames.dtd">
<attributenames language="de">
   ...
   <group key="AG_MERGER" name="Merger"/>
   ...
   <attribute key="AT_EPK_KEY" name="EPK-Schlüssel"/>
   ...
</attributenames>
```

The following extract from the file ***_attributenames_en.xml** contains the attribute type names in English:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE attributenames SYSTEM "attributenames.dtd">
<attributenames language="en">
  ...
  <group key="AG_MERGER" name="Merge"/>
```

```
  ...
  <attribute key="AT_EPK_KEY" name="EPC key"/>
  ...
</attributenames>
```

### 6.2.3.1 Attribute type and attribute type group definition in multi-byte character sets

The following extracts from the XML configuration files for attribute type definitions show examples of the definition options for user-defined attribute types and attribute type groups when using a multi-byte character set.

#### ENTRIES IN THE FILE ATTRIBUTETYPES.XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE attributetypes SYSTEM "attributetypes.dtd">
<attributetypes>
...
<attributedefinition id="5013" key="ΙΔ_ΧΡ_ΕΠΕΞ"
        type="TIMESPAN" group="ΣΥΝ_ΙΔ_ΔΕΙΚΤ_ΧΡΟΝ"/>
...
</attributetypes>
```

#### CORRESPONDING ENTRIES IN THE FILE ATTRIBUTENAMES_EL.XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE attributenames SYSTEM "attributenames.dtd">
<attributenames language="el">
  ...
 <attribute key="ΙΔ_ΧΡ_ΕΠΕΞ" name="χρόνος επεξεργασίας"/>
  ...
  <group key="ΣΥΝ_ΙΔ_ΔΕΙΚΤ_ΧΡΟΝ" name="δείκτης χρόνου"/>
  ...
</attributenames>
```

# 7 Process merge

In order to be able to merge the imported fragments into complete process instances, you require information from the source system, which PPM uses to identify the fragments belonging to a process instance and to reconstruct the time sequence of the fragments (process logic).

The process merge runs in two stages.

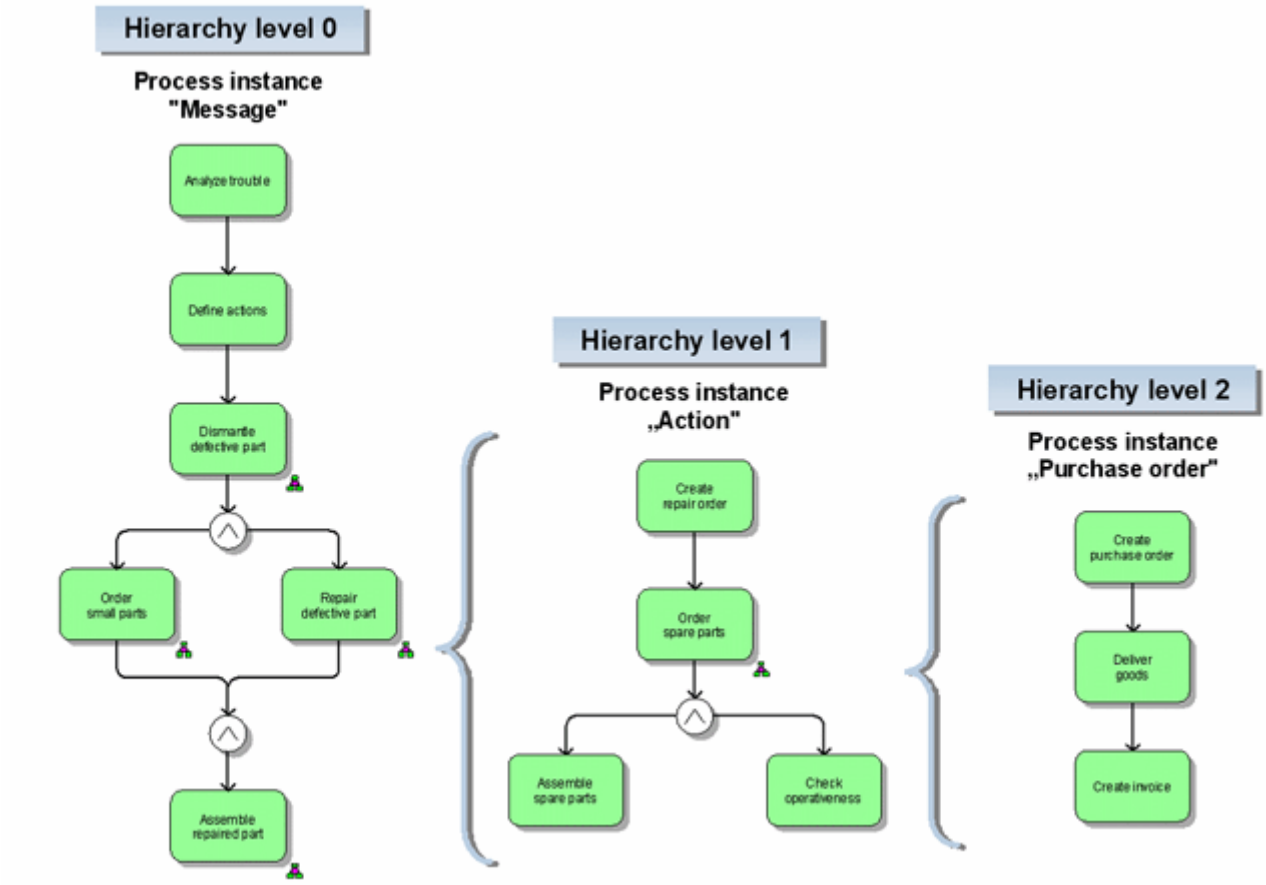
**Procedure**

1. All fragments belonging to a process instance are identified and collected into a process instance (process merge).
2. The unlinked fragments of a process instance are linked to one another (event merge). The event merge can be either key based or based on sort order.

## 7.1 Process hierarchies

Similar to the assignments in ARIS, subordinate process instances can be assigned to functions in PPM. In the EPC view, these functions are given the assignment symbol familiar from ARIS. The assigned process instance can be displayed using the **Open assignment** option in the pop-up menu for the function.

The chart below illustrates the hierarchical refinement of a process instance:




Each assigned process instance is a separate process instance. The process hierarchy only represents an assignment between process instances. It can have any level of detail. All process

instances within a refinement make up a hierarchy structure. Each process instance involved in this hierarchy structure is on a different hierarchy level.

A process instance may not be assigned to multiple functions within a hierarchy structure, as the multiple consideration of attribute values can lead to incorrect results in the measure calculation.

**Warning**

When setting up a hierarchy structure for your processes, ensure that each process instance within the hierarchy structure is assigned to a different process type group.

Each function to which a process instance is assigned represents the subordinate process instance. The AT_INTERNAL_HIER_REF function attribute is a unique reference to the subordinate process instance. The value of the hierarchy key for the subordinate process instance corresponds to the value of the AT_INTERNAL_HIER_REF function attribute. The value of the function attribute is extracted from the source system adapter.

**Warning**

The **AT_INTERNAL_HIER_REF** function attribute cannot be changed later. When setting up process hierarchies, the attribute value at the time of importing is decisive.

The process attributes of the assigned process instance are copied to the function of the higher-level process instance as part of measure calculation in addition to the existing function attributes. Existing attributes of the function are overwritten by attributes of the assigned process instance with the same name. If different measures are calculated for the assigned process instance due to assignment to a different process type when processing the imported process instance fragments (runppmimport), the attributes already copied to the function of the higher level process instance are not deleted and continue to be included in analyses. The function also retains the copied attributes if the assigned process instance is deleted.

To delete the copied attributes for functions with an assigned process instance, you need to re-import the process instance fragments for the higher-level process instance. Appropriate object key rules ensure that the functions are overwritten when the import is repeated.

## 7.2 Key rules

The key rules are divided into five categories according to their purpose:

| Category | XML element | Description |
|---|---|---|
| Process key rules | processkeyrule | Merge associated fragments in a process instance |
| Hierarchy key rules | hierarchykeyrule | Creation of process hierarchies |
| Shared fragment key rules | sharedfragmentkeyrule | Copying shared fragments in process instances |

| Category | XML element | Description |
|---|---|---|
| Merge key rules | mergekeyrule | Combine merge events within a process instance |
| Object key rules | internalobjectkeyrule | Identification of identical objects |

**Warning**

Do not use leading or trailing whitespace characters (such as a blank space or a tab) in keys created by key rules, in attributes that contribute to key rules, or in attributes that refer to any of the keys (for example, AT_INTERNAL_HIER_REF).

## 7.2.1 Process key rules

Process keys uniquely assign process instance fragments to a process instance. Process instance fragments with identical process keys are written unlinked to a process instance.

Process keys can be created efficiently by selecting process instance-specific attribute values (for example, **Order number** or **Processing number**).

When importing, at least one process key must be created for each process fragment.

### Example

The file extract defines a process key rule, which uses the **AT_AUFTRAGSNUMMER** attribute type for the **EVT_START** and **EVT_END** events to create the process key.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyrules SYSTEM "keyrules.dtd">
<keyrules>
  ...
  <processkeyrule>
    <refobjects>
      <refobject objecttype="OT_EVT">
        <objectname name="EVT_START"/>
        <objectname name="EVT_END"/>
      </refobject>
    </refobjects>
    <keyparts>
      <keypart attributetype="AT_AUFTRAGSNUMMER"/>
    </keyparts>
  </processkeyrule>
  ...
</keyrules>
```

**Warning**

Process fragment instances, for which no process key can be calculated, are not imported as they cannot be assigned to a process instance. A warning message is output.

### RETAIN ALL PROCESS ATTRIBUTES WHEN MERGING

By default, when merging two process instances only the process attributes of the most recent fragment (imported later) are retained in the resulting fragment.

When merging two process instance fragments, if the combined set of process attributes for both fragments is to be transferred to the merged fragment, you need to overwrite the default behavior by specifying the **ZRetainingProcessAttributesPMAlgo** class. The following file extract illustrates the merge configuration:

```
...
<mergerconfig>
  <mergehandling>
    <processmerge>
      <algorithm classname="com.idsscheer.ppm.server.
                        merger.merger.impl.ZRetaining
                        ProcessAttributesPMAlgo"/>
    </processmerge>
    ...
  </mergehandling>
</mergerconfig>
...
```

The Java class used is a component of the standard PPM installation.

## 7.2.2 Hierarchy key rules

Hierarchy key rules assign process instances to a higher-level function and are used to create process hierarchies (see chapter **Process hierarchies** (page 19)). They can be shown as process hierarchy keys in the detailed view of the process instance.

The hierarchy key rules are applied to all imported process instances.

**Example**

For the functions with the identifiers **FCT_ANGEBOT_ERSTELLEN**, **FCT_AUFTR_ANLEGEN** and **FCT_RECHNG_ERSTELLEN** a hierarchy key rule is generated that creates a hierarchy key from the values of the **AT_AUFTRAGSNUMMER**, **AT_RECHNUNGSNUMMER** and **AT_ANGEBOTSSNUMMER** attributes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyrules SYSTEM "keyrules.dtd">
<keyrules>
  ...
  <hierarchykeyrule>
    <refobjects>
      <refobject objecttype="OT_FUNC">
        <objectname name="FCT_ANGEBOT_ERSTELLEN"/>
        <objectname name="FCT_AUFTR_ANLEGEN"/>
        <objectname name="FCT_RECHNG_ERSTELLEN"/>
      </refobject>
    </refobjects>
    <keyparts>
      <keypart attributetype="AT_AUFTRAGSNUMMER"/>
      <keypart attributetype="AT_RECHNUNGSNUMMER"/>
      <keypart attributetype="AT_ANGEBOTSSNUMMER"/>
```

```
    </keyparts>
  </hierarchykeyrule>
  ...
</keyrules>
```

## 7.2.3 Shared fragment key rules

Shared fragment keys assign shared fragments to process instance fragments. Shared fragments are special process fragments, which contain exclusively functions involved in several process instances. As these functions are only executed once in the source system but occur in several process instances, they are called shared functions.

Shared fragments are imported in graph format using the XML import. The definition of the graph for a shared fragment contains the **AT_IS_SHARED_FRAGMENT** process instance attribute with the value **TRUE**. All functions of a shared fragment must be identified as shared functions by the value **TRUE** for the **AT_IS_SHARED_FUNCTION** function attribute. During importing, at least one shared fragment key is calculated for each imported shared fragment. Shared fragments for which no key can be calculated are not imported.

The shared fragment key rules are applied to all imported process instance fragments. During subsequent processing (rumppmimport) all fragments (shared fragments and process instance fragments) for which identical shared fragment keys have been calculated are written to a process instance and then linked using the merge rules.

The shared fragment key copies the shared fragments to a process instance. To link shared fragments with one another or with normal process instance fragments, appropriate rules are specified, which depend on the merge procedure used.

You can use a shared fragment key only once for copying a process fragment to a process instance. After the first use, the key is removed from the process instance. Therefore, copies of shared fragments are not updated in a process instance when a shared fragment changes.

### Example

A shared fragment key rule is created for the events with the identifiers **EVT_ACE** and **EVT_GIK**, which generates a shared fragment key from the value of the **AT_XYZ** attribute.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyrules SYSTEM "keyrules.dtd">
<keyrules>
  ...
  <sharedfragmentkeyrule>
    <refobjects>
      <refobject objecttype="OT_EVT">
        <objectname name="EVT_ACE"/>
        <objectname name="EVT_GIK"/>
      </refobject>
    </refobjects>
    <keyparts>
      <keypart attributetype="AT_XYZ"/>
    </keyparts>
  </sharedfragmentkeyrule>
  ...
</keyrules>
```

## 7.2.4 Merge key rules

Merge key rules are used for merging merge events within a process instance. In this way, the process instance fragments assigned using process keys are linked to form a process instance.

Merge keys are calculated from particular object attributes of the process instance fragment. They are used to reconstruct the process logic of the process instance and the unlinked fragments are linked accordingly.

PPM differentiates between two merge procedures:

- Key-based merge
- Merge based on sort order

The relevant merge procedure is specified in the XML configuration file ***_merger.xml**. The structure of this file is specified by the DTD **mergerconfig.dtd**.

### FILE MERGERCONFIG.DTD (PART 1)

```
<!ELEMENT mergerconfig ( mergehandling, connectorhandling? )>
<!ELEMENT mergehandling ( sharedfragmentmerge?, processmerge?, eventmerge+ )>
<!ATTLIST mergehandling eventmode (startevent | endevent | importtime) "importtime">
<!ELEMENT sharedfragmentmerge ( algorithm )>
<!ELEMENT processmerge ( algorithm?,  mergeattributes?)>
<!ATTLIST processmerge
    mode (replace|update) 'replace'>
<!ELEMENT eventmerge ( mode, condition?, algorithm? )>
<!ATTLIST eventmerge
    key ID #IMPLIED
    priority CDATA #IMPLIED>
<!ELEMENT condition EMPTY>
<!ATTLIST condition
    classname NMTOKEN #REQUIRED
    value (TRUE|FALSE) 'TRUE'
    comment CDATA #IMPLIED>
<!ELEMENT algorithm EMPTY>
<!ATTLIST algorithm
    classname NMTOKEN #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT mergeattributes (attribute+)>
<!ELEMENT attribute EMPTY>
<!ATTLIST attribute
    key CDATA #REQUIRED>
<!ELEMENT mode ( keymerge | sortmerge )>
<!ELEMENT keymerge EMPTY >
<!ELEMENT sortmerge ( criterion*, algorithm? )>
<!ELEMENT criterion EMPTY>
<!ATTLIST criterion
    name NMTOKEN #REQUIRED>
```

| XML tag | Description |
|---|---|
| mergerconfig | Grouping of merge configuration |

| XML tag | Description |
|---|---|
| mergehandling | Merge type to be configured. At least the **eventmerge** element must be specified. |
| sharedfragmentmerge (optional) | Algorithm differing from the default algorithm for merging the shared fragments with process instance fragments |
| processmerge (optional) | Algorithm differing from the default algorithm for merging the process instance fragments. Available for selection are the modes **Replace** or **Update**. The default value is **Replace**. |
| eventmerge | Algorithm differing from the default algorithm for merging the merge events |
| key (optional) | ID that can be used to reference the eventmerge element. |
| priority (optional) | Priority of the eventmerge element – the lower the integer value, the higher the priority. |
| condition (optional) | Condition for merging of merge events |
| classname | Name of JAVA class, which checks the specified condition |
| value (optional) | Condition is met if checking by the JAVA class returns the specified value (**TRUE** or **FALSE**). The default value is **TRUE**. |
| algorithm (optional) | Calculation rule (JAVA class) |
| classname | JAVA class that implements a particular calculation rule |
| mode | Merge procedure for event merging |
| keymerge | Key-based event merge based on defined merge keys |
| sortmerge | Event merge based on sort order of functions |
| criterion | Criterion (function attribute) to be used as a basis for sorting the fragments. Multiple criteria can be specified. |
| name | Name of function attribute |

## FILE MERGERCONFIG.DTD (PART 2)

```
<!ELEMENT connectorhandling ( multiindegreehandling?, multioutdegreehandling?,
andhandling?, orhandling?, xorhandling?)>
<!ELEMENT multiindegreehandling (algorithm) >
<!ELEMENT multioutdegreehandling (algorithm) >
<!ELEMENT andhandling (algorithm) >
<!ELEMENT orhandling (algorithm) >
<!ELEMENT xorhandling (algorithm) >
```

| XML tag | Description |
|---|---|
| connectorhandling (optional) | Handling of connectors in process graphs by specifying a JAVA class (**algorithm**) |
| multiindegreehandling (optional) | Algorithm, which controls the inserting of connectors with multiple incoming connections for the object merge |
| multioutdegreehandling (optional) | Algorithm, which controls the inserting of connectors with multiple outgoing connections for the object merge |
| andhandling (optional) | Algorithm for handling of AND connectors |
| orhandling (optional) | Algorithm for handling of OR connectors |
| xorhandling (optional) | Algorithm for handling of XOR connectors |

### Example

For linking process instance fragments, the key-based event merge is used.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mergerconfig SYSTEM "mergerconfig.dtd">
<mergerconfig>
  ...
  <mergehandling>
    <eventmerge>
      <mode>
        <keymerge/>
      </mode>
    </eventmerge>
  </mergehandling>
  ...
</mergerconfig>
```

During the event merge, the combined set of attributes of both merge events is copied to the remaining event. Existing object attributes are not overwritten. The first merge event imported is deleted.

### 7.2.4.1 Key-based merge

The key-based merge is used to merge events with identical merge keys. The first merge event imported is deleted and the number of identical merge keys is reduced. The merge process is repeated until no more identical merge keys are found within the current process instance.

Merge key rules are defined in the XML file **KeyRules.xml**. This is done by specifying the attributes of a fragment event involved in the creation of the merge key. The merge key itself is created by combining the specified attribute values.

#### EXAMPLE

The file extract below defines a merge key rule, which uses the internal object name of the event (**AT_OBJNAME_INTERN** attribute type) to create the merge key for the start and end event in a process instance fragment. As several fragments with the same fragment definition can occur in a process instance, the merge key is extended to include the value of the **AT_END_TIME** attribute.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyrules SYSTEM "keyrules.dtd">
<keyrules>
  ...
  <processkeyrule>
    ...
  </processkeyrule>
  ...
  <mergekeyrule>
    <refobjects>
      <refobject objecttype="OT_EVT">
        <objectname name="EVT_START"/>
        <objectname name="EVT_END"/>
      </refobject>
    </refobjects>
    <keyparts>
      <keypart attributetype="AT_OBJNAME_INTERN"/>
      <keypart attributetype="AT_END_TIME"/>
    </keyparts>
  </mergekeyrule>
  ...
</keyrules>
```

### 7.2.4.2 Merge based on sort order

The merge based on sort order merges events based on particular sort criteria. Any number of sorting criteria can be specified in the form of function attribute types. The following event for a function is merged with the predecessor event of the following function.

By default, alphanumeric and chronological sorting procedures are implemented in PPM. The method used is specified by the data type of the specified sorting criterion.
An example of a sorting criterion could be the **AT_END_TIME** function attribute with the **TIME** data type.

In a merge based on sort order, the imported process instance fragments may not contain rules. Process instance fragments with sequential functions are divided into minimal EPCs (event-function-event).

**Warning**

Make sure that the sorting criterion you defined is available at each function of the instances to be merged and includes the corresponding values.

In PPM 4.0 and above, you can use both merge methods in a client configuration.

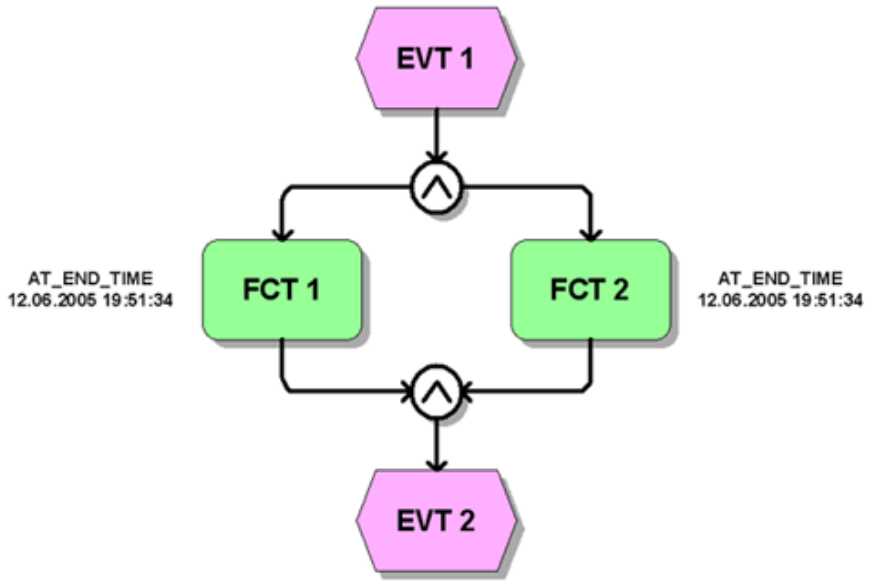
### Example

The **AT_END_TIME** function attribute is used for the merge based on sort order. The time stamp must be specified for each function of the instance.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mergerconfig SYSTEM "mergerconfig.dtd">
<mergerconfig>
  ...
  <mergehandling>
    <eventmerge>
      <mode>
        <sortmerge>
          <criterion name = "AT_END_TIME" />
        </sortmerge>
      </mode>
    </eventmerge>
  </mergehandling>
  ...
</mergerconfig>
```

If the sort attribute for several functions has the same value, AND rules are used to create parallel paths in the merged process instance.

### Example




As the sort criterion used (**AT_END_TIME**) returns an identical value for the two functions **FCT 1** and **FCT 2**, AND rules are used to create a parallel path in the merged process instance.

### 7.2.4.3 Combining merge methods

In order to be able to merge fragments using different methods, you need to define multiple merge methods. To do this, the **key** and **priority** attributes are added to the merger configuration (**eventmerge** XML element).
The value of the **key** attribute specifies the name of the merge method and is referenced by the **AT_INTERNAL_EVENT_MERGE_MODE** graph attribute for fragments to be imported. If fragments with different merge methods are merged during an import operation, the method with the lowest priority (**priority** XML element) is used.

The default merge method is used for fragments for which the **AT_INTERNAL_EVENT_MERGE_MODE** attribute is not specified. The default method is the one that is defined in the merge configuration without specifying a key.

**Warning**

Specify different priorities for all merge methods.

To specify the merge method to be used for different system event types, specify the key for the relevant merge method in the **AT_INTERNAL_EVENT_MERGE_MODE** process attribute for the fragment definition EPCs. All system events imported with this fragment definition are then automatically merged using the specified method.

**Warning**

If the merge method specified in the fragment to be imported does not exist, the fragment is not imported and an error message is displayed. The error message is also saved in the **AT_MERGE_ERROR_MESSAGE** attribute for the corresponding fragment. In addition, the value **true** is entered for the **AT_MERGE_ERROR_FLAG** attribute.

**Example**

The following merge configuration is used for the data import and defines 3 methods:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mergerconfig SYSTEM "mergerconfig.dtd">
<mergerconfig>
  <mergehandling>
    <eventmerge priority="10">
      <mode>
        <keymerge/>
      </mode>
    </eventmerge>
    <eventmerge key="SORTMERGE_ID" priority="3">
      <mode>
        <sortmerge>
          <criterion name = "AT_ID" />
        </sortmerge>
      </mode>
    </eventmerge>
    <eventmerge key="SORTMERGE_DATE" priority="4">
      <mode>
        <sortmerge>
          <criterion name = "AT_START_TIME" />
        </sortmerge>
      </mode>
    </eventmerge>
```

```
  </mergehandling>
</mergerconfig>
```

Fragments without the **AT_INTERNAL_EVENT_MERGE_MODE** attribute are merged using the key-based merge method. Fragments with the attribute value **SORTMERGE_ID** are merged based on sort order according to the **AT_ID** function attribute. Fragments with the attribute value **SORTMEGRE_DATE** are merged based on sort order according to the **AT_START_TIME** function attribute.

## 7.2.5 Object key rules

Object key rules are used when re-importing data to identify and overwrite identical objects. Two event or function objects are identical if the same object key has been calculated for them. If objects are identified as being identical, the last object imported overwrites the previously imported object. The process logic of the process instance is modified accordingly. The calculated object key is written to the corresponding object as the **AT_INTERNAL_OBJECT_KEY** attribute type.

### Example

For all functions, the object key is created from the values of the **AT_OBJNAME_INTERN** and **AT_END_TIME** attributes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyrules SYSTEM "keyrules.dtd">
<keyrules>
  ...
  <processkeyrule>
    ...
  </processkeyrule>
  ...
  <internalobjectkeyrule>
       <refobjects>
         <refobject objecttype = "OT_FUNC" />
       </refobjects>
       <keyparts>
         <keypart attributetype="AT_OBJNAME_INTERN"/>
         <keypart attributetype="AT_END_TIME"/>
       </keyparts>
  </internalobjectkeyrule>
  ...
</keyrules>
```

**Warning**

When creating the object key rules, make sure that different object keys are calculated for different object types (function or event). Overwriting objects of different types leads to undefined results.

## 7.2.6 Output behavior of messages

For the **processkeyrule**, **hierarchykeyrule**, **mergekeyrule** and **sharedfragmentkeyrule** rules, you can influence the output behavior of system messages using the **onmissingkeypart** XML attribute, if the sub-key specified in the **keyparts** XML element cannot be calculated. The attribute can have one of the values **info**, **warning** or **ignore**, the default value is **warning**.

| Attribute value | Description |
|---|---|
| **info** | The message is output as information. |
| **warning** | The message is output as a warning. |
| **ignore** | No message is output. |

**Example**

If you are using the predecessor merge method, the information for calculating the key for the preceding fragment to the first fragment in the process instance is normally missing. To suppress the expected messages, specify the value **ignore** for the **onmissingkeypart** XML attribute.

## 7.2.7 Configuration file

The XML configuration file contains all key rules. A rule consists of a list of object attribute names, whose values are used to create the keys. By default, a key is created by combining the values of the specified attribute types. An alternative type of processing can be set by using a different Java class (see below).

Process and hierarchy keys are saved in the database.

Shared fragment keys are also attribute types of the process instances and are saved in the database.

Merge and object keys are written to the corresponding object as an object attribute. Merge keys are saved in the **AT_MERGE_KEY_1** to **AT_MERGE_KEY_10** attribute types of the referenced object, while the object key is saved in the **AT_INTERNAL_OBJECTKEY** attribute type.

## FILE KEYRULES.DTD (PART 1):

```
<!ELEMENT keyrules ((processkeyrule |
                     hierarchykeyrule |
                     mergekeyrule |
                     internalobjectkeyrule |
                     sharedfragmentkeyrule)*)>
  <!ELEMENT processkeyrule    (refobjects, condition?,
                               algorithm?, keyparts)>
  <!ELEMENT hierarchykeyrule (refobjects, condition?,
                               algorithm?, keyparts)>
  <!ELEMENT mergekeyrule      (refobjects, condition?,
                               algorithm?, keyparts)>
  <!ELEMENT internalobjectkeyrule (refobjects,condition?,
                               algorithm?, keyparts)>
  <!ELEMENT sharedfragmentkeyrule (refobjects,condition?,
                               algorithm?, keyparts)>
```

XML file contains a list of rule types for key rules.

Each rule type contains one **refobjects** one **keyparts** und optional element and optionally the **condition** or **algorithm** elements.

## FILE KEYRULES.DTD (PART 2):

```
<!ELEMENT refobjects (refobject+)>
  <!ELEMENT refobject (objectname*)>
  <!ATTLIST refobject objecttype (OT_ALL |
                                  OT_FUNC |
                                  OT_EVT |
                                  OT_ORG |
                                  OT_RULEAND |
                                  OT_RULEOR |
                                  OT_RULEXOR)
                                  #REQUIRED>
    <!ELEMENT objectname EMPTY>
    <!ATTLIST objectname
                  name     CDATA   #REQUIRED>
<!ELEMENT keyparts (keypart+)>
  <!ELEMENT keypart EMPTY>
  <!ATTLIST keypart
            attributetype NMTOKEN #REQUIRED
            nodetype (PROCESS | OT_ALL
                      | OT_FUNC | OT_EVT
                      | OT_ORG | OT_RULEAND
                      | OT_RULEOR | OT_RULEXOR)
                                  #IMPLIED
            objectname    CDATA   #IMPLIED>
<!ELEMENT condition EMPTY>
<!ATTLIST condition
          classname NMTOKEN #REQUIRED
          value (TRUE | FALSE) #IMPLIED>
<!ELEMENT algorithm EMPTY>
<!ATTLIST algorithm
          classname NMTOKEN #REQUIRED>
```

The **refobjects** element contains at least one **refobject** element.

A **refobject** element identifies objects to which a key rule is applied. An object type and optionally any number of object identifiers are specified.

Objects have an identifier (corresponds to the contents of the PPM attribute **AT_OBJNAME_INTERN**).

The **keyparts** element gives at least one **keypart** element.

Each **keypart** element contains a rule for creating key parts.

You specify the Java class containing the algorithm which is used to check the condition in the **classname** attribute. The specification ot the **value** attribute is optional.

You specify the name of the Java class which implements the algorithm for the treatment of keys in the **classname** attribute.

All of the rules specified in the configuration file reference the **AT_OBJNAME_INTERN** object attribute.

The **refobjects** XML element specifies a list of objects to which the relevant rule relates. In the **refobject** XML element, the **objecttype** XML attribute specifies the object type and the

**objectname** XML element specifies an object identifier. Several **objectname** XML elements can be specified.

As the key parts (**keyparts** XML element), specify the names of the attributes to be used to generate the key.

If you want to use a processing type other than combining, in the **algorithm** XML element specify the name of the Java class (**classname** XML attribute), which implements a different algorithm for processing the key attribute values.

**Warning**

The fixed attributes (**AT_MERGE_KEY_1** to **AT_MERGE_KEY_10**) are provided to store calculated merge keys. They may not be changed or assigned values from the XML import.

Keys are calculated when importing the process instance fragments. If you change the key rules for an existing process type, you need to import all process instance fragments of this process type again. Otherwise the subsequent merging of the process instances will lead to unwanted results. Changes to key rules for process instance fragments in a process for which process instances already exist in the PPM system should therefore only be made with extreme caution. Extending the rules when adding a new process type to the PPM system is not so critical if the existing process instances remain unaffected.

## 7.3 Process fragment merge

For a process merge you can select either the **Replace** or the **Update** mode.

PPM Customizing Toolkit provides two merge variants, **Replace attributes/objects (default)** and **Merge attributes/objects**, in the **Merge** component of the **Process merge** module. By default, the **Replace attributes/objects (default)** merge mode is activated.

The **Replace** and **Update** modes only affect process merge. The event merge works the same for both.

In the merger configuration (page 24) (mergerconfig.dtd), you can specify for each PPM client whether attributes are to be replaced or merged during a merge.

### 7.3.1 Merge mode "Replace"

In **Replace** merge mode, the process attributes of the newer (last imported) process instance (except for the merge attributes) are transferred to the resulting fragment. For identical functions and events (identical internal object key), the last imported (newer) object replaces the older object including all attributes. Only object attributes and organizational units of the newer (last imported) object are retained. The **AT_ORIG_EPK_ID** determines which object is newer.

As an option, you can specify a list of process attributes to be transferred from an older process instance to the resulting fragment when merging fragments. The process attribute of the previously imported fragment or the existing process instance is overwritten with the process attribute of the fragment imported later.

In the **mergeattributes** XML element in the merge configuration, specify a list of process attributes to be transferred when merging fragments. All other process attributes are ignored.

**Example**

The merge configuration file extract below causes the **AT_SAPSYSTEM** and **AT_SAPCLIENT** process attributes to the resulting fragment or an existing process instance when merging fragment instances.

```
<mergerconfig>
  <mergehandling>
    <processmerge>
      <mergeattributes>
        <attribute key = "AT_SAPSYSTEM"/>
        <attribute key = "AT_SAPCLIENT"/>
      </mergeattributes>
    </processmerge>
...
  </mergehandling>
</mergerconfig>
```

**Tip**

Transferring process attributes enables you to directly overwrite dimension values based on process attributes by importing a fragment that contains only the process attribute with a new value for which the dimension has been created.

Please remember that existing process attributes will be overwritten when you copy object attributes (see Attribute copy rules chapter) at a later time.

If the process attribute is specified for multiple fragments with the same process key, and the import of all fragments is split over several import operations, it is not possible to ensure that the attribute value of the last fragment imported will be transferred to the resulting fragment.

## 7.3.2 Merge mode "Update"

In **Update** merge mode the process attributes of the newer (last imported) process instance are merged with the process attributes of the older process instance. The same applies to functions with function attributes and associated organizational units.

### TIME OF IMPORT AT THE ATTRIBUTE LEVEL

The time of import is the factor in **Update** mode which determines which object is newer. The time of import is recorded for process instances, functions, events, organizational units (each as AT_ORIG_EPK_ID), and at the attribute level. After import, each attribute's time of import is known. The time of import is written to the imported EPC during XML import or process import.

If the time of import is unknown for an attribute (for example, for inventory data imported in **Replace** merge mode) the time of import of the object (function, process, etc.) that the attribute is assigned to is used.

### ADDITIVE MERGE AT THE FUNCTION LEVEL

If two functions with identical internal object keys are identified during the merge they will be merged as follows.

1. The newer function (last imported) including its attributes and organizational units will be transferred to the merged process instance.
2. All attributes of the old function are copied to the new function. If an attribute exists at both functions the attribute of the old function will be copied if it is newer.
3. All organizational units of the old and the new function will be merged. The following chapter describes the merging of organizational units.

## ORGANIZATIONAL UNITS

When merging two functions all organizational units of the old function are copied to the new function. If the same organizational unit exists at both functions the newer organizational unit including its associated connection and attributes will be retained. In this case, attributes of the connection assigned to the older organizational unit and attributes of the older organizational unit will not be transferred.

### EQUALITY OF ORGANIZATIONAL UNITS

1. During the merge, the **AT_OBJNAME** attribute determines if two non-anonymized organizational units are identical.

   For an anonymized organizational unit (that is, its original object name was changed) to be identified as identical during a reimport, an internal object key **AT_INTERNAL_OBJECT_KEY** must exist at the organizational unit. The object key is defined using object key rules.
2. Anonymized and non-anonymized organizational units are identical when the object key **AT_INTERNAL_OBJECT_KEY** matches.
3. Two anonymized organizational units are identical when the object keys **AT_OBJNAME** and **AT_INTERNAL_OBJECT_KEY** match.

### TIME OF IMPORT OF THE ORGANIZATIONAL UNIT

If no **AT_ORIG_EPK_ID** key is defined for an organizational unit the **AT_ORIG_EPK_ID** key of the associated function is used to determine the time of import of the organizational unit.

## ADDITIVE MERGE AT THE PROCESS LEVEL

During the merge, all attributes of the newer and older process instance are copied to the resulting fragment. If an attribute exists at both process instances the newer attribute is transferred.

## ADDITIVE MERGE AT THE EVENT LEVEL

If two events with identical internal object keys are identified during the merge they will be handled like in **Replace** mode. This means that the newer event replaces the older event and that attributes of the older event are not transferred to the newer event.

## CONFIGURATION

The DTD mergerconfig.dtd (page 24) contains the **Replace** and **Update** modes for configuration.

```
<!ELEMENT processmerge ( algorithm?,  mergeattributes?)>
<!ATTLIST processmerge
      mode (replace|update) 'replace'
>
```

The **mode** attribute is optional, and if it is missing the **Replace** mode is applied by default.

In **Update** mode, merge attributes (mergeattributes) are not evaluated. If a configuration containing (non-empty) merge attributes is imported with the **Update mode** attribute a corresponding message is output.

### CHANGE THE MERGE MODE

You can change the merge mode anytime via the merge configuration. This means that you can switch existing clients in **Replace** mode to **Update** mode.

The list of merge attributes is not used in **Update** merge mode.

### SHARED FRAGMENT

For the merge of two shared fragments in **Update** mode when using the default algorithm (ZDefaultSharedFragmentMergeAlgorithm) the same merge mode is automatically used that is also specified in the merge configuration for the merge of two normal fragments.

### SPECIAL ATTRIBUTES

Special attributes (for example, internal PPM attributes or attributes such as AT_IS_SHARED_FUNCTION) in **Update** mode are handled like all other attributes.

### PROCESS TYPIFICATION

To transfer process type information directly from the source system in **Update** mode (without typification rules), the attributes **AT_INTERNAL_PROCTYPE** and **AT_INTERNAL_PROCTYPEGROUP** including typification information (process type and process type group) must be specified.

### CALCULATED ATTRIBUTES

Calculated attributes at the process or at functions are also copied by the **Update** mode.

If a calculated attribute of the old function is copied to the new function during the merge of two functions, and if that attribute is not calculated later, the older, calculated value would exist at the merged function.

If you wish to turn off this behavior you need to set the parameter **calcattr delete=yes** for the calculation rule. If this parameter is set attributes that cannot be calculated will be deleted later.

## 7.4 Merge events

### 7.4.1 Parallel paths with multi-valued keys

If you want to merge parallel process paths again when using a key-based merge, you must calculate several merge keys for the end events of the preceding process fragments for the start event of the merging fragment. You specify a multi-valued attribute and a separator for this purpose in the **multikey** XML element. Multi-valued means that the value of the specified attribute is split into several parts using a separator. A merge key is calculated for each part.

#### Example

The fragments of a process instance are linked by the **THIS_KEY** and **PREV_KEY** merge attributes. The key for the predecessor is saved in the **PREV_KEY** attribute. If a system event has several predecessors, each key for the predecessors is written to the log file multiple times in the **PREV_KEY** attribute. The attribute mapping used is configured in such a way that the **PREV_KEY** attribute is instantiated with several values for the merge events as **AT_KEY**.

The data extraction from your source system includes the following system event:

```
...
  <attribute type="EVENTTYP">Change customer order</attribute>
  <attribute type="PROC_ID">123456</attribute>
  <attribute type="THIS_KEY">3</attribute>
  <attribute type="PREV_KEY">1</attribute>
  <attribute type="PREV_KEY">2</attribute>
  <attribute type="USER">Team A</attribute>
...
```

The mapping file used contains the following attribute mapping for start and end events:

```
...
<!-- mapping startevents -->
    <attribute ppmattributetype="AT_KEY">
      <multieventattributetype delimiter=";">PREV_KEY</multieventattributetype>
    </attribute>
<!-- mapping endevents -->
    <attribute ppmattributetype="AT_KEY">
      <eventattributetype>THIS_KEY</eventattributetype>
    </attribute>
...
```

The active merge configuration contains the following rule:

```
...
<mergekeyrule>
  <refobjects>
    <refobject objecttype="OT_EVT"></refobject>
  </refobjects>
  <keyparts>
    <multikey attributetype="AT_KEY" delimiter=";" />
  </keyparts>
</mergekeyrule>
...
```

In the fragment whose start event has a **THIS_KEY** attribute with the value **3**, the preceding fragments whose end events have a **THIS_KEY** attribute with the value **1** or **2** are merged.

After merging fragments, the attributes are transferred unchanged, that is, the resulting set of sub-keys is not written back to the merge attribute for the remaining event in consolidated form.

**Example**

For example, if you are using rules that calculate a merge key from the multi-valued **AT_KEY** attribute, and **AT_KEY** has the value **x;y** at the system event **A** and the value **y;z** at the system event **B**, the **AT_KEY** attribute has the value of the attribute of the system event **B**, assuming that the system event **B** was imported later. Merging of fragments is unaffected by this, as fragments are merged using merge keys that are already calculated during the import.

## 7.4.2 Merge mode

When merging the merge events, you can optionally specify which of the merge events will be transferred to the resulting fragment using the **eventmode** attribute for the **mergehandling** XML element in the merger configuration. Valid values are **STARTEVENT**, **ENDEVENT** and **IMPORTTIME**, with a default value of **IMPORTTIME**.

The following event types exist:

| Type | Description |
|---|---|
| Start event | A standard event has no predecessor (outgoing connection only). |
| Coupling event | A coupling event has both predecessors and successors (incoming and outgoing connection). |
| End event | An end event has no successors (incoming connection only). |

### KEY-BASED MERGE

When using a key-based merge, the behavior when merging merge events is as follows:

| eventmode | Description |
|---|---|
| IMPORTTIME | The event imported later is transferred regardless of the event type.<br>Default value |
| STARTEVENT | The event imported later that is not an end event is transferred. An end event is only transferred if two end events are being merged. |
| ENDEVENT | The event imported later that is not an start event is transferred. A start event is only transferred if two start events are being merged. |

### MERGE BASED ON SORT ORDER

When using a merge based on sort order, the selected event types are given priority directly when merging as the process instance is always broken down into individual fragments of the form Event-Function-Event before the merge. If two merge events of the same type are being merged, the one imported later is transferred.

## 7.5 Attribute copy rules

Process instance attributes are required to calculate instance-related measures and to create dimensions. When importing data in PPM event format, attributes of the process instance fragments cannot be imported directly because the instance fragment is created dynamically from a fragment definition. Object attributes of the instanced process fragment are therefore copied to the process instance.

The rules for copying object attributes to the process instance are made up of the following sections:

- List of attributes to be copied. The specified attribute copy rule is used for each attribute type in the list.
- Source object type of objects whose attribute type is to be copied
- Prioritized list of objects (AT_OBJNAME_INTERN object attribute) whose attribute type is to be copied. The list of objects is processed from top to bottom. As soon as it was possible to copy the attribute type, the next copy rule is processed.

If the attribute type is not specified for any of the objects indicated or for the process instance, the attribute type is created with the default value specified in the **#PCDATA** section of the **attributspec** XML element.

The following example copies the **AT_ID** attribute for the **FCT_Create_order** function to the process instance. If the attribute cannot be accessed as it is not specified or the function does not exist, the attribute for the next object indicated, **FCT_Create_invoice** is copied.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE copyattributerules SYSTEM "copyattributerules.dtd">
<copyattributerules>
   ...
   <copyattributerule>
      <attributespec srcattrname="AT_ID"/>
      ...
      <sourceobjectspec nodetype="OT_FUNC">
         <objectname name="FCT_Create_order"/>
         <objectname name="FCT_Create_invoice"/>
         ...
      </sourceobjectspec>
   </copyattributerule>
   ...
</copyattributerules>
```

The DTD **CopyAttributeRules.dtd** describes the structure of the XML file for the attribute copy rules:

```
<!ELEMENT copyattributerules (copyattributerule*)>

  <!ELEMENT copyattributerule (attributespec+,
                               sourceobjectspec)>

    <!ELEMENT attributespec (#PCDATA)>
    <!ATTLIST attributespec
              srcattrname  NMTOKEN #REQUIRED
              destattrname NMTOKEN #IMPLIED >

    <!ELEMENT sourceobjectspec (objectname+)>
    <!ATTLIST sourceobjectspec
              nodetype (OT_FUNC |
                        OT_EVT |
                        OT_ORG |
                        OT_RULEAND |
                        OT_RULEOR |
                        OT_RULEXOR ) #REQUIRED >

      <!ELEMENT objectname EMPTY>
      <!ATTLIST objectname
                name     CDATA   #REQUIRED>
```

XML file contains list of attribute copy rules.

Each rule contains at least one **attributespec** element and exactly one **sourceobjectspec** element.

The **attributespec** element posesses the **srcattrname** attribute (source attribute) and optionally the **destattrname** attribute (target attribute).

List of source objects with at least one **objectname** element.
Each source object is specified by entering its name (object attribute **AT_OBJNAME_INTERN**).
The object type is specifed in the **nodetype** attribute.

Name of the source object which has the attribute that is copied.

All of the source object names specified in the configuration file reference the **AT_OBJNAME_INTERN** attribute.

The copying of object attributes also allows you to transfer process type information directly from the source system when using PPM event format. The attributes corresponding to the process type and the process type group are written to the process instance objects that occur in each process instance by mapping as the **AT_PROCTYPE** and **AT_PROCTYPEGROUP** attributes and are copied to the process instance using the attribute copy rules.

## 7.6 Anonymizing

It can be useful not to display the names of the processors involved in executing a function, for example, for data protection reasons. After initializing the PPM client database, in the PPM front-end you can use the **Organizational units** administration component to specify how the names of the processors occurring in the instance data are replaced by the names of organizational units (anonymized).

When aggregating process instances, the information on the processor of functions is lost. To transfer information about the processors into an aggregated EPC, they must be anonymized.

To do this, you create PPM organizational units and assign all relevant processors to organizational units. When importing date, the names of the processors are replaced by the names of the corresponding organizational unit.

An organizational unit is defined by the following properties in the PPM system:

| User interface item | Description |
|---|---|
| Name | Name of the organizational unit (freely selectable) |

| User interface item | Description |
|---|---|
| Processor | List of processors that are assigned to the selected organizational unit |
| Cost rate | Cost of a member of staff from an organizational unit per unit of time. The cost rate affects the calculation of certain measures in the process cost calculation (see chapter **Definition of process cost measures** (page 140)). |
| Ignore during measure calculation | The selected organizational unit is not included in the measure calculation. This may be specified, for example, when processors perform batch processing functions (so-called batch users). |
| All non-assigned processors to this organizational unit | Processors that are not assigned to an organizational unit are anonymized using the name of this organizational unit.<br>You must define a default organizational unit in order to be able to save the configuration. The organizational unit must have at least one processor. |

You can use the **runppmconfig** command line program to export organizational units to an XML file that you can also modify manually. For further information about the command line program, please refer to the **PPM Operation Guide**.

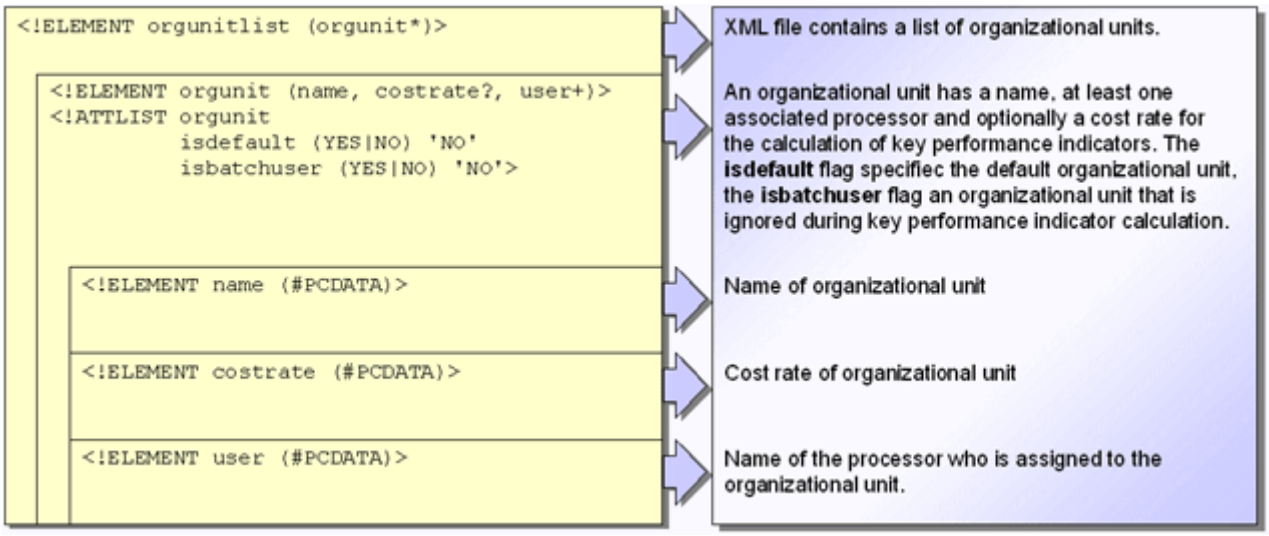
### Example

**Mr Brown** and **Mrs Smith** work in the Order acceptance department, **Mr Miller** in the Accounting department. When importing data, the processors **Mr Brown** and **Mrs Smith** are replaced by the name of the Order acceptance organizational unit, while **Mr Miller** is replaced by the name Accounting, while all other processors are replaced by the name of the default organizational unit, **Not specified**. The specified cost rates are used by the Measure calculator for process cost analysis.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE orgunitlist SYSTEM "orgunits.dtd">
<orgunitlist>
  <orgunit isdefault="NO" isbatchuser="NO">
    <name>ORDER ACCEPTANCE</name>
    <costrate>15.75 EUR_PER_HOUR</costrate>
    <user>Mrs Smith</user>
    <user>Mr Brown</user>
  </orgunit>
  <orgunit isdefault="NO" isbatchuser="NO">
    <name>ACCOUNTING</name>
    <costrate>25.3 EUR_PER_HOUR</costrate>
    <user>Mr Miller</user>
  </orgunit>
```

```
  <orgunit isdefault="YES" isbatchuser="NO">
    <name>Not specified</name>
    <costrate>24 EUR_PER_HOUR</costrate>
    <user>DUMMY</user>
  </orgunit>
</orgunitlist>
```

The document type definition **orgunits.dtd** for this XML file looks like this:




You can import the XML file created when initializing the database by specifying the file name, so that anonymizing rules are available immediately.

# 8 Process typification

Automatic assignment of process instances to a particular process type is done using a typification rule that is registered in the process tree configuration at the corresponding process type with its unique internal name.

The typification rule is defined as a calculation function in the measure configuration. You can define a maximum of one typification rule per process type. See Create typification rules (page 43) for details.

Alternatively, the process typification can be done by importing values in specific attributes, the so-called "pretypification". See Typification by attribute calculation (page 131) for details.

## 8.1 Create typification rules

The chapters below describe the two steps for creating a typification rule:

- Definition of a typification rule in the measure configuration
- Use of a typification rule in the process tree configuration

Use PPM Customizing Toolkit to create typification rules. This allows you to create rules easily and avoid sources of errors, particularly with more complex calculation rules for typification rules (see chapter **Typification rules in CTK** (page 131)). The changes are imported into the PPM system by activating the configuration.

### 8.1.1 Measure configuration

The typification rules are defined based on attribute calculations in the measure configuration. The calculation rule is specified in the **function** XML element.

| XML tag | Description |
|---|---|
| function name | Name of the typification rule – referenced by the **typifierrule function** XML element for the process tree configuration. |
| resulttype | Result value. Must be of the BOOLEAN type. |
| datatype | Data type. Must be of the BOOLEAN type. |

A typification rule for the above example could look like this:

**Example**

Extract from file **Keyindicator.xml**

```
...
<function name="typifierrule_OrderProcessing_StandardOrder"
          resulttype="BOOLEAN" datatype="BOOLEAN">
  <in>
    <constant>
      <dataitem>
        C
        <datatype name="TEXT">Text</datatype>
```

```
      </dataitem>
    </constant>
    <attribute name="AT_SAP_VBTYP" nodetype="OT_FUNC">
      <in>
        <constant>
          <dataitem>
            SO
            <datatype name="TEXT">Text</datatype>
          </dataitem>
        </constant>
        <attribute name="AT_SAP_VKBELEGART"
               nodetype="OT_FUNC" objectname="this"/>
        </in>
      </attribute>
    </in>
</function>
...
```

Calling up the **typifierrule_OrderProcessing_StandardOrder** function (typification rule) checks whether the process instance to be typified is typified as **Order processing\Standard order** (return value = **true**) or not (return value = **false**).
The process instance is assigned to the **Standard order** process type under the following condition: The process instance must contain at least one function that includes the attribute type combination **AT_SAP_VBTYP** with the value **C** and **AT_SAP_VKBELEGART** with the value **SO**.

## 8.1.2 Process tree configuration

The typification rules defined previously in the file **Keyindicator.xml** must now be assigned to the individual process types in the file **Processtree.xml**. Only one typification rule can be specified for each process type. The **typifierrule** XML element is optional. If no typification rule is specified for a process type, it is ignored during typification.

You must specify the corresponding information for the **function** and **priority** attributes. The **function** attribute is used to specify the name of the typification rule you want to use for this process type, taken from the measure configuration.

**Example (extract from process tree configuration)**

```
<processtree name="Processes">
  <processtypegroup name="OrderProcessing">
  ...
    <processtype name="StandardOrder" autovisible="TRUE">
      <typifierrule function=
           "typifierrule_OrderProcessing_StandardOrder"
                                    priority="0"/>
      <processparamset>
        ...
      </processparamset>
      ...
      <functionparamset>
        ...
      </functionparamset>
      ...
      <useki name="..." assessment="..."/>
        ...
      <usedim name="..."/>
        ...
```

```
    </processtype>
    ...
  </processtypegroup>
  ...
</processtree>
```

#### 8.1.2.1 Prioritization

A process instance is always assigned to a single process type. If several typification rules apply to a process instance, the **priority** integer attribute is used to specify which process type the process instance is ultimately assigned to. A rule with priority **0** has the highest priority and is prioritized by the typifier.

If, for example, three typification rules with the priorities **2**, **3** and **6** apply to the process instance, it is typified with a priority of **2** in line with the typification rule. The typification rules **3** and **6** are ignored for this process instance. The typifier first of all checks whether a typification rule with priority **0** applies to the process instance to be typified. If not, a rule checks whether the next priority level applies and so on.
As soon as a rule applies, processing of the typification rules is ended and the process instance is assigned to the corresponding process type.

You specify the prioritization of typification rules in the **Processes** CTK module using the button **Prioritize typification rules** on the selected typification rule.

### 8.1.3 Definition of attribute calculations

To calculate a measure or create a dimension, either the value of an existing attribute is used or the algorithm for calculation of the attribute is specified in the XML configuration file (**calcattr** XML element). The specified attribute name must be contained in the imported attribute definition for the PPM system (files **AttributeTypes.xml** and **AttributeNames.xml**). The data type and base unit are defined by the attribute definition. The calculation of an attribute is always made in the base unit of the attribute type. The result is also saved as a value in the base unit.

Attributes are only valid within a process instance. It is not possible to calculate attributes for other process instances or to include them in the calculation.

The calculation of an attribute can be made dependent on other attributes, which can themselves be calculated attributes. All attributes specified with the **depends** XML element are calculated before the calculation of the current attribute is executed. Cyclic dependencies are detected during import of the measure configuration and acknowledged by an error message.

A default value can optionally be specified (**defaultvalue** XML element), which can be assigned to the attribute value if the calculation could not be successfully carried out. The default value must always be specified with a unit that is permissible for the attribute data type. This is the only way for the value in the base unit to be calculated correctly.

If an attribute cannot be calculated and no default value is specified, the attribute is not created for the process instance or the process instance objects and a corresponding message is output.

The XML structure for definition of an attribute calculation looks like this:

```
...
<calcattr name="..." type="..." objectname="..."
                           scale="..." delete="...">
  <depends attrname="..." type="..."/>
  <defaultvalue>..."</defaultvalue>
```

Either

```
  <calculation> ... </calculation>
</calcattr>
...
```

or

```
  <calcclass name="..."/>
</calcattr>
...
```

| XML tag | Description |
|---|---|
| **name** | Internal name of the attribute to be calculated.<br>The attribute is created for all object types specified by **type** in the process instance currently being processed. Any existing attribute is overwritten. |
| **type** | Object type to which the attribute is written<br>**PROCESS**: Calculated attribute is written to the process instance.<br>**OT_FUNC**: Calculated attribute is written to all functions in the process instance.<br>**OT_ORG**: Calculated attribute is written to all organizational units in the process instance.<br>**OT_EVT**: Calculated attribute is written to all events in the process instance.<br>**RELATION**: Calculated attribute is written to the relation in the process instance that is specified using **relname**. |
| **relname**<br>(optional) | Only for **type="RELATION"**. Specifies the relation to which the attribute calculation is to relate. To be used instead of dependsrel. |
| **objectname**<br>(optional) | Internal name of the function (**AT_OBJNAME_INTERN** object attribute) to which the attribute is written. This option may only be used for the calculation of function attributes. Multiple object names are specified separated by commas, the placeholders ***** and **?** can be used as required in the object name. |

| XML tag | Description |
|---|---|
| **scale**<br>(optional) | The result of a calculation rule is written to the attribute in the specified scale.<br>The scale value depends on the data type of the attribute on which the calculation is based.<br>If nothing is specified, the result is output in the base unit for the attribute data type. |
| **delete** | If the value is yes a previously calculated attribute value is deleted before the new calculation (for example, specified for the definition of critical time attributes used by the Early alert system, see **Time dimensions for the Early alert system** (page 176) chapter). Default value: **no** |

To selectively specify a calculation rule for one or more particular functions, specify the name of the corresponding function in the **objectname** XML attribute.

### Example

The calculation rule is only executed for functions whose internal names match the specified pattern.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_END_TIME" type="OT_FUNC"
                          objectname="FCT_AUFT??_*,
                                     FCT_END_*_?,
                         *_AUFTRAG,FCT_AUFTR_START">
    <calculation>
    ...
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

The definition of a calculation rule is completed by specifying the following XML elements:

| XML element | Description |
|---|---|
| **depends** (optional) | Name and type of an attribute (**PROCESS**, **OT_FUNC**, **OT_EVT**, **OT_ORG**, or **RELATION**), which must exist for the calculation to be executed. If the specified attribute is a calculated attribute, this is calculated first. The **relname** attribute specifies the relation on which there is a dependency (only for **type="RELATION"**).<br>Several **depends** elements can be specified simultaneously.<br>Not to be used in conjunction with **dependsrel**. |
| **dependsrel** (optional) | Name of the relation on which there is a dependency (only for **type="RELATION"**). To be used instead of **relname**.<br>Not to be used in conjunction with **depends**. |
| **defaultvalue** (optional) | Default value of the attribute if the attribute cannot be calculated for whatever reason. |
| **calcclass**<br>**calculation** | Unique specification of the calculation rule using one of the two XML elements.<br>You use **calcclass** to specify the algorithm by entering a complete Java class path. Using **calculation** specifies a calculation formula directly in XML notation. |
| **calcparam** (optional) | Only for **calcclass**. Transfers any number of parameters (calcparam) when calling up a calculation class. The unique internal name of the parameter is defined using **key** and the corresponding parameter value using **value**. |

For calculating an attribute using the **calculation** XML element, a comprehensive set of rules for the definition of calculation rules is available.

### Example

The example below shows the XML definition of the calculation of the processing time for a process instance. The processing time is defined as the difference between the earliest start time and the latest end time of all functions in a process instance. To store the measure value, the **AT_KI_PROCESSTIME** attribute (type: time span) is selected. The calculation is only to be carried out if the two attributes **AT_START_TIME** and **AT_END_TIME** are specified for at least one function of the process instance. This does not necessarily have to be the same function. If the calculation fails for any reason, the result attribute is assigned the default value of **0 SECOND**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_KI_PROCESSTIME" type="PROCESS">
    <depends attrname="AT_END_TIME" nodetype="OT_FUNC">
    <depends attrname="AT_START_TIME" nodetype="OT_FUNC">
    <defaultvalue>0 SECOND</defaultvalue>
    <calculation>
      <timespan>
        <filteredattribute name="AT_END_TIME"
                   nodetype="OT_FUNC" filter="LATEST"/>
        <filteredattribute name="AT_START_TIME"
                    nodetype="OT_FUNC" filter="EARLY"/>
      </timespan>
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

A calculated attribute always contains the result value and the result unit. The result unit is always specified in the base unit corresponding to that in the data type of the event attribute.

### 8.1.3.1 Calculation classes

This chapter describes all of the calculation rules contained in PPM, which can be specified as a class name for calculation of an attribute using the **calcclass** XML element. The class name must be specified with the Java package structure path.

**Example**

```
<calcattr name="AT_KI_FDLZWK" type="OT_FUNC">
  <calcclass name="com.idsscheer.ppm.server.keyindicator.
       attributecalculator.ZAttributeCalculatorFDLZWK"/>
</calcattr>
```

The fixed part of the class name is omitted below. Instead of **com.idsscheer.ppm.server.keyindicator.attributecalculator.ZAttributeCalculatorFDLZWK**,
**ZAttributeCalculatorFDLZWK** is specified.

#### 8.1.3.1.1 Log output for calculation classes

You have the option of specifying which messages are to be output for each calculation class using the **loglevel** XML attribute. Enter one of the values **SILENT**, **DEFAULT**, or **VERBOSE**.

The following table shows the default relationship between the **loglevel** XML attribute and the type of messages that are output from the calculation class. The message type is determined by the assigned logger module in the client-specific configuration file **Server_Log_settings.properties**.

| Log level | Description |
|---|---|
| **SILENT** | All log output is suppressed. |
| **DEFAULT** | Warnings and error messages are output. |
| **VERBOSE** | Information, warnings, and error messages are output. |

**Server_Log_settings.properties** file extract:

```
...
#MODULE_CALCCLASS_SILENT
log4j.logger.LOG.CCS=FATAL
#MODULE_CALCCLASS_DEFAULT
log4j.logger.LOG.CCD=WARN
#MODULE_CALCCLASS_VERBOSE
log4j.logger.LOG.CCV=INFO
...
```

**Example**

The following file extract specifies that information, warning, and error messages are not to be output for the calculation of the function cycle time.

```
...
<calcattr name="AT_KI_FDLZ" type="OT_FUNC"
                                  loglevel="SILENT">
  <calcclass name="com.idsscheer.ppm.server.keyindicator.
        attributecalculator.ZAttributeCalculatorFDLZ"/>
</calcattr>
...
```

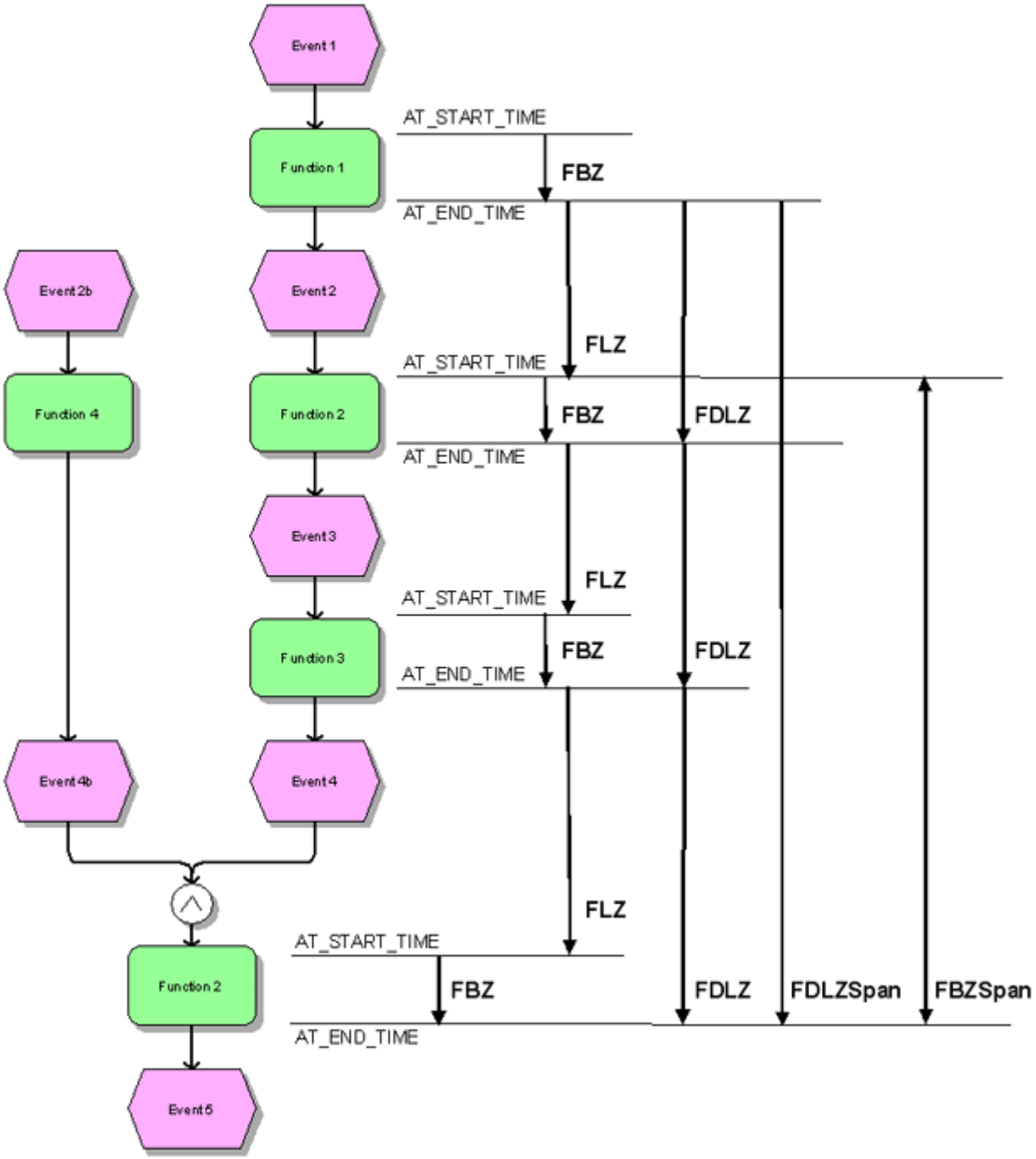
If you specify arguments in the command line to control log output, these arguments take precedence over the specifications in **Server_settings.properties**. (see chapter on **Common arguments**)

### 8.1.3.1.2 Time measures

The **AT_START_TIME** and **AT_END_TIME** function attributes are used to calculate time measures. These attributes are created by mapping a suitable source system attribute during the XML import.

### 8.1.3.1.3 Function measures

The diagram below illustrates the calculation of the function measures **Processing time** (FBZ), **Processing span** (FBZSpan), **Cycle time** (FDLZ), **Cycle time span** (FDLZSpan) and **Wait time** (FLZ):




The calculation rules also apply to relations between branching rules. For analysis of the **Cycle time** and **Wait time** measures for function 2, it is assumed that the start and end time of function 3 are after the start and end time of function 4.

The calculated measure attribute is written to the function at which the arrow in the diagram ends. Negative time differences are returned as a measure value of **0**.

When calculating the **Processing span** and **Cycle span** measures, all functions with the same name are taken into account (**AT_OBJNAME** function attribute), even if these occur within a process instance in independent process chains.

**Warning**

If only end times can be extracted from the source system, only the **Cycle time** and **Cycle span** measures can be calculated.

## FBZ

| Name | Function processing time |
|---|---|
| Type | Time span |
| Source attributes | AT_START_TIME<br>AT_END_TIME |
| Result | Difference between end time and start time of a function instance |

## FBZWK

As for **FBZ**, but based on the factory calendar.

## FBZSPAN

| Name | Function processing span |
|---|---|
| Type | Time span |
| Source attributes | AT_START_TIME<br>AT_END_TIME |
| Result | Difference between latest end time and earliest start time of all function instances with the same name |
| Note | The result is saved for each of the function instances with the same name. If a function only occurs once in a process instance, **FBZSpan** is the same as **FBZ**. |

## FBZSPANWK

As for **FBZSpan**, but based on the factory calendar.

## FDLZ

| Name | Function cycle time |
|---|---|
| Type | Time span |
| Source attributes | AT_END_TIME |
| Result | Difference between the end time of a function instance and the latest end time of its preceding function instances |

| Name | Function cycle time |
|---|---|
| Note | For merging rules, the end times of all preceding function instances are taken into account. |

## FDLZWK

As for **FDLZ**, but based on the factory calendar.

## FDLZWKBYPARAM

As for **FDLZWK**, but using any user-defined factory calendar. The corresponding configuration settings are transferred in the form of parameters when calling up the class.

**Warning**

Note that the name (**key**) of each parameter is written in upper case.

```
<calcattr name="AT_KI_FDLZWKByParam" type="OT_FUNC">
  <calcclass name="com.idsscheer.ppm.server.keyindicator.
   attributecalculator.ZAttributeCalculatorFDLZWKByParam">
  <calcparam key="FC_ATTRIBUTENAME" value="AT_FC_NAME"/>
  <calcparam key="FC_DIRECTORY" value="calculations\fc"\>
  </calcclass>
</calcattr>
```

The **AT_FC_NAME** attribute type, which contains the name of the factory calendar file to be used, is specified using the **FC_ATTRIBUTENAME** parameter. The attribute type must be specified for the function or process instance for which the calculation is executed and must appear in the data files to be imported in the form **<attribute type="AT_FC_NAME">ExampleFactoryCalendar.xml</attribute>**.

You specify the corresponding directory containing the factory calendar file to be used relative to the PPM data directory in the FC_DIRECTORY parameter. The PPM data directory data_ppm is located under <PPM installation directory>\ppm\server\bin\work\.

### Example

PPM data directory: C:\SoftwareAG\ppm\server\bin\work\data_ppm

- absolute path to the directory containing the factory calendar file: C:\SoftwareAG\ppm\server\bin\work\data_ppm\calculations\fc
- relative path to the directory containing the factory calendar file: calculations\fc.

The two parameters FC_ATTRIBUTENAME and FC_DIRECTORY must always be specified together.

The function cycle time **FDLZWKByParam** is calculated in the same way as the default calculation of the **FDLZWK** time measure, except that the factory calendar defined by the specified parameters is used for the calculation instead of the factory calendar imported into PPM by default.

Configure the calculations for the **FBZWKByParam**, **FLZWKByParam**, and **PDLZWKByParam** time measures in the same way, if you want to use custom factory calendar files to calculate the standard measures **FBZWK**, **FLZWK**, and **PDLZWK**.

## FDLZSPAN

| Name | Function cycle time span |
|---|---|
| Type | Time span |
| Source attributes | AT_END_TIME |
| Result | Difference between the latest end time of all function instances with the same name and the earliest end time of all function instances preceding those function instances |
| Note | The result is saved for each of the function instances with the same name. If a function only occurs once in a process instance, **FDLZSpan** is **0**. |

## FDLZSPANWK

As for **FDLZSpan**, but based on the factory calendar.

## FLZ

| Name | Function wait time |
|---|---|
| Type | Time span |
| Source attributes | AT_START_TIME<br>AT_END_TIME |
| Result | Difference between the start time of a function instance and the latest end time of its preceding function instances |
| Note | For merging rules, the end times of all preceding function instances are taken into account. |

## FLZWK

As for **FLZ**, but based on the factory calendar.

## 8.1.3.1.4 Process measures

### PDLZ

| Name | Process cycle time |
|---|---|
| Type | Time span |
| Source attributes | AT_START_TIME<br>AT_END_TIME |
| Result | Difference between the latest and earliest time for the end and start times of all function instances in the process instance |
| Note | This calculation method enables calculation of process cycle times even if only AT_END_TIME is specified for function instances, for example, when extracting from SAP systems. |

### PDLZWK

As for **PDLZ**, but based on the factory calendar.

### PDLZWKBYPARAM

As for PDLZWK, but taking into account the user-defined factory calendar Definition as for **FDLZWKByParam**.

## 8.1.3.1.5 Frequency measures

## 8.1.3.1.5.1 Function measures

### FEDFREQ

| | |
|---|---|
| Name | Number of executions |
| Type | Integer |
| Source attributes | AT_COUNT_PROCESSINGS |
| Result | Value of the AT_COUNT_PROCESSINGS attribute. If the attribute is not specified for the function instance, the sum of all AT_COUNT_PROCESSINGS attribute values for the connections between the function instance and the assigned organizational units is returned. |
| Note | Source attribute values less than zero are returned as 0. |

## FEDFREQSPAN

| | |
|---|---|
| Name | Number of executions (span) |
| Type | Integer |
| Source attributes | AT_COUNT_PROCESSINGS |
| Result | Sum of the values of the AT_COUNT_PROCESSINGS attribute for all function instances with the same name. If the attribute is not specified at a function instance, the sum of all AT_COUNT_PROCESSINGS attribute values for the connections between the function instance and the assigned organizational units is used instead of the attribute value. |
| Note | Source attribute values less than zero are returned as 0. |

## FFREQ

| | |
|---|---|
| Name | Function frequency |
| Type | Integer |
| Source attributes | - |
| Result | Is assigned the value 1. |
| Note | When calculating the measure, the values are added together and divided by the number of days given by the selected scaling of a time dimension, for example, (1+1)/365 days when using a time dimension with the step width Yearly. |

## FNUM

| | |
|---|---|
| Name | Number of functions |
| Type | Integer |
| Source attributes | - |
| Result | Is assigned the value 1. |

| | |
|---|---|
| Note | When calculating the measure, the values are added together and divided by the number of days given by the selected scaling of a time dimension, for example, (1+1)/365 days when using a time dimension with the step width Yearly. |

### FOEFREQ

| | |
|---|---|
| Name | Number of different users |
| Type | Integer |
| Source attributes | - |
| Result | Number of different organizational units that process a function instance. To differentiate, the AT_OBJNAME attribute of the organizational units is used. |
| Note | If no organizational units are specified at a function instance, the value 1 is returned. |

### FOEFREQB

As for **FOEFREQ**, except that the value 0 is returned if no organizational units are specified at a function instance.

## 8.1.3.1.5.2 Process measures

### PEDFREQ

| | |
|---|---|
| Name | Number of executions |
| Type | Integer |
| Source attributes | AT_COUNT_PROCESSINGS for all function instances in the process instance |
| Result | Sum of the values of the **AT_COUNT_PROCESSINGS** attribute for all function instances in the process instance. If the sum cannot be calculated, the result is **1**. |

## PFREQ

| | |
|---|---|
| Name | Process frequency |
| Type | Integer |
| Source attributes | **-** |
| Result | Is assigned the value **1**. |
| Note | When calculating the measure, the values are added together and divided by the number of days given by the selected scaling of a time dimension, for example, (1+1)/365 days when using a time dimension with the step width **Yearly**. |

## PINT, PINTB, PINTC

For function instances to which no processors are assigned, you can specify how they are to be handled when calculating the **Number of processors** measure. To calculate the **AT_KI_PINT** attribute, specify one of the following calculation classes in the measure definition:

| **<Class name>** | **Description** |
|---|---|
| ZAttributeCalculatorPINT | Each function instance without a processor assigned is handled like a function instance with a new, different processor (default setting). |
| ZAttributeCalculatorPINTb | Any function instance without a processor assigned will be ignored in measure calculation. |
| ZAttributeCalculatorPINTc | All function instances without a processor assigned are only taken into account once for the entire process instance, that is, the same processor is assumed. |

## PNUM

| | |
|---|---|
| Name | Number of processes |
| Type | Integer |
| Source attributes | **-** |

| | |
|---|---|
| Result | Is assigned the value **1**. |
| Note | For an aggregated EPC, the number of processes specifies the number of aggregated process instances. EPCs whose **Number of processes** measure is greater than 1 are EPCs for aggregated process instances. |

## 8.1.3.1.5.3 Process cost rates

The **PKSS** and **PKSR** calculation classes calculate process costs for function instances. The process costs for process instances are determined by an attribute calculator calculation rule.

### PKSS

| | |
|---|---|
| Name | Process costs based on performance standard |
| Type | Costs |
| Source attributes | AT_COSTRATE (organizational units)<br>AT_LS (function)<br>AT_COUNT_PROCESSINGS (connections between organizational units and functions) |
| Result | Average process costs for one-off execution of a function (see chapter **Definition of process cost measures** (page 140)). |
| Note | The performance standard (**AT_LS** function attribute) must be specified at the function instance. |

### PKSR

| | |
|---|---|
| Name | Process costs based on processing time |
| Type | Costs |
| Source attributes | AT_COSTRATE (organizational units)<br>AT_KI-FBZ (function)<br>AT_COUNT_PROCESSINGS (connections between organizational units and functions) |
| Result | Average process costs for one-off execution of a function (see chapter **Definition of process cost measures** (page 140)). |
| Note | The processing time (**AT_KI_FBZ** function attribute) is calculated automatically; start and end times (**AT_START_TIME** and **AT_END_TIME** function |

| | |
|---|---|
| | attributes) must be specified at the function instance. |

## 8.1.3.1.5.4 More process measures

### ISGRAPHCONNECTED

| | |
|---|---|
| Name | Linked EPC |
| Type | Logical value |
| Source attributes | - |
| Result | Returns **TRUE** if all objects in the EPC are linked to one another by a connection. |
| Note | - |

### ORIGINATOR

| | |
|---|---|
| Name | Organizational unit |
| Type | Text |
| Source attributes | AT_OBJ_NAME of organizational units |
| Result | Returns the name of the organizational units specified at the function instance, which have the same names (**AT_OBJNAME** attribute of an organizational unit). |
| Note | If no organizational units are specified for a function instance or if organizational units with different names are specified, the result is a string of length 0. |

## 8.1.3.1.5.5 Environmentally relevant calculations

### ZATTRIBUTECALCULATORTRANSFORMUNIVERSALMAPPINGBYPARAM

This calculation class calculates the value of an attribute using a mapping file based on another attribute value. The attribute value transformation can be applied to both process and object attributes.

You can control the behavior of the calculation class by specifying the following parameters:

| Parameter | Description | Example value |
|---|---|---|
| **attrname** | Source system attributes | AT_PLZ |

| Parameter | Description | Example value |
|---|---|---|
| | whose values are converted | |
| **mappingfile** | File that contains the mapping information (key-value pairs)<br>Specify the directory containing the mapping file relative to the data_ppm\bin directory. The directory is located under <PPM installation directory>\ppm\server\bin\work\. | ..\custom\<ppmclient>\xml\↵ attrtrans\PLZ_Ort.mappings |
| **defaultcopy** | Specifies the behavior if no mapping is found for the attribute value.<br>Valid values: **TRUE**, **FALSE**<br>**TRUE**: The value of the source attribute is written unchanged to the target attribute.<br>**FALSE**: The value written to the target attribute is **Not specified**. | FALSE |

Note that a default value specified in the calculation rule (**defaultvalue** XML element) is used regardless of the **defaultcopy** parameter. If a default value is specified, it takes priority.

### Example

In the example below, the values of the **AT_PLZ** process attribute are converted into the **AT_ORT** process attribute. The conversion is specified in the file **PLZ_Ort.mappings**. If the postal code does not relate to any city, the **AT_ORT** attribute is assigned the value **Not specified**.

#### EXTRACT FROM THE MEASURE CONFIGURATION

```
...
<calcattr name="AT_ORT" type="PROCESS">
  <defaultvalue>Not specified</defaultvalue>
  <calcclass name="com.idsscheer.ppm.server.
                keyindicator.attributecalculator
     ZAttributeCalculatorTransformUniversalMappingByParam">
    <calcparam key="attrname" value="AT_PLZ"/>
    <calcparam key="mappingfile" value="PLZ_Ort.mappings"/>
  </calcclass>
</calcattr>
```

...

Content of the mapping file **PLZ_Ort.mappings**:

```
66115 = Saarbruecken
10117 = Berlin Center
14612 = Falkensee
```

Please make sure that the mapping files you are using are regular Java property files. If they contain umlauts or other special characters, they must be converted with **native2ascii**. Further information on converting files with native content to ASCII files is available on the help pages on the Sun Microsystems, Inc. Web page.

## OBJECTCOUNTERBYEPCENV

The **ObjectCounterByEpcEnv** calculation class calculates the number of functions preceding or following the current function, ignoring any events and connectors. The current function is the function for which the calculation rule is being executed.

You can control the behavior of the calculation class by specifying the following parameters:

| Parameter | Description | Example value |
|---|---|---|
| **DIRECTION**<br>(one value) | Direction of search for predecessor or successor functions starting from the current function | **FORWARD** (successor functions) or<br>**BACKWARD** (predecessor functions) in relation to the function(s) referenced in the associated **calcattr** tag |
| **ENVTYPE**<br>(one value) | Search for functions in the immediate vicinity of the function or in the entire process instance | **DIRECT** (only immediately adjacent functions)<br>**INDIRECT** (all functions in the specified search direction) |
| **OBJECTNAMEFILTER**<br>(optional, multiple values) | Limits the search to particular internal function names, with use of the placeholders ***** and **?** as required. Several name patterns are specified using **key** as follows:<br>**OBJECTNAMEFILTER.0**<br>**OBJECTNAMEFILTER.1**<br>**OBJECTNAMEFILTER.2**, etc. | ***AUFT***,**????AUFT***<br>**FCT_AUFT1**, **FCT_AUFT_2**<br>***AUFT??** |

**Warning**

Note that the name (**key**) of each parameter is written in upper case.

**Example (extract from measure configuration)**

```
...
<calcattr name="AT_KI_COUNTFUNC_ENV" type="OT_FUNC"
                        objectname="FCT_CREATE_ORD">
  <calcclass name="com.idsscheer.ppm.server.
                keyindicator.attributecalculator.
            ZAttributeCalculatorObjectCounterByEpcEnv">
    <calcparam key="DIRECTION" value="FORWARD"/>
    <calcparam key="ENVTYPE" value="DIRECT"/>
    <calcparam key="OBJECTNAMEFILTER.0"
                            value="FCT_CREATE_*"/>
    <calcparam key="OBJECTNAMEFILTER.1"
                              value="FCT_ORDER_*">
  </calcclass>
</calcattr>
...
```

The configured search retrieves all successor functions directly adjacent to the current function. The set of functions retrieved is further limited by the specified filter expressions **FCT_CREATE_*** and **FCT_ORDER_***. The number of objects in the set of functions retrieved is calculated and is saved in the calculated **AT_KI_COUNTFUNC_ENV** attribute type.

## ATTRIBUTECOPIERBYEPCENV

Use this calculation class to define dependencies between several attribute calculations (**<depends attrname="..." type="..."/>**). For example, it can be useful to only execute a particular attribute calculation when certain attribute type values have already been copied to selected functions using the **AttributeCopierByEpcEnv** calculation class.

In the actual calculation, a configurable search starting from the current function is used to retrieve particular adjacent functions.
Particular attribute type values for a function are then copied to one or more functions in line with the specified parameters. A distinction is made between the following two main cases:

### 1. CASE

The current function will be the destination of the copying operation**<calcparam key="COPYROLE" value="DESTINATION"/>**).
If several adjacent functions are retrieved, a function needs to be selected from the set of functions retrieved to be the source of the copying operation. In this case, the **COPYTYPE** parameter must have the value **1-TO-1**. You should sort the set of functions, otherwise the value is copied from a random function, or no value is copied if the function does not have the attribute.

### 2. CASE

The current function will be the source for the copying operation (**<calcparam key="COPYROLE" value="SOURCE"/>**).
The specified attribute type values are to be copied from the starting function to another specified function. The **COPYTYPE** parameter must have the value **1-TO-1** as the search can retrieve

several adjacent functions. The **SORTATTRIBUTE** and **SORTTYPE** parameters are used to select a destination function from the set retrieved.

If the specified attribute type values are to be copied from the starting function to all adjacent functions, the **COPYTYPE** parameter must have the value **1-TO-N**. In this case, it is not necessary to specify the **SORTATTRIBUTE** and **SORTTYPE** sorting parameters as there is no need to make any further distinction between the functions found.

### IDENTIFY THE SOURCE FOR COPYING

To establish the source and target for the copying operation the **SORTATTRIBUTE** and **SORTTYPE** parameters are used to determine the function that is in first position based on the specified sort direction.

### ATTRIBUTE TYPE VALUES TO BE COPIED

The list of attribute type values to be copied is specified with consecutive numbering using the **SOURCEATTRIBUTE.<x>** or **DESTINATIONATTRIBUTE.<x>** parameters, with **<x>** being an integer. If no destination attribute type is specified, the source attribute type is created as the destination attribute type with corresponding values for the specified functions.

You can control the behavior of the **AttributeCopierByEpcEnv** calculation class by specifying the following parameters:

| Parameter | Description | Example value |
|---|---|---|
| **DIRECTION**<br>(one value) | Direction of search for predecessor or successor functions starting from the current function | **FORWARD** (successor functions) or<br>**BACKWARD** (predecessor functions) in relation to the function(s) referenced in the associated **calcattr** tag |
| **ENVTYPE**<br>(one value) | Search for functions in the specified direction in the immediate vicinity of the function or in the entire process instance. All found functions will be added to an unsorted set. | **DIRECT** (only immediately adjacent functions)<br>**INDIRECT** (all functions in the specified search direction) |

| Parameter | Description | Example value |
|---|---|---|
| **OBJECTNAMEFILTER**<br>(optional, multiple values) | Limits the search to particular internal function names, with use of the placeholders ***** and **?** as required. Several name patterns are specified using **key** as follows:<br>**OBJECTNAMEFILTER.0**<br>**OBJECTNAMEFILTER.1**<br>**OBJECTNAMEFILTER.2**, etc. | ***AUFT***,**????AUFT***<br>**FCT_AUFT1**, **FCT_AUFT_2**<br>***AUFT??** |
| **COPYROLE**<br>(one value) | Role of the current function in the copying process. The adjacent functions identified by the search each take opposing roles. | **SOURCE** (copying source)<br>**DESTINATION** (copying destination) |
| **COPYTYPE**<br>(one value) | If multiple functions are identified as the copying destination, this parameter specifies whether the value is to be copied to all of them or just to one function. | **1-TO-1** (value is copied to one function)<br>**1-TO-N** (value is copied to all functions retrieved) |
| **SORTATTRIBUTE**<br>(optional, one value) | Attribute used as sorting criterion if multiple functions are retrieved. This will only be considered if COPYTYPE 1-TO-1 is specified. | Existing<br>PPM attribute type |
| **SORTTYPE**<br>(optional, one value) | Sorting direction for the selected sorting criterion. This will only be considered if COPYTYPE 1-TO-1 is specified. | **ASC** (default value: ascending)<br>**DESC** (descending) |
| **SORTNULLVALUES**<br>(optional, one value) | Specifies whether objects whose sort attribute has not been entered are to be sorted by maximum or minimum value, or if an error message is to be output. | **MIN**, **MAX**, **DEFAULT** (default value) |

| Parameter | Description | Example value |
|---|---|---|
| **SOURCEATTRIBUTE.<x>** (multiple values) | The source attribute type whose value is to be copied. <x> is an integer and corresponds to destinationattribute.<x> | An existing PPM attribute type |
| **DESTINATION↵ ATTRIBUTE.<x>** (optional, multiple values) | The destination attribute type to which the value is to be copied. <x> is an integer and corresponds to sourceattribute.<x> | Existing PPM attribute type |

### Example

The configured search retrieves all predecessor functions directly adjacent to the current function, which is to be used as the source for the copying operation. If multiple predecessor functions are retrieved, the set of functions identified is sorted in descending order using the sorting criterion **AT_START_TIME**. The value of the **AT_START_TIME** attribute type is copied from the latest function to the **AT_END_TIME** attribute type for the destination function.

### EXTRACT FROM THE MEASURE CONFIGURATION

```
...
<calcattr name="AT_END_TIME" type="OT_FUNC" objectname="...">
  <calcclass name="com.idsscheer.ppm.server.
                keyindicator.attributecalculator.
           ZAttributeCalculatorAttributeCopierByEpcEnv">
    <calcparam key="DIRECTION" value="BACKWARD"/>
    <calcparam key="ENVTYPE" value="DIRECT"/>
    <calcparam key="COPYROLE" value="SOURCE"/>
    <calcparam key="COPYTYPE" value="1-TO-1"/>
    <calcparam key="SORTATTRIBUTE" value="AT_START_TIME"/>
    <calcparam key="SORTTYPE" value="DESC"/>
    <calcparam key="SOURCEATTRIBUTE.0"
                              value="AT_START_TIME"/>
    <calcparam key="DESTINATIONATTRIBUTE.0"
                               value="AT_END_TIME"/>
  </calcclass>
</calcattr>
...
```

## ATTRIBUTEAGGREGATORBYEPCENV

The **AttributeAggregatorByEpcEnv** calculation class adds up the specified numerical attribute type for all adjacent functions. The result is saved in the specified attribute type.

You can control the behavior of the **AttributeAggregatorByEpcEnv** calculation class by specifying the following parameters:

| Parameter | Description | Example value |
|---|---|---|
| **DIRECTION**<br>(one value) | Direction of search for predecessor or successor functions starting from the current function | **FORWARD** (successor functions) or<br>**BACKWARD** (predecessor functions) in relation to the function(s) referenced in the associated **calcattr** tag |
| **ENVTYPE**<br>(one value) | Search for functions in the specified direction in the immediate vicinity of the function or in the entire process instance | **DIRECT** (only immediately adjacent functions)<br>**INDIRECT** (all functions in the specified search direction) |
| **OBJECTNAMEFILTER**<br>(optional, multiple values) | Limits the search to particular internal function names, with use of the placeholders ***** and **?** as required. Several name patterns are specified using **key** as follows:<br>**OBJECTNAMEFILTER.0**<br>**OBJECTNAMEFILTER.1**<br>**OBJECTNAMEFILTER.2**, etc. | ***AUFT***,**????AUFT***<br>**FCT_AUFT1**, **FCT_AUFT_2**<br>***AUFT??** |
| **AGGREGATION_↵**<br>ATTRIBUTE<br>(one value) | Attribute type to be aggregated | Existing PPM attribute type |

### Example

In the example below, all successor functions are retrieved for the current function in the process instance, whose internal name begins with the string **FCT_ORDER_** or **FCT_INVOICING_**

followed by any four characters. The **AT_ORDER_VOL** attribute type is aggregated for the set of functions retrieved and saved in the **AT_KI_ORDER_VOL_AGG** attribute type.

### EXTRACT FROM THE MEASURE CONFIGURATION

```
...
<calcattr name="AT_KI_ORDER_VOL_AGG" type="OT_FUNC"
                                objectname="ORDER*">
  <calcclass name="com.idsscheer.ppm.server.
                keyindicator.attributecalculator.
       ZAttributeCalculatorAttributeAggregatorByEpcEnv">
    <calcparam key="DIRECTION" value="FORWARD"/>
    <calcparam key="ENVTYPE" value="INDIRECT"/>
    <calcparam key="OBJECTNAMEFILTER.0"
                               value="FCT_ORDER_*"/>
    <calcparam key="OBJECTNAMEFILTER.1"
                          value="FCT_INVOICING_????"/>
    <calcparam key="AGGREGATION_ATTRIBUTE"
                              value="AT_ORDER_VOL"/>
  </calcclass>
</calcattr>
...
```

## 8.1.3.1.6 Relation measures

### ORGCOPYATTRFROMFUNC

The **OrgCopyAttrFromFunc** calculation class copies the specified function attribute to the executing organizational unit. The calculation class is only available when using the **Interaction analysis** module. The result is saved in the specified attribute type for the relevant organizational unit.

For calculations using the **OrgCopyAttrFromFunc** class, you must specify the following parameters.

| Parameter | Description | Value or example |
|---|---|---|
| **attrname** | Identifier of function attribute to be copied | **AT_END_TIME** |

**Warning**

Only one function attribute can be copied for each **calcattr** XML element. Make sure that the data types of the source and target attributes are compatible. Essentially, all (including user-defined) numerical data types (see chapter on **Data types** (page 11): LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) are compatible with one another. The **convert** operator allows you to perform appropriate advance data type conversions (see chapter on **Logical operators** (page 100)). Make sure that the conversion is always carried out in the base scaling of the target data type.

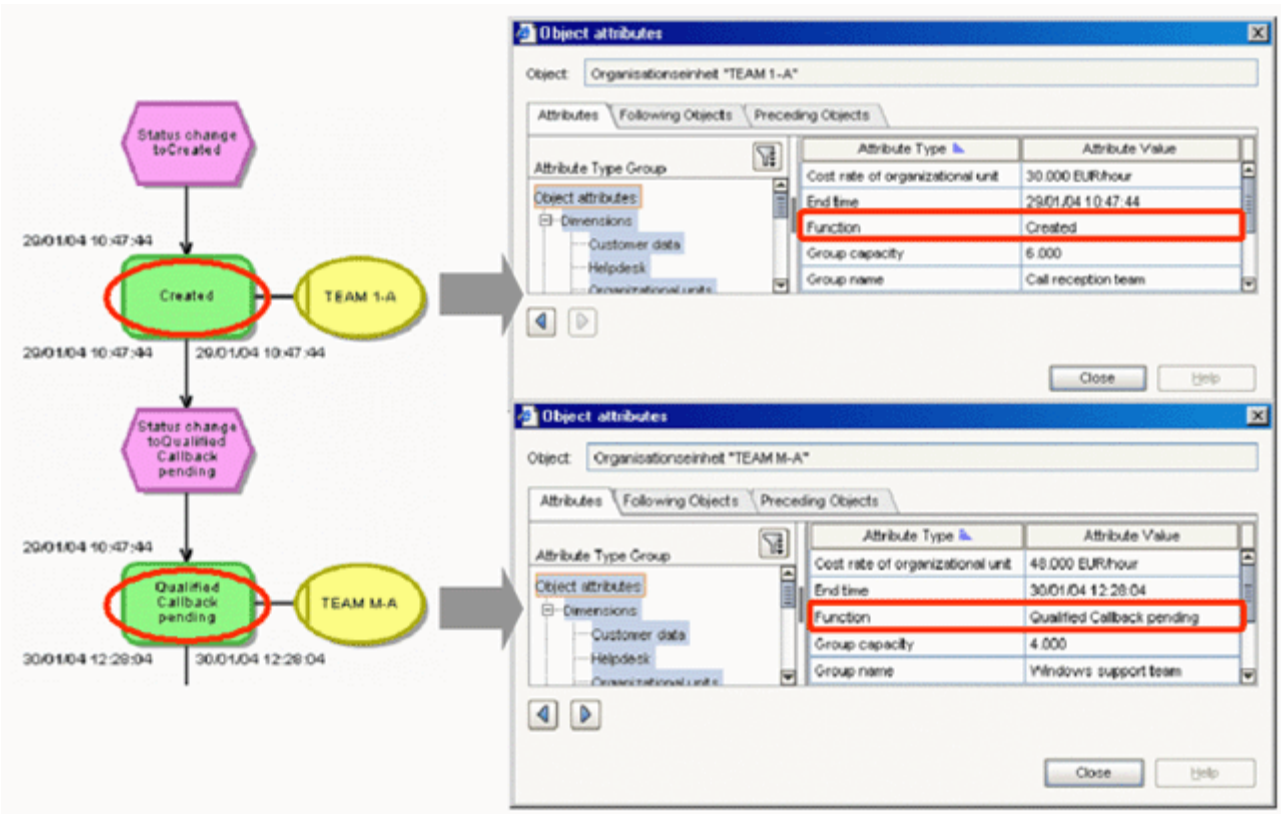
### Example

In the following example from the measure configuration, the **AT_OBJNAME** function attribute is copied to each executing organizational unit (**type="OT_ORG"**) as the **AT_FUNC** attribute. The source and destination attribute of the copying operation are both of the **TEXT** data type.

```
<calcattr name="AT_FUNC" type="OT_ORG">
  <calcclass name="com.idsscheer.ppm.server.keyindicator.
                                attributecalculator.
              ZAttributeCalculatorOrgCopyAttrFromFunc">
      <calcparam key="attrname" value="AT_OBJNAME"/>
  </calcclass>
</calcattr>
```

If the **AT_FUNC** attribute is already specified at the organizational units, you can use **delete="yes"** in the **calcattr** instruction to define that the copying operation should first delete the attribute value.

The following example graphic from the **Interaction analysis** module shows the result of the attribute copy operation in the open object attribute dialogs for the two organizational units **TEAM 1-A** and **TEAM M-A**. The relevant value of the **AT_OBJNAME** function attribute has been copied to each organizational unit as the **Function** attribute (**AT_FUNC**) of the function executed by the corresponding organizational unit.



## 8.1.3.1.7 Process conformance

From version 10.2, PPM provides a process conformance check for processes that have been modeled in ARIS and that are to be imported into PPM.

PPM provides a special conformance configuration package that contains all customizing elements required for calculating process conformance. Among other things, the package contains the conformance measure **Conformance rate** (KI_CONFORMANCE_RATE (page 70)) calculated on the process instances and an additional relation **Conformance issue** (REL_CONFORMANCE_ISSUE (page 70)) that contains detailed information about why process instances were considered non-conformant.

For details on the process conformance check, see the chapter ARIS process conformance check in the documentation PPM Customizing Toolkit.

### 8.1.3.1.7.1 Conformance rate measure

The conformance customizing package contains the **KI_CONFORMANCE_RATE** measure with source attribute **AT_KI_CONFORMANCE_RATE**. The attribute is calculated by calculation class **ZAttributeCalculatorConformanceRate**.

In CTK, the attribute calculation class is named **Conformance rate**.

The XML structure of **AT_KI_CONFORMANCE_RATE** looks like this:

```
<calcattr name="AT_KI_CONFORMANCE_RATE" type="PROCESS" delete="no">
      <calcclass name="com.idsscheer.ppm.server.keyindicator.
            attributecalculator.ZAttributeCalculatorConformanceRate"
            loglevel="VERBOSE" />
</calcattr>
```

The output of the calculation is a value of either 0.0 (non-conformant) or 1.0 (conformant). The measure is aggregated by average and shows the proportion of conformant processes to all processes.

### 8.1.3.1.7.2 Conformance issue relation

The **Conformance issue** relation consists of a source object dimension **Preceding function**, a target object dimension **Non-conforming function**, a single level text dimension **Conformance issue type** containing a keyword for the issue type, and a measure **Number of conformance issues** that counts issues within a process instance.

The XML structure of the conformance issue relation **REL_CONFORMANCE_ISSUE** looks like this:

```
<relation name="REL_CONFORMANCE_ISSUE">
      <description name="Conformance issue" language="en" />
      <sourcedim name="D_PRECEDING_FUNCTION" />
      <targetdim name="D_NONCONFORMING_FUNCTION" />
      <refki name="RNUM_REL_CONFORMANCE_ISSUE" />
      <refdim name="D_CONFORMANCE_ISSUE_TYPE" />
</relation>
```

The relation must have a dependency on AT_KI_CONFORMANCE_RATE (page 70) due to the internal workings of the calculation. The calculation rule is

```
<calcrel name="REL_CONFORMANCE_ISSUE">
      <depends attrname="AT_KI_CONFORMANCE_RATE" type="PROCESS" />
      <calcclass
name="com.idsscheer.ppm.server.keyindicator.relation.calculator.
       ZRelationCalculatorConformanceIssues" loglevel="VERBOSE" />
</calcrel>
```

The relation is calculated by calculation class **ZRelationCalculatorConformanceIssues**.

The issue type information is stored in the key attribute of the relation that was configured for the assigned dimension. The attribute name can be freely chosen. The dimension must be a single level text dimension with keyword **D_CONFORMANCE_ISSUE_TYPE**. Otherwise, the relation is not calculated and none of the associated measures and dimensions have a value. If you need to use a different keyword, for example because there already is another dimension **D_CONFORMANCE_ISSUE_TYPE** in the customizing, you can supply that keyword to the calculation class **ZRelationCalculatorConformanceIssues** as the value of the parameter **issue_type_dimension_keyword**.

### 8.1.3.1.8 Convert time spans in milliseconds

The **ZAttributeCalculatorConvertMillisecondDuration** class is a parameterized attribute calculator class to be used for converting time spans in the internal Software AG format **MillisecondDurationType** into a PPM time span format. The **MillisecondDurationType** format consists of a value in float format (no unit) containing a time span number in milliseconds.

**Example of a time span output in PPM event format**

```
<attribute type="DURATION_IN_MS">12618.0</attribute>
```

**Example of the use of an attribute calculator class**

```
<calcattr name="AT_KI_DURATION" type="OT_FUNC\" >
  <calcclass name="com.idsscheer.ppm.server.keyindicator.attributecalculator.
                ZAttributeCalculatorConvertMillisecondDuration">
    <calcparam key="ATTRIBUTE_MILLISECOND_DURATION" value="AT_DURATION_IN_MS"/>.
  </calcclass>
</calcattr>"
```

In this example, the attribute calculator identifies at all functions of a process instance the value of the **AT_DURATION_IN_MS** attribute, interprets the value as a value in milliseconds, and converts it into seconds. The result is rounded to full seconds and written to the **AT_KI_DURATION** attribute.

A precondition for using the calculator class is that the source attribute (in the example: AT_DURATION_IN_MS) be of the **TEXT** or **DOUBLE** type and the target attribute (in the example: **AT_KI_DURATION**) be of the **TIMESPAN** type.

### 8.1.3.1.9 Mark as large EPC

The **ZAttributeCalculatorFunctionCount** attribute calculator writes the number of function nodes in the EPC to a configurable attribute at the process level.

**Example**

```
<calcattr name="AT_FCT_COUNT" type="PROCESS" >
    <calcclass name="com.idsscheer.ppm.server.keyindicator.
      attributecalculator.ZAttributeCalculatorFunctionCount">
```

```
    </calcclass>
</calcattr>
```

There is no calculation for aggregated EPCs.

Further information on **How to handle large EPCs** is available in the documentation **PPM Data Import**.

## 8.1.3.2 Operands

Operands provide the input values (parameters) for calculation rules. The attribute calculator differentiates between three types of operators: Set of values, Value and Constant.

### 8.1.3.2.1 Set of values (XML element attribute)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="..." type="...">
    <calculation>
      ...
      <attribute name="..." nodetype="..."
                      objectname="..." onerror="..."/>
      ...
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

The **attribute** XML element returns a set of attribute values as its result. It contains all object attribute values (**nodetype** not equal to **PROCESS**) specified in the process instance for the specified attribute.
For process instance attributes (**nodetype="PROCESS"**), the set of values only contains the value of the attribute specified for the instance.
If the attribute is not specified within the process instance, the set of values is empty.

| XML tag | Description |
|---|---|
| **name** | Internal name of the attribute<br>If the attribute name is specified with a ***** placeholder at the end, the values of all attributes whose names begin with the specified string are included in the set of results. |
| **nodetype** | Attribute type: Function (OT_FUNC) or process instance attribute (**PROCESS**) |

| XML tag | Description |
|---|---|
| **objectname** (optional) | For function attributes (**nodetype="**OT_FUNC**"**) the set of values can be limited to attribute values for the specified object name.<br>If **this** is specified as the object name, the attribute value is retrieved for only the function for which the calculation is currently being executed.<br>If **like** is specified as the object name, the attribute value of all functions with the same name is retrieved. |
| **onerror** (optional) | Controls the behavior of the Measure calculator if no set of attribute values can be retrieved:<br>**EXIT_WARNING**: Cancels the current attribute calculation and outputs a warning to the log.<br>**EXIT_NO_WARNING**: Cancels the current attribute calculation with no output of a warning to the log.<br>**CONTINUE**: Default value. The current attribute calculation is continued with an empty set. The superordinate operators determine error handling procedures. There is no output in the log. |

**Warning**

Specifying an object name of **this** or **like** in the **objectname** XML attribute is only permitted for the calculation of function attributes (**nodetype="OT_FUNC"**).

### Example 1

The set of values contains all **AT_KI_FDLZ** attribute values (function cycle time) for functions that have the same name (**AT_OBJNAME_INTERN** function attribute) as the currently calculated function.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_KI_FDLZSUM" type="OT_FUNC">
    <calculation>
      <sum>
        <attribute name="AT_KI_FDLZ" nodetype="OT_FUNC"
                                  objectname="like"/>
      </sum>
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

### Example 2

The values of all attributes whose names begin with **AT_SALES_VOLUME_** are taken into account.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_KI_XXX" type="PROCESS">
    <calculation>
      <sum>
        <attribute name="AT_SALES_VOLUME_*"
                                 nodetype="OT_FUNC"/>
      </sum>
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

Pattern matching of the internal attribute names is subject to the following limitations:

- Placeholders are not permitted in filtered attributes (filteredattribute XML element) as this operand returns a single attribute value and only relates to a single attribute.
- The placeholder * is only supported at the end of an attribute name.
- The attributes affected by pattern matching must be of the same data type.

### 8.1.3.2.2 Values (XML element filteredattribute)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="..." type="...">
    <calculation>
 ...
      <filteredattribute name="..." nodetype="..."
           objectname="..." filter="..." onerror="..."/>
      ...
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

To calculate a concrete value from a set of values, all of the mathematical functions presented in chapter **Operators producing a value** (page 94) can be used, which deliver a specific value as the result. Alternatively, you can use the **filteredattribute** XML element to calculate a specific value from a set of attribute values.

The entries for **name**, **nodetype**, **objectname**, and **onerror** correspond to those for the **attribute** XML element.

| XML tag | Description |
|---|---|
| **name** | Internal name of attribute<br>Use of pattern matching is not supported with the **filteredattribute** XML element, as it relates to a single attribute only. |

| XML tag | Description |
|---|---|
| **nodetype** | Attribute type: Function attribute (**OT_FUNC**), process instance attribute (**PROCESS**), or relation attribute (**RELATION**) |
| **objectname** (optional) | For function attributes (**nodetype="OT_FUNC"**) the set of values can be limited to attribute values for the specified object name.<br>For relation attributes (**nodetype="RELATION"**), the following values are permitted:<br>**this**<br>The currently calculated attribute (**calcattr name="..." type="RELATION" relname="REL_..."**) is searched at the relation.<br>**source**<br>The source object of the relation is searched for the attribute.<br>**target**<br>The target object of the relation is searched for the attribute. |
| **filter** (optional) | Filter that is used to select the element from the set of values (not for **objectname="this"**):<br>**EARLY**<br>The attribute value is transferred for the object for which one of the **AT_START_TIME** and **AT_END_TIME** attributes gives the earliest time overall.<br>**LATEST**<br>The attribute value is transferred for the object for which one of the **AT_START_TIME** and **AT_END_TIME** attributes gives the latest time overall. |
| **onerror** (optional) | Controls the behavior of the Measure calculator if no attribute value can be identified:<br>**EXIT_WARNING**: Cancels the current attribute calculation and outputs a warning to the log.<br>**EXIT_NO_WARNING**: Cancels the current attribute calculation with no output of a warning to the log.<br>**CONTINUE**: Default value. The current attribute calculation is continued with NULL. The superordinate operators determine error handling procedures. There is no output in the log. |

By specifying the object type (**nodetype**), attributes with the same name in the process instance and for objects belonging to the process instance can be differentiated.

### 8.1.3.2.3 Constants (XML element constant)

The value of a constant is specified in the **CDATA** section of the **<dataitem>** XML element. The following example defines a time span constant of ten minutes:

```
<constant>
   <dataitem>
      10 MINUTE
      <datatype name="TIMESPAN"></datatype>
   </dataitem>
</constant>
```

If the entries for the data type and value of the constants are correct, possible entries in the **value** attribute for the **<dataitem>** element are ignored. The following definition creates a constant of two hours:

```
<constant>
   <dataitem value="9">
      2 HOUR
      <datatype name="TIMESPAN"></datatype>
   </dataitem>
</constant>
```

If the entry in the **CDATA** section of the **<dataitem>** element returns no value or a value with an invalid data type, it is ignored. Instead, the entries in the **value** XML attribute are processed. In the following example, the value of the constant with the **DOUBLE** data type is specified, although the **LONG** data type is expected. The incorrect value entry is ignored and the value of the **value** attribute (**"2"**) is written to the constant instead:

```
<constant>
   <dataitem value="2">
      4.0
      <datatype name="LONG"></datatype>
   </dataitem>
</constant>
```

If the value specified in the **value** attribute does not match the expected data type, the calculation is canceled:

```
<constant>
   <dataitem value="2.4">
      4.0
      <datatype name="LONG"></datatype>
   </dataitem>
</constant>
```

In the following example the calculation is canceled, as the data type and the value specified do not match and there is no entry in the **value** XML attribute:

```
<constant>
   <dataitem>
      4.0
      <datatype name="LONG"></datatype>
   </dataitem>
```

```
</constant>
```

A constant must always be specified with the unit that is permissible for the attribute data type. The data type must be known in the PPM system.

Numerical constants consist of the specification of the value with a unit that is permissible for the data type and the data type itself.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM 'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="..." type="...">
    <calculation>
      ...
      <constant>
        <dataitem>
          10 MINUTE
          <datatype name="TIMESPAN"></datatype>
        </dataitem>
      </constant>
      ...
    </calculation>
  </calcattr>
  …
</keyindicatorconfig>
```

| XML element | Description |
|---|---|
| **dataitem value** | Value of constant with unit |
| **datatype name** | Name of the data type. Both internal and user-defined data types can be used. |

Alphanumeric constants are specified as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="..." type="...">
    <calculation>
      ...
      <constant>
        <dataitem>
          Constant text
          <datatype name="TEXT"/>
        </dataitem>
      </constant>
      ...
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

### 8.1.3.2.4 Determining attribute values

Attribute values both with and without an object reference can be used for attribute calculation.

### 8.1.3.2.4.1 Attribute values without object reference

The specified attribute is used for all process instance objects of the object type specified by **nodetype** for which it is entered. This results in a set of values, which contains a number of elements corresponding to the occurrence of the attribute.

**Example**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="..." type="...">
    <calculation>
 ...
      <attribute name="AT_ABC" nodetype="OT_FUNC"/>
      ...
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

The values of the **AT_ABC** attribute for all functions in the process instance currently being calculated are included in the set of values.

### 8.1.3.2.4.2 Attribute values with object reference

The specified attribute is only used for the functions (**nodetype="OT_FUNC"**) with the specified name (**objectname"FCT_..."**). (The object name specified with **objectname** corresponds to the value of the **AT_OBJNAME_INTERN** function attribute.) Once again, a set of values containing more than one element can result as the specified object can occur several times in the process instance.

**Example 1**

The values of the **AT_AUFNR** attribute for all functions in the process instance currently being calculated with the name **FCT_CREATE_ORDER** are included in the set of values.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="..." type="...">
    <calculation>
 ...
      <attribute name="AT_AUFNR" nodetype="OT_FUNC
                objectname="FCT_AUFTRAG_ANLEGEN"/>
      ...
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

The color of functions or events in the EPC view can be specified using the default **AT_BGND_COLOUR** attribute. The following calculation rule assigns a red color to all functions with the internal name **SAP.WAUS**:

```
<calcattr name="AT_BGND_COLOUR" type="OT_FUNC"
                        objectname="SAP.WAUS">
   <calculation>
      <constant>
         <dataitem>
            <datatype name="TEXT">255,0,0</datatype>
         </dataitem>
      </constant>
   </calculation>
</calcattr>
```

This calculation rule can be used within a conditional attribute calculation, for example. The relevant color value is specified as an RGB value. Particular objects or object types can also be assigned a color in attribute mapping.

**Example 2**

The calculation rule totals the cycle time for the function (**AT_KI_FDLZ**) for functions with the same name (identical value for the **AT_OBJNAME_INTERN** attribute).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_KI_FDLZSUM" type="OT_FUNC">
    <calculation>
      <sum>
        <attribute name="AT_KI_FDLZ" nodetype="OT_FUNC"
                                objectname="like"/>
      </sum>
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

**Warning**

Specifying an object name or the options **this** or **like** in the **objectname** XML attribute is only permitted for the calculation of function attributes (OT_FUNC).

To use an operator that expects single values as an operand (for example, **<plus>**) with an operand that returns sets of values (for example, **<attribute>**), you need to use suitable operators to retrieve a single value from a set of values (for example, **<min>** or **<max>**). Alternatively, you can use the **<filteredattribute>** XML element to retrieve one value from a set of values to be used for the subsequent attribute calculation.

## 8.1.3.3 Conditional attribute type access

Within a calculation rule for calculation of a function attribute, you can limit the set of attribute values to be taken into account by specifying a condition relating to other attribute types for the same function (**objectname="this"**). To configure the condition, you need to use a Boolean

operator as the root operator (see chapter Logical operators (page 100)). The condition can be nested at any depth. If the condition check results in the value **TRUE**, the value of the attribute type for which the condition is defined is included in the subsequent calculation.
You can specify conditions for **<attribute>** and **<filteredattribute>**.

**Example**

From the set of values for the **AT_HRMODUL** function attribute, only those attribute type values of functions for which the **AT_VORG_TYPE** attribute type is also specified (**objectname="this"**) with the value **019** are to be taken into account.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_KI_HRMODUL" type="PROCESS">
    <calculation>
      <max>
        <attribute name="AT_HRMODUL" nodetype="OT_FUNC">
          <in>
            <constant>
              <dataitem>
               019
               <datatype name="TEXT">Text</datatype>
              </dataitem>
            </constant>
            <attribute name="AT_VORG_TYPE"
                  nodetype="OT_FUNC" objectname="this"/>
          </in>
        </attribute>
      </max>
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

Define conditional attribute type access in PPM Customizing Toolkit to prevent syntax errors. You can create the corresponding calculation rules in the **Measures and dimensions** module using the **Calculated attribute types** menu. In particular, this prevents incorrect use of the **attribute** and **filteredattribute** operands with the corresponding logic operators.

## 8.1.3.4 Operators

In calculation rules for attribute types or calculation functions, the individual operand types (sets of values, values, constants) are linked to one another using operators. When linking attribute types, if all operands have the same data type, the results of attribute calculations are returned as this data type.

For each operator, you can use the **mode** XML attribute to specify how exceptions are to be handled (for example, **<addtimespan mode="PPM4">**). Valid values are **PPM3** for the behavior up to and including PPM 3.2.1 and **PPM4** for the more fault-tolerant behavior from PPM

4.0 onward.
For reasons of backwards compatibility, the default value is **PPM3**.
In calculation rules that you create using PPM Customizing Toolkit the operators used are assigned the value **PPM4** by default.

The calculation and error behavior of the two different modes is described for each operator starting from chapter **Mathematic operators** (page 82).

In your attribute type calculations, define a default return value **defaultvalue**, which is assigned to the attribute type to be calculated if the attribute calculation fails.

**Warning**

Do not combine numerical values with non-numerical values in a calculation rule (for example, **TEXT** with **DOUBLE**), as such calculation rules lead to the calculation being canceled.

Numerical data types can be freely combined with one another (for example, using the **set** operator). Values are always given the base unit for the attribute type. For attribute type calculations with mixed numerical data types (for example, **DOUBLE**, **TIMESPAN**, **FACTORYTIMESPAN**) all values are used without units and the result is saved as the **DOUBLE** data type. You can then save this value in the relevant base unit as a PPM target attribute of another data type.

An operation is specified in the form of inverted Polish notation, that is, the operator type is specified first, followed by the operands. In XML notation, it looks like this:

```
<operator 1>
    <operand m>
    ...
    </operand m>
    <operand m+1>
    ...
    </operand m+1>
    <operator 2>
        <operand n>
        ...
        </operand n>
        <operand n+1>
        ...
        </operand n+1>
    </operator 2>
</operator 1>
```

The **operator** XML element returns the calculated numerical value (numerical result of the operands linked by the operator). The unit for the result is determined by the data type of the attribute type to which the result value is assigned. Operators themselves can be part of a higher-level operator.

### Example

Calculation of the circumference of a circle with a radius of 6
(Circumference = 2 * p * radius):

```
  <times>
    <constant>
      <dataitem value="2">
        <datatype name="DOUBLE"/>
      </dataitem>
```

```
    </constant>
    <constant>
      <dataitem value="3.1415">
        <datatype name="DOUBLE"/>
      </dataitem>
    </constant>
    <constant>
      <dataitem value="6">
        <datatype name="DOUBLE"/>
      </dataitem>
    </constant>
  </times>
```

An alternative option leading to the same result is to create a set from the operands and to multiply all elements in the set by one another:

```
<product>
  <set>
    <constant>
      <dataitem value="2">
        <datatype name="DOUBLE"/>
      </dataitem>
    </constant>
    <constant>
      <dataitem value="3.1415">
        <datatype name="DOUBLE"/>
      </dataitem>
    </constant>
    <constant>
      <dataitem value="6">
        <datatype name="DOUBLE"/>
      </dataitem>
    </constant>
  </set>
<product>
```

The operators described in the following chapters are available for the calculation of attribute type values.

### 8.1.3.4.1 Mathematic operators

The following operators are available: plus, minus, timespan, times, divide, abs, div, mod, squareroot, and round.

#### ADDITION

| | |
|---|---|
| XML tag: | plus |
| Operands: | at least two values |

| | | |
|---|---|---|
| Synopsis: | <plus><br>  <value 1><br>  <value 2><br>  <value n><br></plus> | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value | |
| Result type: | Operand data type with identical data type.<br>DOUBLE for mixed numerical data types that, in this case, are automatically converted to DOUBLE. | |
| Description: | Adds the values specified in the XML element | |
| Calculation (PPM3) | Result | Sum of all operands |
| | Error | If at least one operand equals NULL or at least one operand is of a non-numerical data type |
| Calculation (PPM4) | Result | NULL if at least one operand equals NULL, otherwise sum of all operands. |
| | Error | Only if the data type is non-numerical |
| Example: | - | |

## SUBTRACTION

| | | |
|---|---|---|
| XML tag: | minus | |
| Operands: | exactly two values | |
| Synopsis: | <minus><br>  <value 1><br>  <value 2><br></minus> | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value (difference) | |
| Result type: | Operand data type with identical data type.<br>DOUBLE for mixed numerical data types that, in this case, are automatically converted to DOUBLE. | |
| Description: | Subtracts value 2 from value 1 | |
| Calculation (PPM3) | Result | Result of subtracting operand 2 from operand 1 |

| | | |
|---|---|---|
| | Error | If at least one operand equals NULL or at least one operand is of a non-numerical data type |
| Calculation (PPM4) | Result | NULL if at least one operand is equal to NULL, otherwise result of subtracting operand 2 from operand 1 |
| | Error | Only if the data type is non-numerical |
| Example: | - | |

## TIME SPAN

| | | |
|---|---|---|
| XML tag: | timespan | |
| Operands: | Exactly two values (points in time) | |
| Synopsis: | `<timespan>`<br>`  <time 1>`<br>`  <time 2>`<br>`</timespan>` | |
| Operands: | TIME (TIMESTAMP, DATE) | |
| Result: | Value (time span) | |
| Result type: | TIMESPAN or<br>FACTORYTIMESPAN when using a factory calendar | |
| Description: | Calculates the time difference between time 1 and time 2. If the difference is negative, the value 0 is returned. To use the factory calendar to calculate the time difference, give the optional **type** XML element the value **FACTORYCALENDAR**.<br>Default value: **NORMAL** | |
| Calculation (PPM3) | Result | Time span between operand 1 and operand 2 (operand 1 minus operand 2) |
| | Error | If at least one operand equals NULL or at least one is of an invalid data type |
| Calculation (PPM4) | Result | NULL if at least one operand is equal to NULL, otherwise time span between operand 1 and operand 2 (operand 1 minus operand 2) |
| | Error | Only if data type is invalid |

| | |
|---|---|
| Example: | `<timespan type="FACTORYCALENDAR"`<br><br>`directoryname="custom/client/factorycal"`<br>`attributename="AT_FC_XYZ">`<br>`<max>`<br>`<attribute name="AT_GOODS_RECEIPT_DATE"`<br>`nodetype="OT_FUNC"`<br>`objectname="SAP.MM_WE_ANLEG"/>`<br>`</max>`<br>`<min>`<br>`<attribute name="AT_END_TIME"`<br>`nodetype="OT_FUNC"`<br>`objectname="SAP.MM_BANF_ANLEG"/>`<br>`</min>`<br>`</timespan>` |

If you are using a factory calendar you can also calculate negative time spans by specifying the optional XML attribute **negfactorytimespan="TRUE"**.
Default value: **FALSE**

You can also perform time span calculations based on external factory calendars by specifying a factory calendar XML file. In the optional **directoryname** XML attribute, specify the directory containing the factory calendar to be used. The **attributename** attribute is used to specify the name of the attribute type containing the name of the factory calendar file to be used. The attribute type must be specified for the corresponding object or process instance. The two XML attributes must always be specified together.

## MULTIPLICATION

| | | |
|---|---|---|
| XML tag: | times | |
| Operands: | at least two values | |
| Synopsis: | `<times>`<br>`<value 1>`<br>`<value 2>`<br>`<value n>`<br>`</times>` | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value (product) | |
| Result type: | Operand data type with identical data type.<br>DOUBLE for mixed numerical data types that, in this case, are automatically converted to DOUBLE. | |
| Description: | Multiplies the values specified in the XML element. | |
| Calculation | Result | Result of multiplying operands 1 to n |

| | | |
|---|---|---|
| (PPM3) | Error | If at least one operand equals NULL or at least one operand is of a non-numerical data type |
| Calculation (PPM4) | Result | NULL if at least one operand is equal to NULL, otherwise result of multiplying all operands |
| | Error | Only if the data type is non-numerical |
| Example: | - | |

## DIVISION

| | | |
|---|---|---|
| XML tag: | divide | |
| Operands: | exactly two values | |
| Synopsis: | `<divide>`<br>`  <value 1>`<br>`  <value 2>`<br>`</divide>` | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value (quotient) | |
| Result type: | Always DOUBLE | |
| Description: | Divides value 1 by value 2. | |
| Calculation (PPM3) | Result | Result of dividing operand 1 by operand 2 |
| | Error | If at least one operand equals NULL or at least one operand is of an invalid data type, or operand 2 = 0 |
| Calculation (PPM4) | Result | NULL if at least one operand is equal to NULL, otherwise result of dividing operand 1 by operand 2 |
| | Error | If at least one operand is of an invalid data type or operand 2 = 0 |
| Example: | - | |

## AMOUNT

| | |
|---|---|
| XML tag: | abs |
| Operands: | exactly one value |
| Synopsis: | `<abs>`<br>`  <value 1>`<br>`</abs>` |

| | | |
|---|---|---|
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value (absolute value) | |
| Result type: | Operand data type | |
| Description: | Returns the amount of a value. | |
| Calculation (PPM3) | Result | Absolute operand value |
| | Error | If an operand equals NULL or is of a non-numerical data type |
| Calculation (PPM4) | Result | NULL if operand equals NULL, otherwise absolute operand value |
| | Error | Only if the data type is non-numerical |
| Example: | - | |

## INTEGER DIVISION

| | | |
|---|---|---|
| XML tag: | div | |
| Operands: | exactly two integer values | |
| Synopsis: | `<div>`<br>`  <value 1>`<br>`  <value 2>`<br>`</div>` | |
| Operands: | LONG | |
| Result: | Integer value of division | |
| Result type: | LONG | |
| Description: | Returns the integer value for how often value 2 is contained in value 1. Remainders are ignored. For proper fractions, 0 is returned. | |
| Calculation (PPM4 only) | Result | NULL if at least one operand is equal to NULL, otherwise integer result of dividing operand 1 by operand 2 |
| | Error | If at least one operand is of an invalid data type (not LONG) or operand 2 = 0 |

| | |
|---|---|
| Example: | `<div>`<br>`  <max>`<br>`  <attribute name="AT_COST"`<br>`  nodetype="FUNCTION"/>`<br>`  </max>`<br>`  <constant>`<br>`  <dataitem>`<br>`  5`<br>`  <datatype name="LONG">`<br>`  Long`<br>`  </datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`</div>` |

## MODULO

| | | |
|---|---|---|
| XML tag: | mod | |
| Operands: | exactly two integer values | |
| Synopsis: | `<mod>`<br>`  <value 1>`<br>`  <value 2>`<br>`</mod>` | |
| Operands: | LONG | |
| Result: | Integer remainder | |
| Result type: | LONG | |
| Description: | Returns the remainder of an integer division of value 1 by value 2. For proper fractions, the value of the first operand is returned. If value 1 = value 2, 0 is returned. | |
| Calculation (PPM4 only) | Result | NULL if at least one operand is equal to NULL, otherwise remainder of integer division of operand 1 by operand 2 |
| | Error | If at least one operand is of an invalid data type (not LONG) or operand 2 = 0 |
| Example: | `<mod>`<br>`  <filteredattribute name="AT_COST"`<br>`  nodetype="FUNCTION"/>`<br>`  <constant>`<br>`  <dataitem>`<br>`  3`<br>`  <datatype name="LONG">Long</datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`</mod>` | |

## SQUARE ROOT

| | | |
|---|---|---|
| XML tag: | squareroot | |
| Operands: | exactly one value | |
| Synopsis: | `<squareroot>`<br>`  <value 1>`<br>`</squareroot >` | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE, and user-defined types, for example, COST) | |
| Result: | Square root | |
| Result type: | DOUBLE data type | |
| Description: | Calculates the square root of the value entered. | |
| Calculation (PPM4 only) | Result | NULL if the operand equals NULL, otherwise the square root of the numeric operand |
| | Error | If the operand has an invalid data type (not numerical) or if the value of the operand is less than 0 |
| Example: | - | |

## ROUND

| | |
|---|---|
| XML tag: | round |
| Operands: | A single value of the TIMESPAN type |
| Synopsis: | `<round>`<br>`  <value>`<br>`</round>` |
| Operands: | TIMESPAN |
| Result: | Rounded time span value |
| Result type: | TIMESPAN |
| XML attributes | scale (MINUTE\|HOUR\|DAY\|WEEK\|MONTH\|YEAR)<br>roundingkind (ROUND\|FLOOR\|CEIL) "ROUND" |

| | | |
|---|---|---|
| Description: | Returns the rounded value for time spans. Only values of the TIMESPAN data type can be rounded. The scale to be used for rounding must be specified.<br>The following rounding methods exist:<br>ROUND (decimal places < 5 rounded down, >= 5 rounded up)<br>CEIL (rounding up to the next whole number regardless of the value of the decimal place)<br>FLOOR (rounding down to the current whole number regardless of the value of the decimal place)<br>The default value is ROUND. | |
| Calculation (PPM4 only) | Result | The rounded value in the specified scale. |
| | Error | If operand is of an invalid data type or an invalid number of operands. |
| Example: | `<round scale="MINUTE" roundingkind="CEIL">`<br>`  <constant>`<br>`  <dataitem value="4284.0">`<br>`  1,19`<br>`  <datatype name="TIMESPAN">`<br>`  Time span`<br>`  </datatype>`<br>`  <scale name="HOUR" factor="3600.0">`<br>`  Hours`<br>`  </scale>`<br>`  </dataitem>`<br>`  </constant>`<br>`</round>`<br>The time span value 1.19 hours is converted to 71.4 minutes as specified by the scale and is rounded up to 72 minutes (return value) in line with the specified rounding method. | |

## NULL VALUE

| | |
|---|---|
| XML tag: | nullvalue |
| Operands: | exactly two values |
| Synopsis: | `<nullvalue>`<br>`  <value 1>`<br>`  <value 2>`<br>`</nullvalue>` |
| Operands: | Any data type, both operands must be of the same data type. |
| Result: | Value of the first operand if it is not null, otherwise value of the second operand |

| | | |
|---|---|---|
| Result type: | Operand data type | |
| Description: | Replaces the possibly missing value of the first operator (value null) with the value of the second operator. If the first operand supplies a value, this value will be returned, otherwise the value of the second operand will be returned.<br>If both operators do not supply any value, null is returned as a value. This means that the second operator should always supply a value. | |
| Calculation (PPM4 only) | Result | Value of the first operand if it is not null, otherwise value of the second operand. |
| | Error | When operands have different data types |
| Example: | `<nullvalue>`<br>`  <subtext beginindex="3">`<br>`  <filteredattribute name="AT_XYZ"↵`<br>`  nodetype="PROCESS"/>`<br>`  </subtext>`<br>`  <constant>`<br>`  <dataitem>`<br>`  ABC`<br>`  <datatype name="TEXT">Text</datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`</nullvalue>`<br>If the **subtext** operator does not return any value, the constant character string **ABC** is returned. | |

## 8.1.3.4.2 Operators resulting in a set of values

The following operators are available: set, union, intersect, removeduplicates.

### SET CREATION

| | |
|---|---|
| XML tag: | Set |
| Operands: | at least one value |
| Synopsis: | `<set>`<br>`  <value 1>`<br>`  ...`<br>`  <value n>`<br>`</set>` |
| Operands: | All data types, but for non-numerical data types, a uniform data type within the list of operands is necessary. Different numerical data types are automatically converted into the DOUBLE data type. |
| Result: | Set of values |

| | | |
|---|---|---|
| Result type: | DOUBLE for mixed numerical operands, data type of first operand for non-numerical data types | |
| Description: | Creates a set of values from the specified values. | |
| Calculation (PPM3/PPM4) | Result | Empty set if all operands return NULL, that is, the result set never contains NULL |
| | Error | If at least one operand is of an invalid data type (set of values or data type not identical with first operand) |
| Example: | `<set>`<br>`  <constant>`<br>`  <dataitem value="2">`<br>`  <datatype name="DOUBLE"/>`<br>`  </dataitem>`<br>`  </constant>`<br>`  <value 1>`<br>`  <value 2>`<br>`  <value n>`<br>`</set>` | |

## SET UNION

| | | |
|---|---|---|
| XML tag: | Union | |
| Operands: | At least two sets of values (**<attribute ... />** or **<set>...</set>** or **<union>...</union>** or **<intersect>...</intersect>**) | |
| Synopsis: | `<union>`<br>`  <Set of values 1>`<br>`  <Set of values 2>`<br>`  ...`<br>`  <Set of values n>`<br>`</union>` | |
| Operands: | all data types, but not a mixture of numerical and non-numerical data types | |
| Result: | Set of values | |
| Result type: | DOUBLE for mixed numerical operands, data type of first operand for non-numerical data types | |
| Description: | Creates the set union of the specified sets of values. | |
| Calculation (PPM3) | Result | Empty set if all operands are empty sets |
| | Error | If at least one operand is of an invalid data type or at least one operand equals NULL |
| Calculation (PPM4) | Result | Empty set if all operands are empty sets. NULL if at least one operand equals NULL. |

| | | |
|---|---|---|
| | Error | If at least one operand is of an invalid data type |
| Example: | <union><br><attribute name="AT_START_TIME" nodetype="PROCESS"/><br><attribute name="AT_END_TIME" nodetype="PROCESS"/><br><attribute name="AT_START_TIME" nodetype="OT_FUNC"/><br><attribute name="AT_END_TIME" nodetype="OT_FUNC"/><br></union> | |

## INTERSECTION

| | | |
|---|---|---|
| XML tag: | Intersect | |
| Operands: | at least two sets of values | |
| Synopsis: | <intersect><br><Set of values 1><br><Set of values 2><br>...<br><Set of values n><br></intersect> | |
| Operands: | all data types, but not a mixture of numerical and non-numerical data types | |
| Result: | Set of values containing all elements contained in all initial sets | |
| Result type: | DOUBLE for mixed numerical operands,<br>data type of first operand for unmixed data types | |
| Description: | Creates the intersection of the specified sets of values. | |
| Calculation (PPM3) | Result | Empty set if one operand is an empty set. |
| | Error | If at least one operand is of an invalid data type or at least one operand equals NULL |
| Calculation (PPM4) | Result | Empty set if one operand is an empty set.<br>NULL if at least one operand equals null. |
| | Error | If at least one operand is of an invalid data type |
| Example: | - | |

## DUPLICATE REMOVER

| | |
|---|---|
| XML tag: | Removeduplicates |
| Operands: | exactly one set of values |

| | | |
|---|---|---|
| Synopsis: | `<removeduplicates>`<br>`  <Set of values>`<br>`</removeduplicates>` | |
| Operands: | Any data types | |
| Result: | Set of values | |
| Result type: | Operand data type | |
| Description: | Removes elements with identical values from a set of values. | |
| Calculation (PPM4 only) | Result | Set of values containing all elements contained in the initial set, but each one only once. Empty set if operand is an empty set. |
| | Error | If at least one element in the set of values is of an invalid data type |
| Example: | Counting the plants involved in the process:<br>`<card>`<br>`  <removeduplicates>`<br>`  <attribute name="AT_WERK" nodetype"OT_FUNC" />`<br>`  </removeduplicates>`<br>`</card>` | |

## 8.1.3.4.3 Operators producing a value

The following operators are available: sum, product, card, min, max, mean, convert.

### SUM

| | |
|---|---|
| XML tag: | Sum |
| Operands: | exactly one set of values |
| Synopsis: | `<sum>`<br>`  <Set of values>`<br>`</sum>` |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) |
| Result: | Value |
| Result type: | Data type of set of values used, for mixed data types within set always DOUBLE |
| Description: | Creates the sum of all elements in the set of values. |

| | | |
|---|---|---|
| Calculation (PPM3/PPM4) | Result | Sum of values contained in the set of values. NULL if the transferred set is empty. |
| | Error | If at least one element in the set of values is of an invalid data type. |
| Example: | - | |

## PRODUCT

| | | |
|---|---|---|
| XML tag: | Product | |
| Operands: | exactly one set of values | |
| Synopsis: | `<product>`<br>`  <Set of values>`<br>`</product>` | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value | |
| Result type: | Data type of set of values used, for mixed data types within set always DOUBLE | |
| Description: | Creates the product of all elements in the set of values. | |
| Calculation (PPM3/PPM4) | Result | Multiplication of values contained in the set of values. NULL if the transferred set is empty. |
| | Error | If at least one element in the set of values is of an invalid data type. |
| Example: | - | |

## CARDINALITY

| | |
|---|---|
| XML tag: | Card |
| Operands: | exactly one set of values |
| Synopsis: | `<card>`<br>`  <Set of values>`<br>`</card>` |
| Operands: | All data types of the set of values specified |
| Result: | Value |
| Result type: | always LONG |

| | | |
|---|---|---|
| Calculation (PPM3/PPM4) | Result | Calculates the total number of elements in the set of values. For an empty set, the return value is **0**. |
| | Error | None |
| Example: | - | |

## MINIMUM

| | | |
|---|---|---|
| XML tag: | Min | |
| Operands: | exactly one set of values | |
| Synopsis: | `<min>`<br>`  <Set of values>`<br>`</min>` | |
| Operands: | Numerical data types and TIME (TIMESTAMP, DATE), DAY, TIMEOFDAY | |
| Result: | Value | |
| Result type: | Data type of set of values | |
| Description: | Returns the smallest value in the set of values. | |
| Calculation (PPM3/PPM4) | Result | NULL if the set of values is empty |
| | Error | None |
| Example: | - | |

## MAXIMUM

| | | |
|---|---|---|
| XML tag: | Max | |
| Operands: | exactly one set of values | |
| Synopsis: | `<max>`<br>`  <Set of values>`<br>`</max>` | |
| Operands: | Numerical data types and TIME (TIMESTAMP, DATE), DAY, TIMEOFDAY | |
| Result: | Value | |
| Result type: | Data type of set of values | |
| Description: | Returns the greatest value in the set. | |
| Calculation (PPM3/PPM4) | Result | NULL if the set of values is empty |
| | Error | None |

| | |
|---|---|
| Example: | - |

## MEAN

| | | |
|---|---|---|
| XML tag: | Mean | |
| Operands: | exactly one set of values | |
| Synopsis: | `<mean>`<br>`  <Set of values>`<br>`</mean>` | |
| Operands: | Numerical data types (LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, FREQUENCY, PERCENTAGE) | |
| Result: | Value | |
| Result type: | Operand data type | |
| Calculation (PPM3) | Result | Mean of the numerical values contained in the set. NULL if operand is equal to an empty set. |
| | Error | If the data type is invalid or at least one operand equals NULL. |
| Calculation (PPM4) | Result | Mean of the numerical values contained in the set. NULL if operand is an empty set or equals NULL. |
| | Error | If at least one element in the set of values is of an invalid data type. |
| Example: | `<mean>`<br>`  <union>`<br>`  <attribute name="AT_ANZAHL_POS1" nodetype="OT_FUNC"/>`<br>`  <attribute name="AT_ANZAHL_POS2" nodetype="OT_FUNC"/>`<br>`  </union>`<br>`</mean>` | |

## DATA TYPE CONVERSION

| | |
|---|---|
| XML tag: | convert |

| | |
|---|---|
| Operands: | exactly one value |
| Synopsis: | `<convert datatype="...">`<br>`  <value>`<br>`</convert>` |
| Operands: | TEXT or numerical data type, returns the input value for the conversion. |
| Attribute: | The **datatype** attribute specifies the data type into which the input value is to be converted. |
| Result: | Value converted into the data type specified |
| Result type: | LONG, DOUBLE, FREQUENCY, BOOLEAN, TEXT, TIME, TIMESPAN, FACTORYTIMESPAN, DAY, PERCENTAGE |

| | |
|---|---|
| Description: | Conversion of a numerical data type (for example, LONG, DOUBLE, TIMESPAN, FACTORYTIMESPAN, PERCENTAGE) into another numerical data type.<br>Conversion of the TEXT data type into one of these data types: LONG, DOUBLE, BOOLEAN, TIMESPAN, or FREQUENCY.<br>After the conversion, the result is written in base scaling to the result attribute.<br>The internal PPM format is used for conversion. You cannot specify a custom format.<br>Conversion of the LONG data type to TEXT, with leading zeros and separators being removed. The result of the conversion is the converted number without separators in one string. In the example below, the LONG value 000300080191 is converted into the TEXT value 300080191.<br>`<convert datatype="TEXT">`<br>`  <constant>`<br>`  <dataitem>`<br>`  000300080191`<br>`  <datatype name="LONG"/>`<br>`  </dataitem>`<br>`  </constant>`<br>`</convert>`<br>It is possible to convert any data type to TEXT, for example:<br>▪ CONVERT(DOUBLE(-300080191)) -> TEXT("3.00080191E8")<br>▪ CONVERT(TIME(07.01.1971 00:01)) -> TEXT("7.1.1971 0:01")<br>▪ CONVERT(TIME(07.01.2000)) -> TEXT("07.1.1971")<br>▪ CONVERT(BOOLEAN(1)) -> TEXT("FALSE") (everything that is not true is false)<br>The scaling used for the output corresponds to the one at the object, for example:<br>▪ convert(<dataitem>1 YEAR<datatype name='TIMESPAN'/></dataitem>) -> TEXT("1.0 YEAR")<br>▪ convert(<dataitem>1 YEAR<datatype name='TIMESPAN'/><scale name='MONTH'/></dataitem>) -> TEXT("12.166666666666666 MONTH") (== 365 days/30 days) |

| | | |
|---|---|---|
| Calculation (PPM4 only) | Result | Returns the converted value of the operand. NULL if the operand returns NULL |
| | Error | If conversion fails |
| Example: | <convert datatype="LONG"><br>  <filteredattribute name="AT_ABC"<br>  nodetype="PROCESS"/><br></convert><br>An assumed value of **456** for the **AT_ABC** attribute of **TEXT** type is converted to the **LONG** data type. | |

## 8.1.3.4.4 Logical operators

The following operators are available: eq, eqset, lt, gt, gteq, lteq, ne, exists, filled, in, and, or, xor, not, containstext.

### EQUALITY (VALUE)

| | | |
|---|---|---|
| XML tag: | eq | |
| Operands: | at least two values | |
| Synopsis: | <eq><br>  <Value 1><br>  <Value 2><br>  ...<br>  <Value n><br></eq> | |
| Operands: | All data types | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | All specified values equal returns TRUE. | |
| Calculation (PPM3) | Result | TRUE if operands 1 to n have the same data type and value, otherwise FALSE. |
| | Error | If one operand is NULL. |
| Calculation (PPM4) | Result | TRUE if operands 1 to n have the same data type and value, otherwise FALSE. NULL if an operand is NULL. |
| | Error | None |

| | |
|---|---|
| Example: | <eq><br>  <timespan type="NORMAL">  <max>  <attribute name="AT_CUSTDATE_WISH" nodetype="PROCESS"  onerror="EXIT_NO_WARNING"/>  </max>  <min>  <attribute name="AT_END_TIME" nodetype="OT_FUNC"  objectname="SAP.WAUS" onerror="EXIT_NO_WARNING"/>  </min>  </timespan>  <constant>  <dataitem value="0">  <datatype name="TIMESPAN"/>  </dataitem>  </constant><br></eq> |

## EQUALITY (VALUE)

| | | |
|---|---|---|
| XML tag: | ne | |
| Operands: | at least two values | |
| Synopsis: | <ne><br>  <Value 1><br>  <Value 2><br>  ...<br>  <Value n><br></ne> | |
| Operands: | All data types | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | Inequality of all specified values returns TRUE | |
| Calculation (PPM4) | Result | TRUE if all operands 1 to n are not equal (for example, Operand 1 != Operand 2), otherwise FALSE. NULL if an operand is NULL. |
| | Error | None |

| | |
|---|---|
| Example: | <ne><br>  <timespan type="NORMAL">  <max>  <attribute name="AT_CUSTDATE_WISH"  nodetype="PROCESS"  onerror="EXIT_NO_WARNING"/>  </max>  <min>  <attribute name="AT_END_TIME"  nodetype="OT_FUNC"  objectname="SAP.WAUS"  onerror="EXIT_NO_WARNING"/>  </min>  </timespan>  <constant>  <dataitem value="0">  <datatype name="TIMESPAN"/>  </dataitem>  </constant><br></ne> |

## EQUALITY (SET OF VALUES)

| | | |
|---|---|---|
| XML tag: | eqset | |
| Operands: | at least two sets of values | |
| Synopsis: | <eqset><br>  <Set of values 1><br>  <Set of values 2><br>  ...<br>  <Set of values n><br></eqset> | |
| Operands: | All data types | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | Equality of all specified sets of values returns TRUE. Operands 2 to n are compared one by one with operand 1. | |
| Calculation (PPM3) | Result | TRUE if the sets to be compared are of equal size and all their values are identical, otherwise FALSE. |
| | Error | If one operand is NULL. |
| Calculation (PPM4) | Result | TRUE if the sets to be compared are of equal size and all their objects are identical, otherwise FALSE. NULL if an operand is NULL. |
| | Error | None |
| Example: | - | |

## "LESS THAN" COMPARISON

| | | |
|---|---|---|
| XML tag: | lt | |
| Operands: | exactly two values | |
| Synopsis: | `<lt>`<br>`  <Value 1>`<br>`  <Value 2>`<br>`</lt>` | |
| Operands: | Uniform numerical data type | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Calculation (PPM3) | Result | TRUE if operand 1 and operand 2 are of the same data type and operand 1 is less than operand 2, otherwise FALSE. |
| | Error | If at least one operand equals NULL or at least one is of an invalid data type |
| Calculation (PPM4) | Result | TRUE if operand 1 and operand 2 are of the same data type and operand 1 is less than operand 2, otherwise FALSE. NULL if at least one operand equals NULL. |
| | Error | Only if data type is invalid |
| Example: | - | |

## "GREATER THAN" COMPARISON

| | | |
|---|---|---|
| XML tag: | gt | |
| Operands: | exactly two values | |
| Synopsis: | `<gt>`<br>`  <Value 1>`<br>`  <Value 2>`<br>`</gt>` | |
| Operands: | Uniform numerical data type | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Calculation (PPM3) | Result | Returns TRUE if value 1 is greater than value 2 and the operands are of a uniform data type, otherwise FALSE. |
| | Error | If at least one operand equals NULL or at least one operator is of an invalid data type. |

| | | |
|---|---|---|
| Calculation (PPM4) | Result | Returns TRUE if value 1 is greater than value 2 and the operands are of a uniform data type, otherwise FALSE. NULL if at least one operand equals NULL. |
| | Error | Only if data type is invalid |
| Example: | - | |

## "GREATER THAN OR EQUAL" COMPARISON

| | | |
|---|---|---|
| XML tag: | gteq | |
| Operands: | exactly two values | |
| Synopsis: | `<gteq>`<br>`  <Value 1>`<br>`  <Value 2>`<br>`</gteq>` | |
| Operands: | Uniform numerical data type | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Calculation (PPM4) | Result | Returns TRUE if value 1 is greater than or equal to value 2 and the operands are of a uniform data type, otherwise FALSE. NULL if at least one operand equals NULL. |
| | Error | Only if data type is invalid |
| Example: | - | |

## "LESS THAN OR EQUAL" COMPARISON

| | |
|---|---|
| XML tag: | lteq |
| Operands: | exactly two values |
| Synopsis: | `<glteq>`<br>`  <Value 1>`<br>`  <Value 2>`<br>`</lteq>` |
| Operands: | Uniform numerical data type |
| Result: | Logical value |
| Result type: | BOOLEAN |

| | | |
|---|---|---|
| Calculation (PPM4) | Result | Returns TRUE if value 1 is less than or equal to value 2 and the operands are of a uniform data type, otherwise FALSE. NULL if at least one operand equals NULL. |
| | Error | Only if data type is invalid |
| Example: | - | |

## EXISTENCE CHECK

| | | |
|---|---|---|
| XML tag: | exists | |
| Operands: | at least one attribute name (attribute, filteredattribute) | |
| Synopsis: | `<exists>`<br>`  <Attribute 1>`<br>`  <Attribute 2>`<br>`  ...`<br>`  <Attribute n>`<br>`</exists>` | |
| Operands: | All data types | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | Returns TRUE if the specified attributes exist, regardless of whether any values are assigned to the attributes. | |
| Calculation (PPM3/PPM4) | Result | TRUE if all specified attributes exist, otherwise FALSE |
| | Error | None |
| Example: | `<exists>`<br>`  <attribute name="AT_ORDER_VOL" nodetype="OT_FUNC"/>`<br>`</exists>` | |

## CONTENT CHECK

| | |
|---|---|
| XML tag: | filled |

| | | |
|---|---|---|
| Operands: | at least one value or a set of values | |
| Synopsis: | <filled><br>  <Value 1><br>  <Value 2><br>  ...<br>  <Value n><br></filled><br>or<br><filled><br>  <Set of values 1><br>  <Set of values 2><br>  ...<br>  <Set of values n><br></filled><br>or<br><filled><br>  <Value 1><br>  <Value 2><br>  ...<br>  <Value n><br>  <Set of values 1><br>  <Set of values 2><br>  ...<br>  <Set of values n><br></filled> | |
| Operands: | All data types | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Calculation (PPM3/PPM4) | Result | TRUE if all relevant values or sets of values are specified, otherwise FALSE |
| | Error | None |
| Example: | - | |

## CONTENT CHECK OF SETS

| | |
|---|---|
| XML tag: | in |
| Operands: | 1. operand: Value or set of values<br>Operand 2: Set of values |

| | | |
|---|---|---|
| Synopsis: | <in><br>  <Value 1><br>  <Set of values 2><br></in><br>or<br><in><br>  <Set of values 1><br>  <Set of values 2><br></in> | |
| Operands: | All data types | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Calculation (PPM4 only) | Result | TRUE if the value or set of values of the first operand is contained in the set of values specified by the second operand.<br>NULL if one operand returns NULL. |
| | Error | Only if data types are incompatible |
| Example: | <calcattr name="AT_KI_ABL" type="PROCESS">  <calculation>  <in>  <constant>  <dataitem>  HR-ABL  <datatype name="TEXT">Text</datatype>  </dataitem>  </constant>  <attribute name="AT_HRMODUL" nodetype="OT_FUNC" />  </in>  </calculation> </calcattr><br>The **in** operator returns TRUE if there is an **AT_HRMODUL** attribute with the value **HR-ABL** for at least one function in the EPC. | |

## LOGICAL AND

| | |
|---|---|
| XML tag: | and |
| Operands: | at least two logical values |

| | | |
|---|---|---|
| Synopsis: | <and><br>  <Logical value 1><br>  <Logical value 2><br>  ...<br>  <Logical value n><br></and> | |
| Operands: | BOOLEAN | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | Returns TRUE, if all logical values are TRUE. The first time an operand returns FALSE, evaluation of the operand list is canceled and FALSE is returned. | |
| Calculation (PPM3) | Result | TRUE if all operands return TRUE, otherwise FALSE |
| | Error | If one operand returns NULL or at least one operand is of an invalid data type |
| Calculation (PPM4) | Result | TRUE if all operands return TRUE.<br>FALSE if one operand returns FALSE and all preceding operands return TRUE.<br>NULL if one operand returns NULL and all preceding operands return TRUE. |
| | Error | If at least one operand is of an invalid data type (not BOOLEAN) |
| Example: | - | |

## LOGICAL OR

| | |
|---|---|
| XML tag: | or |
| Operands: | at least two logical values |
| Synopsis: | <or><br>  <Logical value 1><br>  <Logical value 2><br>  ...<br>  <Logical value n><br></or> |
| Operands: | BOOLEAN |
| Result: | Logical value |
| Result type: | BOOLEAN |

| | | |
|---|---|---|
| Description: | Returns TRUE if at least one logical value is TRUE. The first time an operand returns TRUE, evaluation of the operand list is canceled and TRUE is returned. | |
| Calculation (PPM3) | Result | TRUE if one operand returns TRUE and all preceding operands return NULL, otherwise FALSE. |
| | Error | If one operand returns NULL and all other operands return FALSE, or if at least one operand is of an invalid data type |
| Calculation (PPM4) | Result | TRUE if one operand returns TRUE and all preceding ones do not return NULL. FALSE if all operands return FALSE. NULL if one operand returns NULL and all preceding operands return FALSE. |
| | Error | If at least one operand is of an invalid data type (not BOOLEAN) |
| Example: | - | |

## LOGICAL EXCLUSIVE OR

| | | |
|---|---|---|
| XML tag: | xor | |
| Operands: | at least two logical values | |
| Synopsis: | `<xor>`<br>`  <Logical value 1>`<br>`  <Logical value 2>`<br>`  ...`<br>`  <Logical value n>`<br>`</xor>` | |
| Operands: | BOOLEAN | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | Returns TRUE if exactly one logical value is TRUE. | |
| Calculation (PPM3) | Result | TRUE if exactly one operand returns TRUE, otherwise FALSE |
| | Error | If one operand returns NULL or at least one operand is of an invalid data type |

| | | |
|---|---|---|
| Calculation (PPM4) | Result | TRUE if exactly one operand returns TRUE. FALSE if no operand or more than one operand returns TRUE. NULL if at least one operand returns NULL |
| | Error | If at least one operand is of an invalid data type (not BOOLEAN) |
| Example: | - | |

## LOGICAL NOT

| | | |
|---|---|---|
| XML tag: | not | |
| Operands: | exactly one logical value | |
| Synopsis: | `<not>`<br>`  <Logical value>`<br>`</not>` | |
| Operands: | BOOLEAN | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Description: | Reverses the specified logical value. | |
| Calculation (PPM3) | Result | TRUE if operand returns FALSE, otherwise FALSE |
| | Error | If operand returns NULL or at least one operand is of an invalid data type |
| Calculation (PPM4) | Result | TRUE if operand returns FALSE. FALSE if operand returns TRUE. NULL if the operand returns NULL |
| | Error | If operand is of an invalid data type (not BOOLEAN) |
| Example: | `<not>`<br>`  <exists>`<br>`  <attribute name="AT_ORDER_VOL" nodetype="OT_FUNC"/>`<br>`  </exists>`<br>`</not>` | |

## CHECKING FOR TEXT WITHIN TEXT

| | | |
|---|---|---|
| XML tag: | containstext | |
| Operands: | Exactly two values of type TEXT | |
| Synopsis: | `<containstext>`<br>`  <value 1>`<br>`  <value 2>`<br>`</containstext>` | |
| Operands: | TEXT | |
| Result: | Logical value | |
| Result type: | BOOLEAN | |
| Calculation (PPM4 only) | Result | TRUE if the text returned by the second operand is a sub-character string of the value returned by the first operand, otherwise FALSE. NULL if at least one operand returns NULL |
| | Error | If at least one operand is of an invalid data type (not TEXT) |
| Example: | `<containstext>`<br>`  <filteredattribute name="AT_ABCDEF"`<br>`  nodetype="OT_FUNC" objectname="this"`<br>`  onerror="EXIT_NO_WARNING"/>`<br>`  <constant>`<br>`  <dataitem>`<br>`  abc`<br>`  <datatype name="TEXT">Text</datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`</containstext>`<br>Operator returns **TRUE** if the string **abc** is contained in the value of the **AT_ABCDEF** attribute. | |

# 8.1.3.4.5 Conditional operator

The following operator is available: if - then - else

## CONDITION CHECK WITH OPTIONAL BRANCH (ELSE)

| | | |
|---|---|---|
| XML tag: | if - then [- else] | |
| Operands: | exactly one logical value | |
| Synopsis: | `<if>`<br>`  <Logical value>`<br>`</if>`<br>`<then>`<br>`  <Value>`<br>`</then>`<br>`<else>`<br>`  <Value>`<br>`</else>` | |
| Operands: | BOOLEAN | |
| Result: | Logical value - Value [- Value] | |
| Result type: | BOOLEAN - Operand data type [- Operand data type] | |
| Calculation (PPM3/PPM4) | Result | Value of second operand if the first operand returns TRUE. Value of the 3rd operand if the 1st operand returns FALSE.<br>NULL if 1st operand is FALSE and 3rd operand is not defined. |
| | Error | If first operand is not of the BOOLEAN data type |

| | |
|---|---|
| Example: | <if><br><exists><br><filteredattribute name="AT_OS" nodetype="OT_FUNC" objectname="SAP.WAUS filter="LATEST"/><br></exists><br><then><br><filteredattribute name="AT_CT" nodetype="OT_FUNC" filter="EARLY"/><br></then><br><else><br><filteredattribute name="AT_KT" nodetype="OT_FUNC" filter="LATEST"/><br></else><br></if><br>If the condition is met (i. e. the **AT_OS** attribute is specified for at least one function with the internal name **SAP.WAUS**) the value of the **AT_CT** attribute is passed on. If the condition is not met, the value of the **AT_KT** attribute is used in the subsequent calculation. |
| | |

## 8.1.3.4.6 String operators

The following operators are available: concat, subtext, indexof.

### CONCATENATION OF STRINGS

| | | |
|---|---|---|
| XML tag: | Concat | |
| Operands: | exactly one set of values (strings) | |
| Synopsis: | <concat><br><Value set 1><br></concat> | |
| Operands: | TEXT | |
| Result: | Value (string) | |
| Result type: | TEXT | |
| Calculation (PPM3/PPM4) | Result | Result of the concatenation of all strings contained in the set of values. NULL if operand is an empty set |
| | Error | If at least one value is of an invalid data type |
| Example: | - | |

## EXTRACTION OF SUBSTRINGS

| | | |
|---|---|---|
| XML tag: | Subtext | |
| Operands: | exactly one value (string) | |
| Synopsis: | `<subtext beginindex="..." [endindex="  "]>`<br>`  <Value>`<br>`</subtext>`<br>Example<br>`<subtext>`<br>`  <Value>`<br>`  <beginindex>...</beginindex>`<br>`  <endindex>...</endindex>`<br>`</subtext>`<br>You should only use one of the two variations outlined (index specified either as an XML attribute or an XML element). | |
| Operands: | TEXT | |
| Result: | Value (extracted string) | |
| Result type: | TEXT | |
| Description: | Extracts a substring from a string by specifying positive indices (from the start of the string) or negative indices (from the end of the string). | |
| Calculation (PPM3) | Result | Returns a substring of the string transferred by the operand.<br>NULL if the specified indices are invalid |
| | Error | If operand is NULL or of an invalid data type |
| Calculation (PPM4) | Result | Returns a substring of the string transferred by the operand.<br>NULL if the specified indices are invalid or the operand is NULL |
| | Error | If data type is invalid (not TEXT) |

| | |
|---|---|
| Example: | `<subtext beginindex="-3" endindex="-1">`<br>`  <filteredattribute name="AT_XYZ"`<br>`  nodetype="OT_FUNC"/>`<br>`</subtext>`<br>Example<br>`<subtext>`<br>`  <beginindex>-3</beginindex>`<br>`  <endindex>-1</endindex>`<br>`  <filteredattribute name="AT_XYZ"`<br>`  nodetype="OT_FUNC"/>`<br>`</subtext>`<br>Assuming the value **ABCDE** for the **AT_XYZ** attribute, the substring **CD** is extracted. |

## IDENTIFY POSITION OF SUB-TEXT IN ANOTHER TEXT

| | | |
|---|---|---|
| XML tag | <indexof> | |
| Description | Returns the index in a text (operand 1), where the first occurrence of a sub-text (operand 2) is located, starting at a specified index (operand 3). | |
| Operands (Position/ Data type) | 1 / TEXT<br>2 / TEXT<br>[ 3 / LONG ], optional, default value is 0 | |
| Calculation | from PPM 9.0 | |
| Calculation (PPM4) | Result | -1, if at least 1 operand equals NULL, or if the substring is not found<br>Index of the first occurrence of the text determined by operand 2 in the text determined by operand 1 starting from the index specified in operand 3 (like Java String.indexOf(String, int)). |
| | Data type | LONG |
| | Exception | If at least 1 operand has an invalid data type or operand 3 < 0 |
| | | |
| | | |
| | | |

| | |
|---|---|
| Example | `<subtext mode="PPM4">`<br>`  <filteredattribute name="AT_TEXT" … />`<br>`  <beginindex>`<br>`  <indexof>`<br>`  <filteredattribute name="AT_TEXT" … />`<br>`  <constant>`<br>`  <dataitem>`<br>`  XYZ`<br>`  <datatypename="TEXT">Text</datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`  </beginindex>`<br>`</subtext>` |
| | In the example, the operators **indexof** and **subtext** are used together. indexof determines the position of the text **XYZ** in the attribute value of the attribute **AT_TEXT**. Then, the **subtext** operator identifies the substring from this position.<br>If the **AT_TEXT** attribute has the value **ABCDEXYZAC**, the above calculation rule would return the value **XYZAC**. |

## 8.1.3.4.7 Time operators

The following operators are available: createday, createtimeofday, createtimestamp, addtimespan, addfactorytimespan, and weekday

### FORMAT CONVERSION (DATE)

| | | |
|---|---|---|
| XML tag: | createday | |
| Operands: | exactly one value | |
| Synopsis: | `<createday>`<br>`  <Value>`<br>`</createday>` | |
| Operands: | TIME (TIMESTAMP, DATE) | |
| Result: | Value (date in **dd.MM.yyyy** format) | |
| Result type: | DAY | |
| Description: | Extracts a date from a PPM time stamp. | |
| Calculation (PPM3) | Result | Date returned by the operand |
| | Error | If operand is of an invalid data type or equal to NULL |
| Calculation (PPM4) | Result | Date returned by the operand NULL if operand is equal to NULL |
| | Error | If operand is of an invalid data type |

| | |
|---|---|
| Example: | `<calcattr name="AT_DAY" type="PROCESS">`<br>`  <calculation>`<br>`  <createday>`<br>`  <filteredattribute name="AT_TIME"`<br>`  nodetype="OT_FUNC"`<br>`  objectname="this" filter="EARLY"/>`<br>`  </createday>`<br>`  </calculation>`<br>`</calcattr>` |

## FORMAT CONVERSION (TIME)

| | | |
|---|---|---|
| XML tag: | createtimeofday | |
| Operands: | exactly one value | |
| Synopsis: | `<createtimeofday>`<br>`  <Value>`<br>`</createtimeofday>` | |
| Operands: | TIME (TIMESTAMP, DATE) | |
| Result: | Value (time of the day in **hh:mm:ss** format) | |
| Result type: | TIMEOFDAY | |
| Description: | Extracts the time of day from a PPM time stamp. | |
| Calculation (PPM3) | Result | Time of the day defined by the operand |
| | Error | If operand is of an invalid data type or equal to NULL |
| Calculation (PPM4) | Result | Time of the day defined by the operand NULL if the operand returns NULL |
| | Error | If operand is of an invalid data type |
| Example: | `<calcattr name="AT_DAY" type="PROCESS">`<br>`  <calculation>`<br>`  <createtimeofday>`<br>`  <filteredattribute name="AT_TIME"`<br>`  nodetype="OT_FUNC"`<br>`  objectname="this" filter="EARLY"/>`<br>`  </createtimeofday>`<br>`  </calculation>`<br>`</calcattr>` | |

## FORMAT CONVERSION (TIME STAMP)

| | |
|---|---|
| XML tag: | createtimestamp |

| | | |
|---|---|---|
| Operands: | one or two values (**Date** or **Date and time**) | |
| Synopsis: | `<createtimestamp>`<br>`  <Date>`<br>`  <Time> [optional]`<br>`</createtimestamp>` | |
| Operands: | DAY, TIMEOFDAY | |
| Result: | Value (time stamp in **dd.MM.yyyy hh:mm:ss** format) | |
| Result type: | TIME (TIMESTAMP, DATE) | |
| Description: | Creates a PPM time stamp from a date or from a date and a time. | |
| Calculation (PPM3) | Result | Time stamp defined by the operands |
| | Error | If at least one operand is of an invalid data type or equal to NULL |
| Calculation (PPM4) | Result | Time stamp defined by the operands. NULL if operand of DAY type returns NULL, or if first operand of TIMEOFDAY data type and second operand return NULL. |
| | Error | If at least one operand is of an invalid data type |
| Example: | `<createtimestamp>`<br>`  <constant>`<br>`  <dataitem>`<br>`  <datatype name="DAY">`<br>`  25.01.2004`<br>`  </datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`</createtimestamp>`<br>Creates the time stamp **25.01.2004 00:00:00**. | |

## ADDITION OF A TIME SPAN

| | |
|---|---|
| XML tag: | addtimespan |
| Operands: | exactly two values (time stamp and time span, date and time span or time and time span) |
| Synopsis: | `<addtimespan>`<br>`  <Time stamp or date or time>`<br>`  <Time span>`<br>`<addtimespan>` |
| Operands: | Operand 1: TIME (TIMESTAMP, DATE) or DAY or TIMEOFDAY<br>Operand 2: TIMESPAN |

| | | |
|---|---|---|
| Result: | Value (time stamp in **dd.MM.yyyy hh:mm:ss** format) | |
| Result type: | Point in time: TIME (TIMESTAMP, DATE) or DAY or TIMEOFDAY | |
| Description: | Adds a time span in the base scaling (SECOND) to a PPM time stamp. The result is a time stamp. | |
| Calculation (PPM3) | Result | Point in time resulting from adding the specified time span (operand 2) to the specified point in time (operand 1) |
| | Error | If at least one operand equals NULL or at least one operand is of an invalid data type |
| Calculation (PPM4) | Result | NULL if at least one operand is NULL Point in time resulting from adding the specified time span (operand 2) to the specified point in time (operand 1) |
| | Error | If data type is invalid |
| Example: | `<calcattr name="AT_NTOFD" type="PROCESS">`<br>`  <calculation>`<br>`  <addtimespan>`<br>`  <constant>`<br>`  <dataitem>`<br>`  <datatype name="TIMEOFDAY">`<br>`  08:35:41`<br>`  </datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`  <constant>`<br>`  <dataitem>`<br>`  <datatype name="TIMESPAN">`<br>`  -30 MINUTE`<br>`  </datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`  </addtimespan>`<br>`  </calculation>`<br>`</calcattr>`<br>At the specified time, a negative time span of thirty minutes is added. The result value **08:05:41** is saved in the **AT_NTOFD** target attribute. | |

## ADDITION OF A TIME SPAN INCLUDING FACTORY CALENDAR

Adds a factory calendar time span to a PPM time stamp. The result is a time stamp. Configuration and usage of the **addfactorytimespan** operator are similar as for **addtimespan**. For this calculation, the specified factory calendar time span is added beginning from a start time. By default, the operator supports only addition of positive factory calendar time spans. If you also want to calculate points in time in the past, you can add negative time spans by specifying the

optional XML attribute **negfactorytimespan="TRUE"** (default value: **FALSE**). If the point in time calculated is exactly on a work time limit the operator returns the earliest point in time possible.

#### Examples

Taking the simplified condition of a daily work time from 9am-5pm:

- addFactoryTimeSpan("01.12.2011 12:00:00", "8 FACTORY_HOUR") = "02.12.2011 12:00:00"
- addFactoryTimeSpan("01.12.2011 12:00:00", "5 FACTORY_HOUR") = "01.12.2011 17:00:00"
- addFactoryTimeSpan{negfactorytimespan="TRUE"}("02.12.2011 12:00:00", "-8 FACTORY_HOUR") = "01.12.2011 12:00:00"
- addFactoryTimeSpan{negfactorytimespan="TRUE"}("02.12.2011 12:00:00", "-3 FACTORY_HOUR") = "01.12.2011 05:00:00 PM"

If you want to use a factory calendar other than the default factory calendar (**factorycalendar.xml**) you can specify an XML file containing the factory calendar to be used. In the **attributename** XML attribute, you specify the function or process instance attribute that determines the name of the XML factory calendar file to be used. The attribute must be specified at the function or process instance for which the calculation is run. In the XML attribute **directory**, you specify the directory in which to look for the specified factory calendar file. The two XML attributes **attributename** and **directory** must always be specified together.

You specify the corresponding directory containing the factory calendar file to be used relative to the PPM data directory. The PPM data directory **data_ppm** is located under <PPM installation directory>\ppm\server\bin\work\.

#### Example

```
...
<addfactorytimespan directory="calc\fc" attributename="AT_FC_NAME">
...
```

If the **AT_FC_NAME** attribute contains the value **myFactoryCalendar.xml**, the factory calendar defined in the file **myFactoryCalendar.xml** is used for calculation. The file is located under **<PPM installation directory>\ppm\server\bin\work\data_ppm\calc\fc**.

The addition of factory calendar time spans is always in the base unit **Person-second**. The conversion factors used for this are independent of the factory calendar and defined in the client-specific configuration file **transformationfactors.xml**. If you do not want to use these, you may use only factory calendar time spans with the units person-second, minute, or hour to add a time span based on a factory calendar.

## DETERMINING THE DAY OF THE WEEK (FROM A DATE)

| | |
|---|---|
| XML tag: | weekday |
| Operands: | exactly one value |

| | | |
|---|---|---|
| Synopsis: | `<weekday>`<br>`  <Value>`<br>`</weekday>` | |
| Operands: | Exactly one operand: TIME or DAY | |
| Result: | Character string in the format **MO**, **TU**, **WE**, **TH**, **FR**, **SA**, or **SU** | |
| Result type: | TEXT | |
| Description: | Determines the day of the week from a PPM date type and returns it as a character string. | |
| Calculation (PPM4) | Result | One of the constants MO, TU, WE, TH, FR, SA or SU, depending on the day of the week of the date transferred. |
| | Error | If operand is of an invalid data type or an invalid number of operands. |
| Example: | `<calcattr name="AT_WEEKDAY" type="PROCESS">`<br>`  <calculation>`<br>`  <weekday>`<br>`  <constant>`<br>`  <dataitem>`<br>`  <datatype name="DAY">`<br>`  25.08.2007`<br>`  </datatype>`<br>`  </dataitem>`<br>`  </constant>`<br>`  </weekday>`<br>`  </calculation>`<br>`</calcattr>`<br>Determines the day of the week for the specified date ('Saturday') and returns it as the TEXT character string **SA**.<br>Values of text dimensions that use results of the **weekday** operator cannot be sorted. | |

## 8.1.3.4.8 Conditional attribute type calculation

The conditional calculation of attribute types allows attribute type calculation to be controlled. This control is based on the existence check for attribute types or the result of comparisons. The existence check distinguishes between the two cases of **Attribute type exists** (**exists** XML element) and **Attribute type specified** (**filled** XML element).

In the example below, the calculation element **<if>** returns the value **Null**, if there is no **AT_B** function attribute in the process instance. In this case, the set of results generated by the **attribute** XML element is empty.
If at least one **AT_B** attribute exists at any function in the process instance, the value of the **filteredattribute** XML element is transferred.

```
<if>
  <exists>
    <attribute name="AT_B" nodetype="OT_FUNC"/>
  </exists>
  <then>
    <filteredattribute name="AT_C" nodetype="OT_FUNC"
                                   filter="EARLY"/>
  </then>
</if>
```

By linking conditions using logical operators, more complex conditions can also be formulated. The example shown is to be expanded to include a test for an existing attribute value.

```
<if>
  <and>
    <exists>
      <attribute name="AT_B" nodetype="OT_FUNC"/>
    </exists>
    <filled>
      <attribute name="AT_B" nodetype="OT_FUNC"/>
    </filled>
  </and>
  <then>
    <filteredattribute name="AT_C" nodetype="OT_FUNC"
                                   filter="EARLY"/>
  </then>
</if>
```

As the existence of the corresponding attribute type is a prerequisite for an existing attribute value, the check for existence can be skipped to optimize the condition.

In the following example, the **<if>** calculation element returns the value **NULL** if an attribute type with the name **AT_G** is not specified for any of the occurring functions.

```
<if>
  <exists>
    <filteredattribute name="AT_G" nodetype="OT_FUNC"
                                  filter="LATEST"/>
  </exists>
</if>
```

### 8.1.3.5 Nesting of operators

Operators can be nested at any depth. If you are combining operators, you need to adhere to the rules specified in the DTD.

**Warning**

Calculation rules based on nesting of operators that is not permitted result in the import being canceled when the measure configuration is imported. Due to the complex dependencies, incorrect calculation rules may result in the database content being entirely unusable.

In the file **KeyindicatorConfiguration.dtd** in the **dtd** directory of your PPM installation, you can check what nesting of operators is permitted.

Example (extract from DTD):

```
<!ELEMENT abs (%numericoperator; | %setoperator; |
        %caseoperator; | filteredattribute | constant)>
```

The **<abs>** operator can be nested with one of the **<filteredattribute>** or **<constant>** XML elements or with an operator for the specified entities (declared units in XML notation to which particular XML elements are assigned):

- % numericoperator (unit of all mathematical operators)
- % setoperator (unit of all operators producing a value)
- % caseoperator (unit of all condition operators)

Which operators are assigned to which entity can be seen in the declaration of the entity.

Example (extract from DTD):

```
<!ENTITY % setoperator "sum|product|card|min|max|mean">
```

The **% setoperator** entity stands for one of the operators **<sum>**, **<product>**, **<card>**, **<min>**, **<max>**, or **<mean>**.

The following example shows a calculation rule compliant to the DTD:

```
<calcattr name="..." type="...">
  <calculation>
    <abs>
      <minus>
        <filteredattribute name="AT_KI_BSP1"
             nodetype="OT_FUNC" objectname="this"
             filter="LATEST" onerror="EXIT_NO_WARNING"/>
        <filteredattribute name="AT_KI_BSP2"
             nodetype="OT_FUNC" objectname="this"
              filter="EARLY" onerror="EXIT_NO_WARNING"/>
      </minus>
    </abs>
  </calculation>
</calcattr>
```

## 8.1.3.6 Calculation functions

Define complex partial calculations for calculation rules that you want to use in several attribute calculations to be used as calculation functions. A calculation function is used by calling up **usefunction** in the calculation rule for an attribute calculation or calculation function.

**Warning**

When calling up calculation functions from other calculation functions, avoid cyclic dependencies. The import of this kind of measure configuration is canceled and an error message is output.

| XML tag | Description |
|---|---|
| **function name** | Internal name of the calculation function. Referenced in the function call. |
| **Resulttype** | Result type (for use with other operators). Valid values:<br>Value (**VALUE**)<br>Set of values (**VALUELIST**)<br>Logical value (**BOOLEAN**) |
| **Datatype** | Data type of calculation result |
| **Usefunction** | Function call |

When defining and calling up a calculation function, the result type (**resulttype**) and data type (**datatype**) must also be specified.

### Example

The following example shows the definition of the getPrincipal calculation function, which returns a value with the **TEXT** data type as the result.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
  ...
  <function name="getPrincipal" resulttype="VALUE"
                                   datatype="TEXT">
    <if>
      <exists>
        <attribute name="AT_PRINCIPAL_NAME"
                                nodetype="PROCESS"/>
      </exists>
      <then>
        <max>
          <attribute name="AT_PRINCIPAL_NAME"
                                nodetype="PROCESS"/>
        </max>
      </then>
      <else>
        <max>
          <attribute name="AT_PRINCIPAL_ID"
                                nodetype="PROCESS"/>
        </max>
      </else>
    </if>
  </function>
  ...
</keyindicatorconfig>
```

Call up the calculation function

The getPrincipal calculation function previously defined is called up in the calculation rule for the **AT_EXP** attribute with **usefunction**. The result type for the calculation function must match the

processing operator. In the example the syntactically correct result type **VALUE** is combined with the **eq** operator that processes values.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
  ...
  <calcattr name="AT_EXP" type="PROCESS">
    <calculation>
      <if>
        <eq>
          <usefunction name="getPrincipal"
                   resulttype="VALUE" datatype="TEXT"/>
          <constant>
            <dataitem>
              KTD
              <datatype name="TEXT">Text</datatype>
            </dataitem>
          </constant>
        </eq>
        <then>
          ...
        </then>
        ...
      </if>
    </calculation>
  </calcattr>
  ...
</keyindicatorconfig>
```

Create calculation functions using PPM Customizing Toolkit. In the **Calculated attributes** menu for the **Measures and dimensions** module, call up the dialog box for creating, editing and deleting calculation functions using the **Configure calculation functions** button. If calculation functions are specified in the system, they are available in the **Define calculation rule** dialog box both for the definition of additional calculation functions and for the definition of attribute calculations.

### 8.1.3.7 Change the attribute type

Mathematical calculations are executed internally using the **DOUBLE** data type. The arithmetic link between any numerical data types is correctly calculated and then converted into the data type for the resulting attribute type.
The link between a time span attribute type and a cost attribute type is also executed correctly from a numerical point of view. The base unit of the result attribute type determines the result unit.

### 8.1.3.8 Summary

A new attribute type calculated using **calcattr** contains the result value in the base unit.

A specified calculation rule is only executed if the specified attribute type is given as an attribute to be calculated (**calculated=TRUE**) in the definition of a measure or dimension (**attrname** XML attribute).

If **PROCESS** is specified as the node type (**nodetype**), the specified attribute type is calculated only once and copied to the process instance.

If a calculation rule **OT_FUNC** is specified as the node type, the specified attribute type is calculated for every function in the process instance. It is also copied to every function.

Within a calculation rule (**calculation**), reference can be made to any existing attribute types. If this calculation is intended to access an attribute type for the function for which this calculation is currently being executed, **this** is used as the object name.

## 8.1.3.9 Example attribute calculations

**Example 1: Delivery performance**

The delivery performance measure compares the actual delivery date (end time of the **SAP.WAUS** function in a process instance) with a default value imported from the source system. If the actual delivery date is before the default value, the measure value is **0**. The value **0** is interpreted as on-time delivery. Otherwise, the measure shows the deviation from the standard value. The default value is stored in the **AT_CUSTDATE_WISH** process instance attribute. Where the **SAP.WAUS** function occurs several times in the process instance, the earliest value is determined.

```
...
<!-Delivery performance -->
<calcattr name="AT_KI_WLFTREU" type="PROCESS">
  <calculation>
    <max>
      <set>
        <constant>
          <dataitem value="0 SECOND">
            <datatype name="TIMESPAN"/>
          </dataitem>
        </constant>
        <timespan>
          <max>
            <attribute name="AT_CUSTDATE_WISH"
                                 nodetype="PROCESS"/>
          </max>
          <min>
            <attribute name="AT_END_TIME"
               nodetype="OT_FUNC" objectname="SAP.WAUS"/>
          </min>
        </timespan>
      </set>
    </max>
  </calculation>
</calcattr>
...
```

The measure is given the maximum (**max**) of a set of values (**set**) as its value. The set of values contains the element **0** (**constant**) and the time difference between the actual delivery date and the target delivery date (**timespan**). As the **attribute** XML element creates a set of values,

appropriate operators must first of all be used to determine an attribute value for further calculation. When determining the attribute value for the **SAP.WAUS** function, using the **min** operator also determines the earliest actual delivery date. The set of values created using **set** is given 2 elements: {**0**, (**Desired date - Delivery date**)}. When determining the maximum of the set of values, a negative time span results in **0** being returned while a positive time span returns the difference between the end time of the **SAP.AUS** function and the **AT_CUSTDATE_WISH** process attribute in seconds (base unit for the **timespan** data type).

As this new attribute is a process instance attribute, it is calculated only once for each process instance. The following results of the calculation can occur:

The new process instance attribute is given the calculated positive time span in the unit **Seconds**. If the calculated time span is negative, the new process instance attribute is given the time span **0 seconds**.

The new process instance attribute is not written at the process instance if the calculation fails for one or more of the following reasons and no default value is specified:

- The **AT_CUSTDATE_WISH** attribute does not exist at the process instance.
- There is no **SAP.WAUS** function at the process instance.
- The **AT_END_TIME** attribute does not exist at the **SAP.WAUS** function.

### Example 2

At each function of a process instance, the **AT_KI_COMPETENCE** attribute should specify whether the values of the **AT_COMPETENCE** and **AT_CREDIT_AMOUNT** attributes for a function match. If they match, the attribute should have the value **1**, otherwise the value should be **0**.

```
<calcattr name="AT_KI_COMPETENCE" type="OT_FUNC">
  <calculation>
    <if>
      <eq>
        <min>
          <attribute name="AT_COMPETENCE"
                  nodetype="OT_FUNC" objectname="this"/>
        </min>
        <max>
          <attribute name="AT_CREDIT_AMOUNT"
                  nodetype="OT_FUNC" objectname="this"/>
        </max>
      </eq>
      <then>
        <constant>
          <dataitem value="1">
            <datatype name="DOUBLE"/>
          </dataitem>
        </constant>
      </then>
      <else>
        <constant>
          <dataitem value="0">
            <datatype name="DOUBLE"/>
          </dataitem>
        </constant>
      </else>
```

```
    </if>
  </calculation>
</calcattr>
```

Specifying the **OT_FUNC** node type and the lack of an object name leads to the calculated **AT_KI_COMPETENCE** attribute being written to all functions in the process instance. In the attribute calculation, specifying **this** as the object name results in every function accessing its own attributes. In this case, the embracing operators **min** and **max** return the value of the referenced attribute, as the object name **this** results in an attribute set containing only one element.

### Example 3

By default, the earliest start time of a function in the process instance is used as the start time for a process instance:

```
<calcattr name="AT_START_TIME" type="PROCESS">
  <calculation>
    <min>
      <attribute name="AT_START_TIME" nodetype="OT_FUNC"/>
    </min>
  </calculation>
</calcattr>
```

If only the start times of particular functions are to be used, these functions must be checked for a particular criterion. In the following example, the auxiliary **AT_TEMP_TIME** attribute is used to filter the **"Rush order type"** criterion (**AT_AUFTRAGSART** function attribute value). The actual start time for the process instance is then determined from the filtered start times of the functions.

```
<calcattr name="AT_TEMP_TIME" type="OT_FUNC">
  <calculation>
    <if>
      <eq>
        <filteredattribute name="AT_AUFTRAGSART" nodetype=
             "OT_FUNC" objectname="this" filter="EARLY"/>
        <constant>
          <dataitem>
            Rush order
            <datatype name="TEXT"/>
          </dataitem>
        </constant>
      </eq>
      <then>
        <filteredattribute name="AT_START_TIME" nodetype=
             "OT_FUNC" objectname="this" filter="EARLY"/>
      </then>
    </if>
  </calculation>
</calcattr>

<calcattr name="AT_START_TIME" type="PROCESS">
  <depends attrname="AT_TEMP_TIME" nodetype="OT_FUNC">
  <calculation>
    <min>
      <attribute name="AT_TEMP_TIME" nodetype="OT_FUNC"/>
```

```
    </min>
  </calculation>
</calcattr>
```

### Example 4

The order group is to be saved as a function attribute in the **AT_KI_AUFTR_GRUPPE** attribute. The order group is given by the first two characters in the order number (**AT_AUFTNR**). For example, the order group **40** belongs to the order number **40268755**.

The **subtext** operator extracts the string **40** from the string **40268755** for the **AT_AUFTNR** function attribute:

```
<calcattr name="AT_KI_AUFTR_GRUPPE" type="OT_FUNC">
  <calculation>
    <subtext beginindex="0" endindex="2">
      <filteredattribute name="AT_AUFTNR" nodetype=
            "OT_FUNC" objectname="this" filter="EARLY"/>
    </subtext>
  </calculation>
</calcattr>
```

| XML attribute | Description |
|---|---|
| **beginindex** | Start index (inclusive, starting at **0**) |
| **endindex** (optional) | end index (exclusive) |

If no end index is specified, the result string begins at the specified start index and ends at the end of the source string.

**Warning**

The **subtext** operator can only be used on attributes and constants of the **TEXT** data type. If you use it on a string that contains fewer characters than the number specified in **beginindex** or **endindex**, the operator returns the value **NULL**.

### Example 5

The date **07.04.2003** is extracted from the time stamp **07.04.2003 17:30:58** and is written to all functions in the process instance as the value of the **AT_CALEN_DAY** attribute.

```
<calcattr name="AT_CALEN_DAY" type="OT_FUNC">
  <calculation>
    <createday>
      <constant>
        <dataitem value="07.04.2003 17:30:58">
          <datatype name="TIME"/>
        </dataitem>
      </constant>
    </createday>
  </calculation>
</calcattr>
```

**Example 6**

A time span of one hour (**3600** seconds in the base unit) is added to the time stamp **22.01.2002 14:55:21** and copied to all functions in the process instance as the time stamp value **22.01.2002 15:55:21** for the **AT_ADD_TSP** attribute.

```
<calcattr name="AT_ADD_TSP" type="OT_FUNC">
  <calculation>
    <addtimespan>
      <!-- Time stamp -->
      <constant>
        <dataitem value="22.01.2002 14:55:21">
          <datatype name="TIME"/>
        </dataitem>
      </constant>
      <!— Time span 3600 seconds -->
      <constant>
        <dataitem value="3600">
          <datatype name="TIMESPAN"/>
        </dataitem>
      </constant>
    </addtimespan>
  </calculation>
</calcattr>
```

## 8.1.3.10 Special features of attribute calculation

### 8.1.3.10.1 AT_INTERNAL_NO_CUBE_ENTRY function attribute

For certain functions, you can specify that they are not to be saved in the function cube. If the attribute **AT_INTERNAL_NO_CUBE_ENTRY** exists at a function and has the value **true** this function instance will not be written to the cube table. The existence of this attribute does not impact the measure calculation of this function, that is, you can create the attribute at the function instance using a calculation rule, as well.

The attribute is evaluated by instance, that is, if the attribute is missing at individual function instances (or if the attribute value is not equal to **true**), these function instances will be saved in the function cube. However, if you overwrite these function instances having the attribute **AT_INTERNAL_NO_CUBE_ENTRY** and the value **true** when reimporting, the entries in the function cube will be deleted, as well.

#### ATTRIBUTE DEFINITION

The attribute **AT_INTERNAL_NO_CUBE_ENTRY** is not included in the default configuration of PPM attributes. If you want to use this feature, you first need to define the attribute **AT_INTERNAL_NO_CUBE_ENTRY** with the **boolean** data type.

The functions displayed in the analysis process tree are based on entries in the function cube. Functions whose instances were not written to the function cube due to the attribute value **true** for the attribute **AT_INTERNAL_NO_CUBE_ENTRY** are not displayed in the process tree.

The feature described does not affect the use of process hierarchies because you can assign process instances to functions that were not calculated. The functions not saved in the function cube are not displayed in the process tree of the assigned process type, either.

### 8.1.4 Typification rules in CTK

You can define typification rules in the **Processes** CTK module. To create, edit or delete a rule for a particular process type, simply select the corresponding process type from the process tree and select the relevant item from the pop-up menu. It is also possible to create a typification rule based on a template. All rules previously defined can be used as a template. The definition of the calculation rule for a typification rule is specified using the familiar operands and operators from the attribute calculation (see **Definition of attribute calculations** (page 45)).

**Warning**

When defining the corresponding calculation rule for a typification rule, you need to ensure that it delivers a return value of the **BOOLEAN** type.
Each calculation rule is automatically checked for correct syntax in the **Configure typification rule "typifierrule_<processtypegroup>_<processtype>"** dialog.

As soon as you save your changes, they are permanently stored in the process tree and measure configurations. When you activate the changed configuration, it is transferred to the PPM system.

## 8.2 Typification by attribute calculation

The typification can be done by using typification rules, or alternatively, by importing values in specific attributes, the so-called "pretypification".

The attributes can always be calculated separately from typification and process assignment. You can do that by assigning the attributes to the process tree root. This attribute calculation done by **runppmimport** is processed between merge and typification/measure calculation.

In this way, an EPC typification can be applied by importing or calculating the attributes **AT_PROCTYPEGROUP** and **AT_PROCTYPE**. If these are set typification will use their values instead of using the typification rules as described above.

Attributes **AT_PROCTYPE** and **AT_PROCTYPEGROUP**:

| Attribute | Description | Usuage |
|---|---|---|
| AT_PROCTYPE | process type | Imported or set by typification rules |
| AT_PROCTYPEGROUP | process type group | Imported or set by typification rules |

# 9 Definition of measures, dimensions, attribute calculations, and relations

Measures in the PPM system supply measurable values of process or function instance properties that can be calculated, such as function cycle time in hours or order volume in euros.

Dimensions further specify the calculated measure values of process and function instances using particular criteria, such as order number, sold-to party, etc.

The following chapters describe how you define measures and dimensions or attribute calculations and relations and make them available to the PPM system through special configurations of the process tree (see chapter **Register measures and dimensions at the PPM system** (page 194)).

## 9.1 Terminology

Key terms of the chapter on **Definition of measures, dimensions, attribute calculations, and relations** are explained in detail below.

### 9.1.1 Measures

The PPM system differentiates between various measure categories:

#### DIFFERENTIATION BY MEASURE TYPE

The following measure types are differentiated by the object type that the measure refers to:

- **Process measures** are measures whose values are available for analysis at the entire process instance.
- **Function measures** are evaluated based on function instances.
- **Relation measures** are measures that are available for the evaluation of relations.
- **Cardinality measures** are available for specific text dimension evaluations.

#### DIFFERENTIATION BY PROCESS REFERENCE

- **Process instance-dependent measures** are measures whose values are calculated with a reference to process instances.
- **Process instance-independent measures** are measures whose values are calculated without a reference to process instances.

#### DIFFERENTIATION BY DEFINITION

- **Standard measures** are defined in the client-specific measure configuration file. A major part of these measures is preconfigured in PPM.
- **User-defined measures** are defined by users in a particular module of the PPM user interface based on standard measures and then saved in a special XML configuration file. Preconfigured user-defined measures are also part of the ARIS Process Performance Manager scope of supply.

The listed measure categories can be combined, for example, you can define process instance-independent process measures.

All measure categories have in common that the concrete value of a measure describes a particular, measurable property of a process instance, for example, like time of execution or number of processors.

Furthermore, measures can be grouped logically. The assignment of a measure to a group must be unique. This means that each measure can only be assigned to one group. The group structure is hierarchical and can be of any depth.

#### 9.1.1.1 Process instance-dependent measures

##### STANDARD MEASURES

The values of standard measures are calculated based on attributes at the process, function, or relation instance level using the attribute calculator component of the Measure calculator.
The calculation uses either a fixed algorithm programmed in the PPM software or an algorithm specified by the user in the XML Measure configuration file.

##### USER-DEFINED MEASURES

The algorithm for calculating user-defined measures can be conveniently created using the PPM front-end.

The fundamental difference from standard measures is the fact that the calculation is not instance-specific and based on attributes, but on the sets of values for already calculated measures. The results are not saved in the PPM database and are recalculated each time the measure is called up. Changes to the algorithm are displayed immediately by calling up the measure again in the PPM user interface.

#### 9.1.1.2 Process instance-independent measures (PIKIs)

The values of process instance-independent measures are calculated based on data that is not process-oriented. Process instance-independent measures can be analyzed in the PPM system just like process instance-dependent process measures and be used in calculation rules of user-defined measures, for example.

Process instance-independent measures are not calculated from process instance data. The concrete measure value does not have any process instance reference.

To find out how process instance -independent measures are defined, please refer to chapter **Definition of process instance-independent measures** (page 143).

The configuration of the file import formats (XML, CSV, XLS) of process instance-independent measures is described in the technical reference **PPM Data Import**.

### 9.1.2 Dimensions

Dimensions are criteria for differentiating process instances and function instances. Dimension values are based on attribute values, which are either transferred directly from the source system (for example, location, product area) or calculated (for example, process type).

The following dimension types exist:

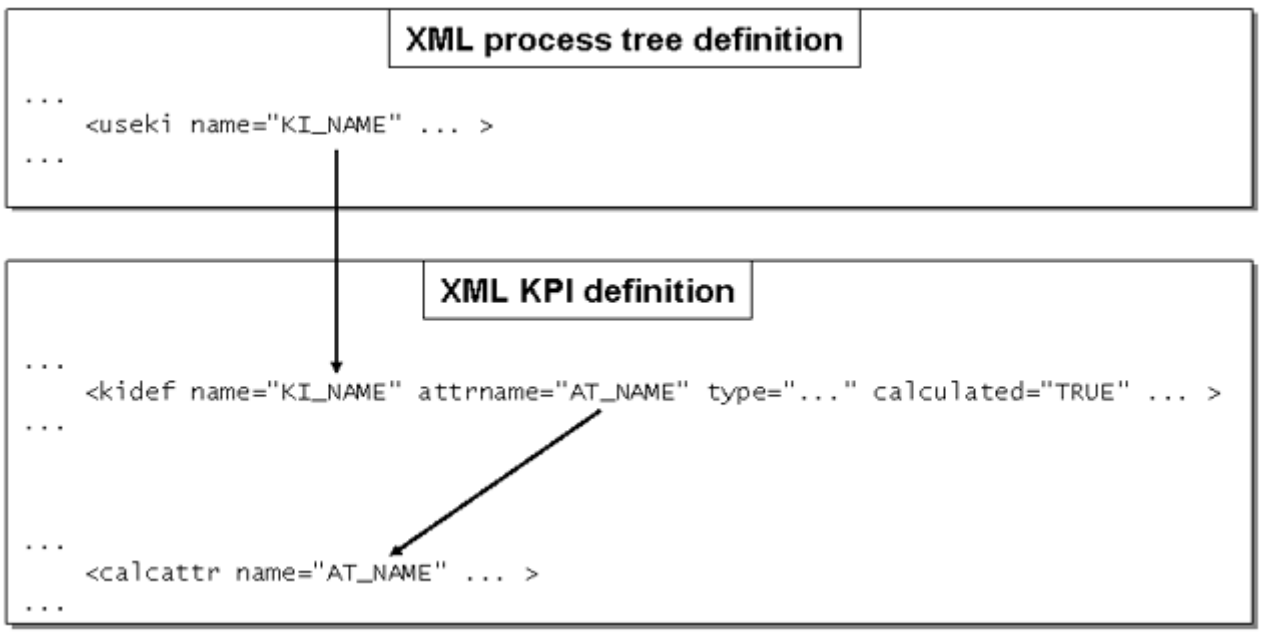
- Text dimensions (page 162) (one-level, two-level, n-level)
- Floating point dimensions (page 173) (floating point number format)
- Time dimension (page 174)
- Time of day dimension (page 181)
- Time range dimension (page 179)
- Search dimensions (page 182)
- Shared function dimension (page 186)

The standard step size for displaying a dimension is either explicitly specified in the configuration file or is calculated automatically by the system for optimum representation.

## 9.2 Definition of measures

The starting point for the calculation of measures and dimensions is the process tree. When calculating, for all the measures specified in the process tree configuration file (**useki** XML element), the definition in the measure configuration (**kidef** XML element) is retrieved and the associated calculation rule (**calcattr** XML element) is executed. This procedure only calculates measures that are used in the process tree, and optimizes the performance of the Measure calculator independently of the number of defined measures. Dimensions are calculated in the same way (**usedim** XML tag).

The graphic below illustrates the dependencies between the measure definition and the process tree definition:



The measure configurations supplied with the PPM system (***_keyindicator.xml** files in the directories **<PPM installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ctk\examples\custom\<client template>\xml** contain definitions of the most common, generally applicable measures and dimensions. This default configuration can easily be expanded in the PPM Customizing Toolkit module **Measures and dimensions** to include project-specific measures.

**Warning**

Avoid using the suffixes **_NUM** and **_SUM** when assigning internal measure names. These suffixes are used internally by the Measure calculator.

## 9.2.1 Definition of standard measures

A measure is defined in the client-specific XML configuration file with the **KeyindicatorConfiguration.dtd** document type definition by the following element:

```
...
<kidef name="..." attrname="..." type="..."
       calculated="..." distribution="..."
       standarddeviation="..." retrievertype="..."
       kigroup="..." sharedfunctionki="..."
       functionspanki="..." colname="..."
                        importmode="OPTIONAL">
  <description language="..." name="...">
   Description text...
  </description>
</kidef>
...
```

| XML attribute | Description |
|---|---|
| **name** | Internal name of the measure. Referenced in the **useki** XML tag in the process tree definition. |
| **type** | Measure type<br>**PROCESS**: Process measure<br>**FUNCTION** (obsolete): Function measure<br>**OT_FUNC**: Function measure<br>**OT_ORG**: Organizational measure<br>**RELATION**: Relation measure |

| XML attribute | Description |
|---|---|
| **location** (optional) | Only for **type="RELATION"**<br>Valid values: **SOURCE** (attribute placement on source reference object of relation)<br>**TARGET** (attribute placement on target reference object of relation)<br>**THIS** (default value: attribute is placed at the relation itself) |
| **description** | Language-specific description of a group, optionally with tooltip (**#PCDATA** section in the **description** element). The description must be specified in at least the default language. |
| **attrname** | Name of the attribute on which the measure is based. This can be an existing attribute value (**calculated=FALSE**) or an attribute value to be calculated (**calculated=TRUE**). |
| **calculated** | **TRUE**: The value of the referenced attribute is calculated using the calculation rule specified by **calcattr**.<br>**FALSE**: The value of the referenced attribute is not calculated. |
| **distribution** | **TRUE**: The measure can be used as a dimension.<br>**FALSE**: The measure cannot be used as a dimension. |
| **standarddeviation** (optional) | **TRUE**: The standard deviation can be calculated for the measure. The standard deviation can be calculated for all measures except **Number of processes** and **Number of functions**.<br>The default value is **TRUE**. |
| **sharedfunctionki** (optional) | **TRUE**: The measure is treated as a shared function measure for calculating measures. The measure for a shared function is only calculated once and applies to all instances of the shared function.<br>Default value is **FALSE**. |

| XML attribute | Description |
|---|---|
| **functionspanki** (optional) | **TRUE**: The measure is a function span measure (for example, cycle span).<br>If the function occurs multiple times within a process instance, the measure value calculated applies only once per instance.<br>Default value is **FALSE**. |
| **retrievertype** (optional) | Type of measure retriever used. Defines how the set of measure values for the process instances involved in a particular analysis is aggregated.<br>Default value: **KEYINDICATOR**.<br>**KEYINDICATOR**:<br>Calculates the average value (for example, cycle time). Numerical types except **LONG** are all permitted as data types.<br>**NUM_KEYINDICATOR**:<br>Aggregates numerical measures (for example, number of processes, number of functions) by adding the values. All numerical data types are valid.<br>**FREQ_KEYINDICATOR**:<br>Aggregates frequencies (for example, process frequency, function frequency). The values are added and then divided by the time span resulting from the selected step width of the dimension and the set time filter.<br>**FACTORY_KEYINDICATOR**:<br>Aggregates measures by calculating the average using the factory calendar.<br>**FACTORY_TIMESPAN** is the only permissible data type. |
| dimreferring | Type of dimension reference<br>**LOOSE**: Loose<br>**STRICT**: Strict<br>Default value: **LOOSE** |
| kigroup (optional) | Measure group |

| XML attribute | Description |
|---|---|
| **importmode** (optional) | Output of error messages when calculating measure values.<br>**OPTIONAL**: Calculation errors are not output.<br>**MANDATORY**: Calculation errors are output<br>Default value: OPTIONAL |

Only one of the **sharedfunctionki** and **functionspanki** attributes may have the value **TRUE**. If one of the two attributes has the value **TRUE**, the **type** measure type must have the value **FUNCTION** (function measure).

The **FACTORY_KEYINDICATOR** measure retriever type is no longer used from PPM 3.x, but is still supported for compatibility reasons. When the configuration is imported, it is replaced by the **KEYINDICATOR** retriever type.

### 9.2.1.1 Formatting measure values

Measure values are rounded to three decimal places by default and are displayed with a thousands separator or in accordance with the specifications pertaining to the **MINIMUM_FRACTION_DIGITS** and **MAXIMUM_FRACTION_DIGITS** keys in the file **Keyindicator_settings.properties**. Alternatively, you can use the **format** XML element to specify different formats for each individual measure, provided that it is not a goal accomplishment indicator.

Goal accomplishment indicator values are always rounded to one decimal place and one significant place.

**Example**

```
...
<kidef name="PDLZ" attrname="..." type="..."
       calculated="..." distribution="..."
       standarddeviation="..." retrievertype="...">
  <description name="Process cycle time" language="de"/>
  <format fractiondigits="1" significantdigits="1" />
</kidef>
...
```

In the analysis in PPM, the values of the **Process cycle time** measure are rounded to one decimal place (**fractiondigits="1"**) when displayed. One significant figure (**significantdigits="1"**) is to be displayed for the relevant measure value in tooltips and model attributes for the EPC view.

The definition of the format specifications is located in the file **_formatinfo.dtd**.

| XML tag | Description |
|---|---|
| **format** | Format specifications for measure values |

| XML tag | Description |
|---|---|
| **fractiondigits** (optional) | Number of decimal places to be displayed for measure values in tables, on EPC object connections, and in filter dialogs.<br>Default value: **3** |
| **significantdigits** (optional) | Only applies to measure values in tooltips and model attributes for the EPC view:<br>Number of significant figures to be displayed (before and after the decimal point and not equal to 0) up to a maximum of ten decimal places in total. For example, if you specify **significantdigits="6"** the value **1453.03500125** will be displayed as **1453.035** regardless of the specification for **fractiondigits**. |
| **usegrouping** (optional) | **TRUE**: Thousands separators are displayed.<br>**FALSE**: Thousands separators are not displayed.<br>Default value: **TRUE** |

### 9.2.1.2 Definition of process cost measures

The process costs of a process instance are given by the total process costs of all function instances within the process instance. The process costs of function instances are calculated using the cost rates for the organizational units assigned to the functions (see **Anonymizing** (page 40) chapter) and the execution times of the functions. The number of executions of a function by an organizational unit is given by the **AT_COUNT_PROCESSINGS** attribute for the connection between the organizational unit and the function. If several organizational units are assigned to a function, this is assessed as repeated execution of the function.

To calculate cost measures, the **Costs** and **Cost rate** data types must be known. The definition of these data types is included in the XML configuration file ***_datatypes.xml** of the corresponding CTK client template (under <PPM installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ctk\examples\custom\), which you can adjust to meet your project requirements.

The execution times of functions required to calculate cost measures can be calculated in two different ways. Depending on the selected calculation method, the calculated costs will be saved as different measures. The calculation method used for the execution times depends on which information is extracted from the source system.

### MEASURES FPKS_R AND PK_R

To calculate the cost rate based on the processing time, you use the processing time (**AT_KI_FBZ** function attribute) calculated from the **AT_START_TIME** and **AT_END_TIME** attributes for a function. The calculated cost rate is saved in the **AT_PKS_R** function attribute.

### MEASURES FPKS_S AND PK_S

To calculate the cost rate based on the performance standard, an estimated standard processing time is extracted from the source system and written to the functions as the **AT_LS** attribute. The calculated cost rate is saved in the **AT_PKS_S** function attribute.

The process cost rate for a function specifies the average costs for processing the function once and is calculated using the following calculation rule for the two calculation methods described: The product of the execution time of a function and the sum of the weighted cost rates of all organizational units assigned to the function is divided by the total number of executions.

The following formula illustrates the calculation rule:

**FPKS** Process cost rate (function)
**FT** Function execution time
**KS** Process cost rate
**FREQ** Processing frequency

$$\mathbf{FPKS} = \frac{\mathbf{FT} * \mathrm{sum}(\mathbf{KS} * \mathbf{FREQ})}{\mathrm{sum}(\mathbf{FREQ})}$$

The method of calculating cost measures is selected by registering the corresponding measures in the process tree. The default configuration of PPM calculates process cost rates based on the performance standard.

Extract from the file ***_processtree.xml**:

```
...
<useki name="FPKS_S" scale="EUR" assessment="NEG"/>
<useki name="PK_S" scale="EUR" assessment="NEG"/>
...
```

## 9.2.2 Measure definition in multi-byte character sets

The following extract from the measure configuration file shows an example of the definition options for user-defined measures when using a multi-byte character set:

**Example with tooltip and attribute calculation:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
...
<calcattr name="IΔ_OΓK_ENT" type="PROCESS">
  <calculation>
    ...
  </calculation>
```

```
</calcattr>
...
  <!-- Ορισμός του δείκτη όγκου των εντολών -->
  <kidef name="ORDERVOL" attrname="ΙΔ_ΟΓΚ_ΕΝΤ"
          type="PROCESS" calculated="FALSE"
          distribution="FALSE" standarddeviation="FALSE"
          retrievertype="NUM_KEYINDICATOR"
          kigroup="KI_GROUP_COST" dimreferring="LOOSE"
          importmode="OPTIONAL" sharedfunctionki="FALSE"
          functionspanki="FALSE">
    <description name="Auftragsvolumen" language="de"/>
      Order volume
    <description name="Order volume" language="en">
      Order volume
    <description name="Όγκος εντολών" language="el">
      Όγκος εντολών κατα αύξοντα αριθμό
    </description>
  </kidef>
  ...
</keyindicatorconfig>
```

## 9.2.3 Definition of cardinality measures

The value of a cardinality measure is based on the number of different values (= max. possible steps) of the referenced text dimension for the specified step width (level). A cardinality measure can be defined for one-level, two-level, and n-level dimensions, and is defined by the following XML element in the measure configuration file:

```
...
<crdkidef name="..." dimreferring="...">
  <description language="de" name="..."/>
  <description language="en" name="..."/>
   <refdim name="..." refinement="..."/>
</crdkidef>
...
```

| XML tag | Description |
|---|---|
| **name** | Internal name of the measure. Referenced in the **useki** XML tag in the process tree definition. |
| **dimreferring** | Type of dimension reference.<br>**LOOSE**: Loose<br>**STRICT**: Strict<br>Default value: **LOOSE** |

| XML tag | Description |
|---|---|
| **refdim** | The **name** XML attribute specifies the name of the dimension to which the calculated cardinality relates. You also have the option of specifying the step width of the dimension for which the cardinality is calculated in the **refinement** XML attribute. If nothing is specified here, the default step width of the dimension is used. It is mandatory to specify the step width for n-level dimensions. Valid values:<br>One-level dimension:<br>**BY_LEVEL_1**<br>Two-level dimension:<br>**BY_LEVEL_1** (rough) or **BY_LEVEL_2** (detailed)<br>N-level dimension:<br>Only<br>**BY_LEVEL<1_N>** (roughest level, for example, **BY_LEVEL1_12**) or<br>**BY_LEVEL<N_N>** (most detailed level, for example, **BY_LEVEL12_12**)<br>The default value is the default step width for the referenced dimension. |
| **kigroup** (optional) | Measure group<br>Default value: **All measures** group |

As well as the measure itself, only the ranking, previous periods, and planned values can be determined for cardinality measures. Statistical evaluations (minimum, maximum, total and standard deviation) cannot be displayed. Cardinality measures cannot be used as a dimension. No filters can be specified for cardinality measures.

Any additional dimension values resulting from import of process instance-independent measures will not be included in the calculation of cardinality measures. The cardinality of dimensions that are used exclusively by process instance-independent measures always return the value **0**.

## 9.2.4 Definition of process instance-independent measures

Process instance-independent measures are defined in the client-specific measure configuration (XML file with the document type definition **keyindicatorconfiguration.dtd**) in the general context of data series.

## DEFINITION OF DATA SERIES

A data series (**pikicube** XML element) consists of process instance-independent measures and referenced dimensions. It must contain at least one process instance-independent measure (**pikidef**) and at least one referenced dimension (**refdim**). Referenced dimensions must be dimensions configured in the PPM system.

Process instance-independent measures in data series are always of the **Process** type in order to ensure maximum usability in the PPM system. Therefore, the type of the data series itself is not important, see chapter **Usage (type) of a data series** (page 149).

For each data series, at least one referenced dimension must be marked as a key dimension (**refdim ... iskeydimension="TRUE"**). By default, all referenced dimensions are key dimensions. A particular value combination of the specified key dimension(s) supplies a unique data row within a data series, that is, a particular value combination exists only once within a data series.

### Example

The following data series contains three data rows that differ by the value combinations of the specified key dimensions (*****):

| D_COUNTRY * | D_PLANT* | D_DEPARTMENT * | D_RECORDED BY | SALES | COSTS |
|---|---|---|---|---|---|
| Germany | Hamburg | 42 | Smith | 400000 | |
| Germany | Frankfurt | 17 | Hartmann | 510000 | 360000 |
| USA | Pittsburgh | 53 | Fox | | 410000 |

Each of the three data rows can occur only once within the data series. The specific value combination of the key dimensions **D_COUNTRY**, **D_PLANT**, and **D_DEPARTMENT** (for example, **Germany; Hamburg; 42**) represents the identifier of a data row.

### CONFIGURATION

The following general file structure illustrates the configuration of a data series (**pikicube** XML element):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
  ...
  <pikicube name="...">
    <description language="de" name="..."/>
    ...
    <pikidef name="..." retrievertype="..." ↵
                      dimreferring="...">
      <description language="de" name="...">
       Descriptive text (tooltip)
      </description>
      ...
      <datatype name="..."/>
    </pikidef>
```

```
    <refdim name="..." refinement="..." ↵
                  iskeydimension="TRUE"/>
    ...
  </pikicube>
  ...
</keyindicatorconfig>
```

The following tables explain the configuration of a process instance-independent data series:

| **ELEMENT and ATTLIST pikicube** | **Description** |
|---|---|
| pikicube | Process instance-independent data series |
| name | Data series name unique in the system. It is also used as the name of the cube in the database. |
| comment (optional) | Comment on the data series; used in PPM Customizing Toolkit. |
| editable | **editable="TRUE"** (default value) enables data input for the data series in the **Configuration / Data input** module of the PPM interface. |
| type | Usage (type of data series) that determines which dimensions may be used in the data series as referenced dimensions. The default value is **PROCESS**, that is, only process dimensions (**dimtype="PROCESS"** in the dimension definition) may be specified in the data series.<br>Other valid values:<br>**OT_FUNC** (only function dimensions allowed in the data series)<br>**RELATION** (only dimension of the **RELATION** type allowed in the data series)<br>For more information, please refer to chapter **Usage (type) of a data series** (page 149). |
| relname | Only for **type="RELATION"**. A single relation existing in the PPM system is to be specified with its name, for example, **relname="REL_WORKS_TOGETHER"**. The data series is assigned to the specified relation. |
| deletedata ↵ onredefinition | Obsolete, no longer used. |
| description | Language-specific description of the data series. The description must be specified in at least the default language. |

| ELEMENT and ATTLIST pikicube | Description |
|---|---|
| pikidef | Definition of a process instance-independent measure, at least one for each data series, see below. |
| refki | Obsolete, no longer used. |
| refdim | Referenced dimension, see below |

| ELEMENT and ATTLIST refdim | Description |
|---|---|
| refdim | A dimension existing in the PPM system, to which the process instance-independent measures of the data series refer. You must specify at least one referenced dimension for each data series.<br>For process instance-independent measures internal dimensions (page 167) are not supported as referenced dimensions (refdim). |
| name | Internal name of the dimension existing in the PPM system. |
| refinement | Dimension step width that data import is to be performed with.<br>The dimension values to be imported must be specified in this step width exactly. |
| iskeydimension | **iskeydimension="TRUE"** (default value) specifies that the referenced dimension is a key dimension of the data series. The value combinations of all specified key dimensions render each data row of a data series unique. |

| ELEMENT and ATTLIST pikidef | Description |
|---|---|
| pikidef | Definition of a process instance-independent measure. You must specify at least one definition for each data series.<br>A process instance-independent measure can be used in a single data series. |
| name | Name of the process instance-independent measure that is unique in the system. |

| ELEMENT and ATTLIST pikidef | Description |
|---|---|
| type | Obsolete, no longer used. |
| retrievertype | Measure retriever type. Default value:<br>**KEYINDICATOR** (averaging)<br>Further values:<br>**NUM_KEYINDICATOR** (summation)<br>**FREQ_KEYINDICATOR** (obsolete, is no longer used)<br>**FACTORY_KEYINDICATOR** (is no longer used) |
| dimreferring | Type of dimension reference<br>**LOOSE**: Loose<br>**STRICT**: Strict<br>Default value: **LOOSE** |
| kigroup (optional) | Assignment of the process instance-independent measure to an existing measure group |
| description | Language-specific description of a process instance-independent measure. The description must be specified in at least the default language. |
| datatype | Data type of the values of a process instance-independent measure |

You can define any number of data series (**pikicube** XML elements) within a measure configuration. In a data series, you can specify any number of process instance-independent measure definitions (**pikidef** XML elements). The same dimension reference applies to all process instance-independent measure definitions of a data series (**refdim** XML elements).

You can conveniently configure process instance-independent data series in the sub-module **Process instance-independent measures** in the CTK module **Measures and dimensions**.

You can export all process instance-independent data series configured in a PPM system via the XML interface using the **runppmconfig** command line program with the parameter **-keyindicator** to an XML file.

**Example 1**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
  ...
  <pikicube name="PIKICUBE_TURNOVER_PROD_GROUP">
    <pikidef name="TURNOVER_PROD_GROUP" ↵
            retrievertype="KEYINDICATOR" ↵
            dimreferring="STRICT" ↵
            kigroup="KI_GROUP_COST">
      <description language="de"
```

```
                name="Umsatz pro Produktgruppe"/>
      <description language="en"
                name="Turnover by product group"/>
      <datatype name="DOUBLE"/>
    </pikidef>
    <refdim name="TIME" refinement="BY_MONTH"/>
    <refdim name="D_PRODUCT_GROUP"/>
    <refdim name="PROCESSTYPE" refinement="BY_LEVEL2"/>
  </pikicube>
  ...
</keyindicatorconfig>
```

A data series with the internal name **PIKICUBE_TURNOVER_PROD_GROUP** is created.

The definition (**pikidef**) of the process instance-independent measure **TURNOVER_PROD_GROUP** specifies a strict dimension reference (**dimreferring="STRICT"**) and an assignment of the process instance-independent measure to the **KI_GROUP_COST** measure group.

The process instance-independent measure (**refdim="..."**) strictly refers to the **TIME**, **D_PRODUCT_GROUP**, and **PROCESSTYPE** dimensions. The reference to the **PROCESSTYPE** dimension is defined with the **detailed** (**refinement="BY_LEVEL2"**) step width.
Since the **iskeydimension** attribute is not specified in the **refdim** elements, the default attribute value **TRUE** is used, that is, all referenced dimensions are used as key dimensions of the data series.

**Example 2**

```
...
  <pikicube name="PIKICUBE_COSTS">
    <description language="en" name="Costs"/>
    <pikidef name="OVERHEAD_COSTS" ↵
            retrievertype="KEYINDICATOR" ↵
            dimreferring="LOOSE">
      <description language="en" name="Overhead costs"/>
      <datatype name="COST"/>
    </pikidef>
    <refdim name="PROCESSTYPE" refinement="BY_LEVEL2" ↵
                          iskeydimension="FALSE"/>
    <refdim name="TIME" refinement="BY_MONTH" ↵
                    iskeydimension="TRUE"/>
    <refdim name="MATERIAL" refinement="BY_LEVEL2" ↵
                        iskeydimension="TRUE"/>
  </pikicube>
...
```

This file extract defines the data series **Costs** with the process instance-independent measure **Overhead costs** of the data type **COST** with the internal name **OVERHEAD_COSTS** which is unique in the PPM system.

The measure value retriever type is averaging (**KEYINDICATOR**) and the dimension reference is loose (**LOOSE**).
In the **refdim** XML elements, the PPM dimensions **TIME** and **MATERIAL** are specified as key dimensions (**iskeydimension="TRUE"**) of the dimension reference for the process instance-independent data series.
Additionally, step widths that differ from the default step widths are specified for the dimension values to be imported.

### REGISTRATION OF PROCESS INSTANCE-INDEPENDENT MEASURES AT THE PPM SYSTEM

Process instance-independent measures are registered in the process tree (**useki** element in the XML file with the document type definition **keyindicatorprocesstree.dtd**) at process type groups and process types.

Further information on registering process instance-independent measures at the process tree is available in chapter **Register measures and dimensions of process instance-independent data series** (page 196).

## 9.2.4.1 Usage (type) of a data series

You need to select one of the following usages (**pikicube type="..."**) for a data series, which specifies the dimensions that are allowed to be used in the PIKI cube:

- Process (**PROCESS** default value)
- Function (**OT_FUNC**)
- Relation (**RELATION**)

Regardless of the selected type of data series, process instance-independent measures are always of the **PROCESS** type, that is, they are handled like process measures.

The effects of the PIKI cube types are as follows.

### PROCESS

Only process dimensions (**dimtype="PROCESS"** in the definition of the dimension) are allowed as referenced dimensions (**refdim="..."**) in the data series.

### OT_FUNC

Only process and function dimensions (**dimtype="PROCESS"** or **"OT_FUNC"** or **"FUNCTION"**) are allowed as referenced dimensions in the data series.

### RELATION (WITH RELATION NAME <X>)

Only process dimensions, relation dimensions of the relation <x>, and source and target dimensions of the relation <x> (that is, **FPROCESSTYPE**, **FROMORG**, **TOORG**, **FUNCTION**, **ORGUNIT**) are allowed as referenced dimensions of the data series.

## 9.2.4.2 Dimension reference

Process instance-independent measures can have a loose or strict dimension reference (**dimreferring** XML attribute). The default value is loose dimension reference (**dimreferring="LOOSE"**).

## LOOSE DIMENSION REFERENCE

A process instance-independent measure with a loose dimension reference can be analyzed for all available dimensions. The process instance-independent measure also delivers values for queries with step widths other than that specified for the process instance-independent measure (**refinement** XML attribute) and for dimensions for which no reference is defined.

If you are analyzing a process instance-independent measure with a dimension for which no dimension reference (**refdim** XML element) has been defined, this dimension is ignored in the value calculation for the process instance-independent measure. The process instance-independent measure values shown only apply to the dimensions referred to in the definition of the process instance-independent measure data series.

Likewise, queries with a more detailed step width return the process instance-independent measure values that refer to the defined step widths. This means that other step widths are ignored in the analysis.

**Example**

| Overhead costs | Total costs | Customer (rough, detailed) | Time (by month) |
|---|---|---|---|
| 1000 € | 25000 € | Germany, Becker | Jan 2001 |
| 3000 € | 68000 € | Germany, Schmidt | Jan 2001 |
| 1500 € | 13000 € | France, Leclerc | Jan 2001 |
| 1200 € | 12000 € | Germany, Becker | Feb 2001 |
| 3400 € | 78000 € | Germany, Schmidt | Feb 2001 |
| ... | ... | ... | ... |

The table lists the process instance-independent measures **Overhead costs** and **Total costs** with reference to the **Customer** and **Time** dimensions. If your analysis queries the overhead costs for the customer **Germany**, **Becker** for **15th Jan 2001**, you obtain the return value **1000 €** for the process instance-independent measure. However, this value actually relates to the whole month of January 2001 (**refinement="BY_MONTH"**).

Make sure to observe the defined dimension references of a process instance-independent measure as well as the specified step widths of the referenced dimensions in order to ensure plausible analysis results.

## STRICT DIMENSION REFERENCE

A process instance-independent measure with a strict dimension reference can only be evaluated with the dimensions to which it refers to in the definition of the data series (**refdim** XML elements). Queries for dimensions to which the process instance-independent measure does not refer are not possible. Queries with a step width other than that defined are not possible, either.

If a different step width or dimension is selected in the analysis, a corresponding error dialog is displayed.

### THE SPECIAL CASE OF THE "PROCESS TYPE" DIMENSION REFERENCE

If you specify the dimension **Process type** (**PROCESSTYPE**) as dimension reference in a process instance-independent data series, only process types that already exist in the PPM system can be used for data import. If you try to import process instance-independent data into a process type that does not exist the import outputs an error message including the involved data rows. The process tree is not automatically extended. Data import is not aborted but the data rows with the non-existing process type are not imported.

## 9.2.4.3 Definition of process instance-independent measures in multi-byte character sets

The following extract from the measure configuration file shows an example of the definition options for process instance-independent data series when using a multi-byte character set:

```
...
<!-- Ορισμος σειράς δεικτών -->
  <pikicube name="PIKICUBE_1">
    <description name="Umsatz" language="de"/>
    <description name="Turnover" language="en"/>
    <description name="Τζίρος" language="el"/>
    <pikidef name="PIKI_1" ↵
            retrievertype="NUM_KEYINDICATOR" ↵
            dimreferring="LOOSE"↵
            kigroup="KI_GROUP_COST">
      <description name="Umsatz" language="de"/>
      <description name="Turnover" language="en"/>
      <description name="Τζίρος" language="el"/>
      <datatype name="COST"/>
    </pikidef>
    <pikidef name="PIKI_2" ↵
            retrievertype="KEYINDICATOR" ↵
            dimreferring="LOOSE">
      <description name="Kundenzufriedenheit" ↵
                              language="de"/>
      <description name="Customer satisfaction" ↵
                              language="en"/>
      <description name="Ευχαρίστηση των πελατών" ↵
                              language="el"/>
      <datatype name="DOUBLE"/>
    </pikidef>
    <refdim name="MATERIAL"/>
  </pikicube>
...
```

## 9.2.4.4 Configuration import

Process instance-independent data series are imported together with the measure configuration by means of the command line program **runppmconfig** (see **PPM Operation Guide**):

```
runppmconfig –user <user name> –password <password>
          [–client <client name>]
           –mode import
          [–overwrite]
           –keyindicator <XML measure configuration>
```

The executing user must have the **Configuration import** function privilege.

## ADDITIVE CONFIGURATION IMPORT

By default, that is, without the option **-overwrite**, the import of the measure configuration is additive, that is, data series that already exist in the PPM system are retained and remain unchanged.

For each imported data series, a database table with the internal name of the data series (**pikicube name="..."**) is created and the corresponding data structure is established on the analysis server.

## OVERWRITING CONFIGURATION IMPORT

When importing a changed configuration of a process instance-independent data series at a later time using the command line option **runppmconfig -mode import -overwrite**, you must observe whether your changes affect the data structure of the existing data series (see below).

If they do, you must first delete the data imported into the data series before you import the changed configuration, if they do not, this is unnecessary.

### CHANGES THAT DO NOT AFFECT THE DATA STRUCTURE

By specifying the option **-overwrite** you can make the following changes to the configuration of data series existing in the PPM system without first having to delete already imported data:

- Add further key dimensions or non-key dimensions and other process instance-independent measures
- Change a referenced dimension to a key dimension (**iskeydimension="TRUE"**)
- Change the description of a data series (PIKI cube)
- Change the usage of a data series (for example, **type="PROCESS"** to **type="FUNCTION"**)
- Assign a data series to a different relation (**relname="..."**)
- Change the dimension reference (loose/strict) for non-key dimensions
- Change the measure value retriever type
- Change the option **editable**

### CHANGES THAT AFFECT THE DATA STRUCTURE

If you want to import configuration changes that affect the data structure of a data series, you may need to delete previously imported data of the data series first (via the PPM user interface or the command line program **runpikidata** with the option **-mode delete**). Only then you can import the changed configuration using the import parameter **-overwrite**.

If the data series still contains data during the import of data structure relevant configuration changes, an error message is output and the new definition of the data series is not transferred. Import of allowed configuration changes is not canceled.

**Configuration changes that affect the data structure include:**

- Deleting a referenced dimension (key dimension, non-key dimension)
- Deleting a process instance-independent measure

- Changing a key dimension to a non-key dimension (**iskeydimension="FALSE"**)
- Changing the step width of a referenced dimension or the data type of a process instance-independent measure

### 9.2.4.5 Data series migration

Please observe the following before you migrate existing process instance-independent data series from a PPM system version 4 to a PPM system version 9 using the command line program **runppmconverter.bat** in <installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-<version>-runnable.zip\ppm\bin:

- Before conversion, you need to back up import data of process instance-independent data series from a PPM database version 4 to an XML file (see **PPM Migration Guide**).
  If configuration changes affecting the structure of data series are required before the conversion of existing data series you need to delete any existing import files before the conversion (see chapter **Configuration import** (page 151)), otherwise, the conversion program aborts with an error message.

If you use the command line converter with the option **-ignorepikidata** all existing import data of existing data series are completely deleted before conversion.

After conversion of the configuration for PPM version 9 you can import the exported data again, see technical reference **PPM Data Import**.

- Data series containing referenced measures that are no longer supported (**refki** XML elements) are not converted automatically. The conversion program outputs a message that, if required, you can export existing data of the data series, adapt the configuration accordingly, and reimport the data including the adapted configuration.
- If you use the XML attribute **deletedataonredefinition**, which is no longer supported, in existing process instance-independent data series, it must be removed manually from the configuration before conversion.
- Data series in the PPM system version **9** are preconfigured with the **PROCESS** type. If the data series to be converted contain different measure types (**RELATION**, **OT_FUNC**, **OT_ORG**) they cannot be converted automatically. The configuration must be adapted manually before conversion.

### 9.2.4.6 Additional information: User-defined measures based on process instance-independent measures

If a user-defined measure created based on process instance-independent measures is used in the analysis with dimensions, which are defined for all measures involved, the user-defined measure only returns values if values of the relevant dimension step can be determined for all measures (intersection of dimension values involved).

When using process instance-independent measures with a strict dimension reference in the calculation of user-defined measures, take account of the following additional points:

- If two or more process instance-independent measures are used in a user-defined measure, the range of values of the individual dimensions to which the process instance-independent measures refer should be identical.
- In order for a process instance-independent measure to be included in the calculation rule for a user-defined measure, at least one dimension to which the process instance-independent measure refers must be registered at the process tree at the same point as the user-defined measure.

If these two requirements are not met, an information dialog like the following is shown when the user-defined measure is called up:

**Example**

The user-defined measure **Total costs** consists of process costs and overhead costs.
The process measure **Process costs** returns values based on process instances from the months of January to March 2001 and June to December 2001.
The process instance-independent measure **Overhead costs** has values for Jan to Jun 2001.

With a monthly analysis in 2001, the user-defined measure **Total costs** only returns values for the months of January, February, March and June 2001.

## 9.2.5 Definition of measure groups

Measure groups are defined in the configuration file **KiGroup.xml**. A distinction is made between the visible measure groups (**kigrouproot** or **kigroup**) and a single invisible measure group (**kigroupinvisible**). A measure can only be assigned to one group.

The grouping of measures does not represent a hierarchy or refinement of the measures; it is merely to improve the clarity.

The **kigroup** elements can be nested at any depth. This allows you to establish an individual folder structure for your measure groups.

Use PPM Customizing Toolkit to create measure groups. Your preferred group structure can be defined easily in the **Measures and dimensions** module.

### Example

```
<kigrouproot>
  <description language="de" name="Alle Kennzahlen">
    Diese Gruppe umfasst alle angezeigten Kennzahlen.
  </description>
  <description language="en" name="All measures">
    This group includes all displayed
                measures.
  </description>
  <kigroupinvisible>
    <description language="de"
              name="Unsichtbare Kennzahlen">
      Diese Gruppe umfasst alle Kennzahlen, die
      nur zur Berechnung weiterer Kennzahlen verwendet
      werden. Diese Kennzahlen werden nicht in der
      Kennzahlenliste angezeigt.
    </description>
    <description language="en" name="Invisible measures">
      This group includes all measures
      that are merely used for calculation
      of additional measures. These
      measures are not displayed in
      the measure list.
  </description>
  </kigroupinvisible>
  <kigroup name="KI_GROUP_COST">
    <description language="de" name="Kostenkennzahlen"/>
    <description language="en" name="Cost KPIs"/>
    <kigroup name="KI_GROUP_COST">
      <description language="de"
                            name="Kostenkennzahlen"/>
      <description language="en" name="Cost KPIs"/>
      <kigroup name="KI_GROUP_COST_ALL">
        <description language="de"
                        name="Gesamtkostenkennzahlen"/>
        <description language="en"
                             name="Total cost KPIs"/>
      </kigroup>
      <kigroup name="KI_GROUP_COST_AVERAGE">
       <description language="de
                 name="Durchschnittskostenkennzahlen"/>
       <description language="en"
                           name="Average cost KPIs"/>
      </kigroup>
    </kigroup>
  </kigroup>
  <kigroup name="KI_GROUP_TIME">
    <description language="de" name="Zeitenkennzahlen"/>
    <description language="en" name="Time KPIs"/>
  </kigroup>
  <kigroup name="KI_GROUP_QUALITY">
    <description language="de"
                         name="Qualitätskennzahlen"/>
    <description language="en" name="Quality KPIs"/>
  </kigroup>
</kigrouproot>
```

The **All measures** group includes the measures from all groups and subgroups except those from the **Invisible measures** group. Measures that you have not explicitly assigned to a specific

group are automatically assigned to the **All measures** group. Even measures that you have assigned to a group that does not exist are also assigned to this group.

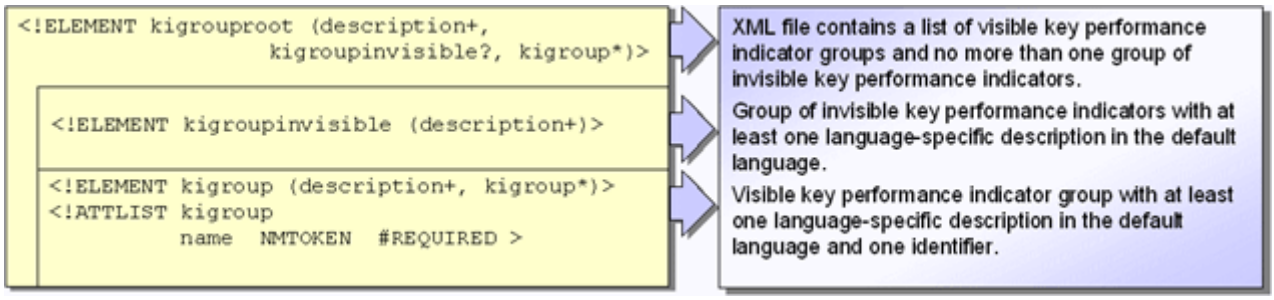
In the PPM front-end, the **Measures** tab displays all measures from the selected measure group and all subgroups, with the exception of measures from the **Invisible measures** group, which are only displayed if that group is selected.

The group of invisible measures **KI_GROUP_INVISIBLE** is located directly below the root and its structure cannot be extended.

**Warning**

The group identifiers **KI_GROUP_ROOT** and **KI_GROUP_INVISIBLE** are fixed by the PPM system and may not be used in a different context or changed.

The structure of the configuration file **KIGroup.xml** is specified by the DTD **KIGroup.dtd**:




## GENERAL STRUCTURE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kigrouproot SYSTEM 'KIGroup.dtd'>
<kigrouproot>
  <description language="de" name="...">...</description>
  <description language="en" name="...">...</description>
  <kigroupinvisible>
  <description language="de" name="...">...</description>
  <description language="en" name="...">...</description>
  </kigroupinvisible>
  <kigroup name="...">
  <description language="de" name="...">...</description>
  <description language="en" name="...">...</description>
  <kigroup name="...">
  <description language="de" name="..."/>
  <description language="en" name="..."/>
  <kigroup name="...">
  <description language="de" name="...">...</description>
  <description language="en" name="...">...</description>
  ...
  </kigroup>
  ...
  </kigroup>
  ...
  </kigroup>
  <kigroup name="...">
  <description language="de" name="..."/>
  <description language="en" name="..."/>
  ...
  </kigroup>
  <kigroup name="...">
  <description language="de" name="..."/>
```

```
  <description language="en" name="..."/>
  ...
  </kigroup>
  ...
</kigrouproot>
```

**Example**

The example below illustrates the display of the XML file in the PPM front-end interface:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kigrouproot SYSTEM 'KIGroup.dtd'>

<kigrouproot>
  <description language="de" name="Alle Kennzahlen">
    Diese Gruppe umfasst alle angezeigten Kennzahlen mit
    Ausnahme der unsichtbaren Kennzahlen.
  </description>
  <description language="en" name="All measures">
    This group includes all displayed measures
    except for the invisible
    ones.
  </description>
  <kigroupinvisible>
    <description language="de" name="Unsichtbare Kennzahlen">
      Diese Gruppe umfasst alle Kennzahlen, die nur
      zur Berechnung weiterer Kennzahlen verwendet werden.
      Diese Kennzahlen werden nicht in der
      Kennzahlenliste angezeigt.
    </description>
    <description language="en" name="Invisible KPIs">
      This group includes all measures
      that are merely used for calculation of additional
      measures. These measures
      are not displayed in the measure
      list, unless the group of
      invisible measures is selected.
    </description>
  </kigroupinvisible>
  <kigroup name="KI_GROUP_COST">
    <description language="de" name="Kostenkennzahlen"/>
    <description language="en" name="Cost KPIs"/>
  </kigroup>
  <kigroup name="KI_GROUP_TIME">
    <description language="de" name="Zeitenkennzahlen"/>
    <description language="en" name="Time KPIs"/>
  </kigroup>
  <kigroup name="KI_GROUP_QUALITY">
    <description language="de" name="Qualitätskennzahlen"/>
    <description language="en" name="Quality KPIs"/>
  </kigroup>
</kigrouproot>
```

### 9.2.5.1 Visible measure groups

A measure group is defined in the configuration file by the following XML element:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE kigrouproot SYSTEM 'KIGroup.dtd'>
<kigrouproot>
```

```
  <description language="de" name="...">
    Description text...
  </description>
  ...
  <kigroup name="...">
    <description language="de" name="..."/>
    <description language="en" name="..."/>
  </kigroup>
  ...
</kigrouproot>
```

| XML tag | Description |
|---|---|
| **kigrouproot** | Root of measure groups. Displayed as the top level group folder in the PPM front-end. |
| **description** | Language-specific description of the measure group root. Must be specified in at least the default language. |
| **kigroup** | Measure group to be defined. Each group can contain sub-groups. You can create any number of groups and sub-groups.<br>The group structure corresponds to a tree structure with any number of branches. |
| **name** | Internal name of the group. Referenced by the **kigroup** XML attribute from the measure definition (**kidef** XML element) in the measure configuration. |
| **description** | Language-specific description of a group, optionally with tooltip (**#PCDATA** section in the **description** element). The description must be specified in at least the default language. |

### 9.2.5.2 Group of invisible measures

The **Invisible measures** group contains all measures that are only displayed in the measure list when the group is actually selected. Only then are they available in the analysis. The invisible measures are not displayed in the measure lists for all other measure groups. The group of invisible measures is unique and cannot be structured.

Assign measures that are exclusively used as an interim result for the calculation of other measures to the group of invisible measures (**<kidef name="..." kigroup="KI_GROUP_INVISIBLE" ... />**).

The group is configured in the XML configuration file ***_kigroup.xml** of the corresponding CTK client template (under PPM installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ctk\examples\custom\) by the following element:

```
...
```

```
  <kigroupinvisible name="...">
    <description language="de" name="..."/>
    <description language="en" name=".."/>
  </kigroupinvisible>
...
```

## 9.3 Definition of dimensions

Dimensions are defined together with measures in the client-specific measure configuration. The PPM system makes a distinction between the following dimension types:

| Dimension | XML element | Description |
|---|---|---|
| One-level | **oneleveldim** | One-level dimensions are used if the number of dimension values is low and no meaningful grouping of the values is possible.<br>Example: Name of the source system, input channel for a call center (for example, call, fax, e-mail)<br>Attribute data type: All. Numerical attribute values are converted to text. |
| Two-level | **twoleveldim** | Two-level dimensions are used if meaningful grouping of the dimension values is possible.<br>Example: Process type group/Process type, Material type/Material<br>Attribute data type: All. Numerical attribute values are converted to text. Dimension values are saved as the **TEXTPAIR** data type. |
| N-level | **nleveldim** | For text dimensions with more than two levels. |
| Floating point number | **floatingdim** | Dimension is based on floating point values. The dimension values represent particular intervals. Example: Order volume<br>Attribute data type: **DOUBLE**, **TIMESPAN**, **PERCENTAGE** and all user-defined types derived from them |

| Dimension | XML element | Description |
|---|---|---|
| Time | **timedim** | Indicates the change of a measure over time.<br>Attribute data types:<br>**DAY**, **TIME** |
| Time of day | **hourdim** | Indicates the change of a measure over time.<br>Attribute data type: **TIMEOFDAY** |
| Period | **timerange** | Indicates the change of a measure within a specific period. |
| Search dimension | **searchdim** | Search for process instances using attribute values<br>Attribute data type: **TEXT**<br>Search dimensions cannot be displayed as dimensions in the analysis. |

The attributes required for creating text and floating point number dimensions are specified in a corresponding **dimitem** XML element. If the referenced attribute is an attribute to be calculated, this must be specified (**calculated="TRUE"**), so that the attribute can be calculated before creation of the dimension. In addition, you need to specify whether it is a process or function dimension.

When defining dimensions, ensure that the data type of the referenced attributes is compatible with the selected dimension type.

## 9.3.1 Definition of dimension groups

Dimension groups are defined in the configuration file **DimGroup.xml**. A distinction is made between visible dimension groups (**dimgroup** or **dimgrouproot**) and a single invisible dimension group (**dimgroupinvisible**). A dimension can only be assigned to one group.

The grouping of dimensions does not represent a hierarchy or refinement of the dimensions; it is merely to improve the clarity.

The **dimgroup** elements can be nested at any depth. This allows you to establish an individual folder structure for your dimension groups.

Use PPM Customizing Toolkit to create dimension groups. Your preferred group structure can be defined easily in the **Measures and dimensions** module.

In the PPM front-end, the **Dimensions** tab shows all dimensions in the selected dimension group and all sub-groups with the exception of dimensions from the **Invisible dimensions** group.

The invisible dimensions group **DIM_GROUP_INVISIBLE** is located directly below the root and its structure cannot be extended. It contains all internal dimensions and is not displayed in the user interface.

Dimensions that you do not assign to a dimension group are automatically assigned to the root **DIM_GROUP_ROOT** (**All dimensions** group).

**Warning**

The group identifiers **DIM_GROUP_ROOT** and **DIM_GROUP_INVISIBLE** are fixed by the PPM system and may not be used in a different context or changed.

**Example**

```
<dimgrouproot>
  <description name="Alle Dimensionen" language="de">
    Diese Gruppe beinhaltet alle angezeigten Dimensionen.
  </description>
  <description name="All dimensions" language="en">
    This group includes all displayed dimensions.
  </description>
  <dimgroupinvisible>
    <description name="Nicht sichtbare Dimensionen"
                              language="de">
      Diese Gruppe beinhaltet alle internen Dimensionen.
    </description>
    <description name="Invisible dimensions"
                              language="en">
      This group includes all internal dimensions.
    </description>
  </dimgroupinvisible>
  <dimgroup name="DIM_GROUP_CRITERIA">
    <description name="Criteria" language="de">
      Diese Gruppe beinhaltet Dimensionen, die
      als Unterscheidungskriterien dienen.
    </description>
    <description name="" language="en" />
    <dimgroup name="DIM_GROUP_CUST">
      <description name="Kundendimensionen" language="de">
        Diese Gruppe beinhaltet alle kundenrelevanten
        Dimensionen.
      </description>
      <description name="Customer dimensions"
                             language="en"/>
    </dimgroup>
    <dimgroup name="DIM_GROUP_PRINC">
      <description name="Auftraggeberdimensionen"
                              language="de">
        Diese Gruppe beinhaltet alle
        auftraggeberrelevanten Dimensionen.
      </description>
      <description name="Principal dimensions"
                             language="en"/>
      <dimgroup name="DIM_GROUP_USA">
        <description name="Auftraggeber in USA"
                              language="de">
          Diese Gruppe beinhaltet alle Dimensionen
          für Auftraggeber in den USA.
        </description>
        <description name="Customers USA" language="en" />
      </dimgroup>
      <dimgroup name="DIM_GROUP_EUROPE">
        <description name="Auftraggeber in Europa"
                                   language="de">
          Diese Gruppe beinhaltet alle Dimensionen für
          Auftraggeber in Europa.
```

```
      </description>
      <description name="Principals Europe"
                                  language="en"/>
    </dimgroup>
  </dimgroup>
</dimgroup>
<dimgroup name="DIM_GROUP_TIME">
  <description name="Zeitdimensionen"
                            language="de">
    Diese Gruppe beinhaltet alle Zeitdimensionen.
  </description>
  <description name="Time dimensions" language="en" />
</dimgroup>
</dimgrouproot>
```

## 9.3.2 Text dimensions

This dimension type comprises three types of dimension that use similar configurations for the definition of the relevant dimension levels: One-level, two-level, and n-level text dimensions. N-level dimensions can have any number of hierarchy levels.

Each definition of a dimension level (**leveldesc** XML element) is made up of an obligatory key (first **dimitem** XML element) and an optional description (second **dimitem** XML element). The language-specific interface names of the dimensions, keys, and descriptions for the individual levels (**description** XML elements) must be specified in the default language. The individual keys and descriptions refer to attributes of the **TEXT** type, which have values that PPM can display in the analysis and in the filter dialogs.

### 9.3.2.1 General XML structure

#### 9.3.2.1.1 One-level dimension

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                     'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
   ...
  <oneleveldim name="..." dimtype="..."
         internal="..." importmode="..." dimgroup="...">
    <description language="..." name="..."/>
    <leveldesc>
      <dimitem attrname="..." colname="..."
        calculated="..." location="..." substvalue="...">
         <description language="..." name="..."/>
         <defaultvalue="..."/>
      </dimitem>
      <dimitem attrname="..."
                       colname="..." calculated="...">
         <description language="..." name="..."/>
      </dimitem>
    </leveldesc>
  </oneleveldim>
   ...
</keyindicatorconfig>
```

### 9.3.2.1.2 Two-level dimension

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <twoleveldim name="..." dimtype="..." internal="..."
                      importmode="..." dimgroup="...">
    <description language="..." name="..."/>
    <leveldesc>
      <dimitem attrname="..." colname="..."
        calculated="..." location="..." substvalue="...">

        <description language="..." name="..."/>
        <defaultvalue="..."/>
      </dimitem>
      <dimitem attrname="..." colname="..."
                                  calculated="...">
        <description language="..." name="..."/>
        <defaultvalue="..."/>
      </dimitem>
    </leveldesc>
    <leveldesc>
      <dimitem attrname="..." colname="..."
                                  calculated="...">
        <description language="..." name="..."/>
        <defaultvalue="..."/>
      </dimitem>
      <dimitem attrname="..." colname="..."
                                  calculated="...">
        <description language="..." name="..."/>
        <defaultvalue="..."/>
      </dimitem>
    </leveldesc>
  </twoleveldim>
  ...
</keyindicatorconfig>
```

### 9.3.2.1.3 N-level dimension

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <nleveldim name="..." dimtype="..." internal="..."
                      importmode="..." dimgroup="...">
    <description name="..." language="..."/>
    <leveldesc>
      <dimitem attrname="..." colname="..."
        calculated="..." location="..." substvalue="...">
        <description language="..." name="..." />
        <defaultvalue="..."/>
      </dimitem>
      <dimitem attrname="..." colname="..."
```

```
                                  calculated="...">
      <description language="..." name="..."/>
      <defaultvalue="..."/>
    </dimitem>
  </leveldesc>
  <leveldesc>
    ...
  </leveldesc>
  <leveldesc>
    ...
  </leveldesc>
  ...
</nleveldim>
...
</keyindicatorconfig>
```

By default, the values of the individual dimension levels are displayed in the form **<Description (Key)>** in PPM, provided that descriptions have been defined. Otherwise, only the key is displayed as the value.

### Example

If the attribute referenced by the first **dimitem** contains the definition of a key **ID** and the attribute referenced by the second **dimitem** contains the corresponding description **Text**, the dimension values for this level are displayed by default in the form <**Text**> (<**ID**>) in the user interface.

The extract from the configuration file for a similar example looks like this.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                     'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <oneleveldim name="VTWEG" dimtype="PROCESS"
                 internal="no" importmode="OPTIONAL">
    <description language="de" name="Vertriebsweg"/>
    <description language="en"
                  name="Distribution channel"/>
    <leveldesc>
      <dimitem attrname="AT_VTWEG"
             colname="FIRST_ID" calculated="FALSE">
        <description language="de"
                      name="ID des Vertriebsweg"/>
        <description language="en"
                name="ID of distribution channel"/>
      </dimitem>
      <dimitem attrname="AT_VTWEG_NAME"
           colname="FIRST_DESC" calculated="FALSE">
        <description language="de"
                            name="Vertriebsweg"/>
        <description language="en"
                  name="Distribution channel"/>
      </dimitem>
    </leveldesc>
  </oneleveldim>
  ...
</keyindicatorconfig>
```

The **dimitem** XML element configures the following settings for the key or description of a dimension level:

| XML tag | Description |
|---|---|
| **attrname** | Name of the referenced attribute. Only the **TEXT** data type is permitted. |
| **calculated** | **TRUE**: The attribute value is calculated.<br>The default value is **FALSE**. |
| **location** | Only for dimtype="RELATION"<br>Valid values: **SOURCE** (attribute placement on source reference object of relation)<br>**TARGET** (attribute placement on target reference object of relation)<br>**THIS** (default value: attribute is placed at the relation itself) |
| **defaultvalue** (optional) | Specifies a default value that is displayed if no attribute value can be retrieved and if no value is or can be retrieved using **substvalue**. If neither **defaultvalue** nor **substvalue** has been specified, the value of the **PPM_NULL** key from the file **Database_settings.properties** is displayed if an attribute value cannot be retrieved. |
| **substvalue** (optional) | Specifies a substitute value that is displayed if no attribute value can be retrieved. The attribute value from the previous, rougher level (**PRED**) or the next, more detailed level (**SUCC**) can be used as a substitute value. Substitute values may cover several consecutive levels. If no value can be retrieved using the specifications for **substvalue** (for example, **substvalue="SUCC"** for a one-level dimension), no substitute value is displayed. Default value: **NONE** (no substitute value) |

| XML element | Description |
|---|---|
| **compression value** (optional sub-element for **dimitem**) | The internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** must be specified (**Configure the internal aggregation attribute** (page 213)).<br>Only for **dimtype="PROCESS"**<br>Identical and differing dimension values are deleted when permanently aggregating using the command |

| XML element | Description |
|---|---|
| | prompt (**runppmcompress**) (see **PPM Operation Guide**) and they are replaced by the specified aggregation value (**Change aggregation behavior** (page 213)). |

## SUBSTITUTE AND DEFAULT VALUES FOR ONE-, TWO-, AND N-LEVEL DIMENSIONS

You can specify default values and substitute values for the keys and descriptions of each individual dimension level. These values are displayed if no attribute value can be retrieved. When you select the dimension value to be displayed, the sequence is as follows:

1. Attribute value
2. Substitute value (**substvalue**)
3. Default value (**defaultvalue**)
4. DB default value (value of the **PPM_NULL** key in the file **Database_settings.properties**)

**Warning**

The two-level **Process type** dimension does not support default or substitute values. If you specify these in the configuration, they are deleted during the import.

**Example (file extract from measure configuration)**

```
...
<nleveldim name="SALE" dimtype="PROCESS"
                       dimgroup="DIM_GROUP_CRITERIA">
  <description name="Sales" language="en"/>
  <leveldesc>
    <dimitem attrname="AT_VKORG" colname="NAME_1"
                               calculated="FALSE">
      <description language="en"
                         name="Sales organization"/>
    </dimitem>
    <dimitem attrname="AT_VKORG_NAME"
              colname="DESC_NAME_1" calculated="FALSE">
      <description language="en"
                   name="Name of sales organization"/>
    </dimitem>
  </leveldesc>
  <leveldesc>
    <dimitem attrname="AT_DIVISION" colname="NAME_2"
                 calculated="FALSE" substvalue="SUCC">
      <description language="en" name="Division"/>
      <defaultvalue>defaultvalue 2nd level ID
      </defaultvalue>
    </dimitem>
    <dimitem attrname="AT_DIVISION_NAME"
              colname="DESC_NAME_2" calculated="FALSE"
                               substvalue="SUCC">
      <description language="en" name="Division name"/>
      <defaultvalue>defaultvalue 2nd level description
      </defaultvalue>
    </dimitem>
```

```
  </leveldesc>
  <leveldesc>
    <dimitem attrname="AT_VTWEG" colname="NAME_3"
                                 calculated="FALSE">
      <description language="en"
                         name="Distribution channel"/>
    </dimitem>
    <dimitem attrname="AT_VTWEG_NAME"
              colname="DESC_NAME_3" calculated="FALSE">
      <description language="en"
                  name="Name of distribution channel"/>
    </dimitem>
  </leveldesc>
</nleveldim>
...
```

Substitute values (**substvalue**) are defined for the key and description of the second level of the **SALE** n-level dimension. These substitute values are transferred to the subsequent third level. If no substitute value can be retrieved, the specified default value (**defaultvalue**) is displayed instead.

Text dimensions are normally based on alphanumeric attribute types. The dimension values are displayed in the interface in alphanumeric order.
When using attributes based on numerical data types, the dimension values are written to the database as strings and are sorted in numerical order when displayed in the interface.

## 9.3.2.2 Configuration

### 9.3.2.2.1 One-level dimensions

A one-level dimension (**oneleveldim** XML element) is described in full by only one level definition (**leveldesc** XML element). The language-specific names of the dimension, key, and description are specified in the **description** XML elements.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <oneleveldim name="VTWEG" dimtype="PROCESS"
                  internal="no" importmode="OPTIONAL">
    <description language="de" name="Vertriebsweg"/>
    <description language="en"
                         name="Distribution channel"/>
    <leveldesc>
      <dimitem attrname="AT_VTWEG" colname="FIRST_ID"
                                 calculated="FALSE">
        <description language="de"
                          name="ID des Vertriebsweg"/>
        <description language="en"
                     name="ID of distribution channel"/>
      </dimitem>
      <dimitem attrname="AT_VTWEG_NAME"
               colname="FIRST_DESC" calculated="FALSE">
```

```
      <description language="de" name="Vertriebsweg"/>
      <description language="en"
                      name="Distribution channel"/>
    </dimitem>
   </leveldesc>
  </oneleveldim>
  ...
</keyindicatorconfig>
```

| XML attribute | Description |
|---|---|
| **name** | Name of the dimension. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| **dimtype** | Dimension type: Valid values:<br>**PROCESS** (process dimension)<br>**FUNCTION** (function dimension, obsolete, only to be used for compatibility reasons)<br>**OT_FUNC** (function dimension)<br>**RELATION** (relation dimension)<br>**OT_ORG** (organizational dimension) |
| **internal** | Internal use of the dimension<br>**yes**: The dimension is used internally and is not displayed in the interface.<br>The default value is **no**. |
| **importmode** | Output of error messages when calculating dimension values<br>**OPTIONAL**: Calculation errors are not output.<br>**MANDATORY**: Calculation errors are output<br>Default value: **OPTIONAL** |
| **dimgroup** (optional) | Dimension group to which the dimension is assigned |

| XML element | Description |
|---|---|
| **compression value** (optional sub-element for **dimitem**) | The internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** must be specified (**Configure the internal aggregation attribute** (page 213)).<br>Only for **dimtype="PROCESS"**<br>Identical and differing dimension values are deleted when permanently aggregating using the command prompt (**runppmcompress**) (see **PPM Operation** |

| XML element | Description |
|---|---|
| | **Guide**) and they are replaced by the specified aggregation value (**Change aggregation behavior** (page 213)). |

To avoid the **not specified** dimension step for a dimension, specify **importmode="MANDATORY"** so that process instances that cannot be assigned to any dimension step are identified when importing the data by the output of a corresponding message.

### INTERNAL DIMENSIONS

Internal dimensions (**internal="yes"**) are not displayed in the PPM front-end interface. They are used to distinguish administrative process instance characteristics.

### USING MULTI-BYTE CHARACTER SETS

The following file extract from the measure configuration shows an example of the definition options for one-level dimensions when using a multi-byte character set:

```
...
<oneleveldim name="D_PRODUCT_GR" dimtype="FUNCTION"
                  internal="no" importmode="OPTIONAL">
  <description name="Produktgruppe" language="de"/>
  <description name="Product group" language="en"/>
  <description name="Ομάδα προϊόντων" language="el"/>
  <leveldesc>
    <dimitem attrname="ΙΔ_ΣΥΝ_ΠΡΟΪΟΝΤ_ΤΑΥΤ"
            colname="Column name_3" calculated="FALSE">
      <description language="de"
                           name="ID Produktgruppe"/>
      <description language="en"
                           name="Product group ID"/>
      <description language="el"
                  name="Ταυτότητα ομάδας προϊόντων"/>
    </dimitem>
    <dimitem attrname="ΙΔ_ΣΥΝ_ΠΡΟΪΟΝΤ_ΠΕΡΙΓΡ"
            colname="Column name_4" calculated="FALSE">
      <description language="de"
                  name="Beschreibung Produktgruppe"/>
      <description language="en"
                   name="Product group description"/>
      <description language="el"
               name="Περιγραφή της ομάδας προϊόντων"/>
    </dimitem>
  </leveldesc>
</oneleveldim>
...
```

## 9.3.2.2.2 Two-level dimensions

Two-level dimensions (**twoleveldim** XML element) are configured in the same way as one-level dimensions, except that they consist of two level descriptions (**leveldesc** XML elements). Multi-byte character sets are used in the same way as for one-level dimensions (see chapter **One-level dimension** (page 162).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <twoleveldim name="PROCESSTYPE" dimtype="PROCESS"
                              importmode="OPTIONAL">
      <description language="de" name="Prozesstyp"/>
      <description language="en" name="Process type"/>
      <leveldesc>
         <dimitem attrname="AT_PROCTYPEGROUP"
            colname="PROCTYPEGROUP" calculated="FALSE">
            <description language="de"
                            name="Prozesstypgruppe"/>
            <description language="en"
                           name="Process type group"/>
         </dimitem>
      </leveldesc>
      <leveldesc>
         <dimitem attrname="AT_PROCTYPE"
                 colname="PROCTYPE" calculated="FALSE">
            <description language="de"
                                 name="Prozesstyp"/>
            <description language="en"
                                name="Process type"/>
         </dimitem>
      </leveldesc>
  </twoleveldim>
  ...
</keyindicatorconfig>
```

| XML attribute | Description |
|---|---|
| **name** | Internal name of the dimension. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| **dimtype** | Dimension type: Valid values:<br>**PROCESS** (process dimension)<br>**FUNCTION** (function dimension, obsolete, only to be used for compatibility reasons)<br>**OT_FUNC** (function dimension)<br>**RELATION** (relation dimension)<br>**OT_ORG** (organizational dimension) |
| **internal** | Internal use of the dimension<br>**yes**: The dimension is used internally and is not displayed in the interface.<br>The default value is **no**. |
| **importmode** | Output of error messages when calculating dimension values<br>**OPTIONAL**: Calculation errors are not output. |

| XML attribute | Description |
|---|---|
| | **MANDATORY**: Calculation errors are output<br>Default value: **OPTIONAL** |
| **dimgroup** (optional) | Dimension group to which the dimension is assigned |

| XML element | Description |
|---|---|
| **compression value** (optional sub-element for **dimitem**) | The internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** must be specified (**Configure the internal aggregation attribute** (page 213)).<br>Only for **dimtype="PROCESS"**<br>Identical and differing dimension values are deleted when permanently aggregating using the command prompt (**runppmcompress**) (see **PPM Operation Guide**) and they are replaced by the specified aggregation value (**Change aggregation behavior** (page 213)). |

## 9.3.2.2.3 N-level dimensions

N-level dimensions are configured in the same way as one-level and two-level dimensions. An n-level dimension consists of at least one and no more than <**n**> levels (**leveldesc** XML elements). Each level contains a key (first **dimitem** XML element) and an optional description (second **dimitem** XML element). All language-specific designations (name of dimension, key, or description) are specified with the **description** XML element. The XML attributes are identical to those used for one-level and two-level dimensions.

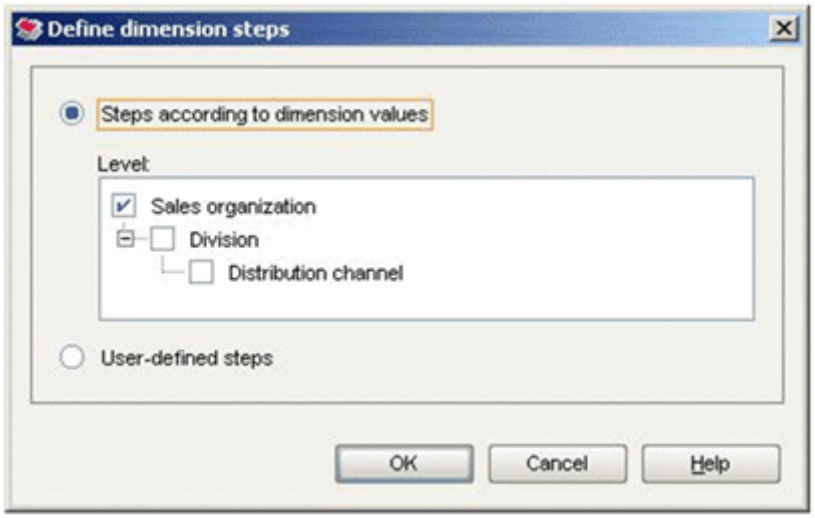
**Example**

(n-level dimension with three levels, with a key and description defined for each level)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <nleveldim name="SALE" dimtype="PROCESS"
                       dimgroup="DIM_GROUP_CRITERIA">
  <description name="Sales" language="de"/>
  <leveldesc>
    <dimitem attrname="AT_VKORG"
                   colname="NAME_1"calculated="FALSE">
      <description language="de"
                        name="Verkaufsorganisation"/>
    </dimitem>
    <dimitem attrname="AT_VKORG_NAME"
```

```
              colname="DESC_NAME_1" calculated="FALSE">
      <description language="de" name="Name der
                              sales organization"/>
    </dimitem>
  </leveldesc>
  <leveldesc>
    <dimitem attrname="AT_DIVISION"
                  colname="NAME_2" calculated="FALSE">
      <description language="de" name="Division"/>
    </dimitem>
    <dimitem attrname="AT_DIVISION_NAME"
              colname="DESC_NAME_2" calculated="FALSE">
      <description language="de" name="Division name"/>
    </dimitem>
  </leveldesc>
  <leveldesc>
      <dimitem attrname="AT_VTWEG"
                  colname="NAME_3" calculated="FALSE">
        <description language="de" name="Vertriebsweg"/>
      </dimitem>
      <dimitem attrname="AT_VTWEG_NAME"
              colname="DESC_NAME_3" calculated="FALSE">
        <description language="de"
                        name="Name des Vertriebswegs"/>
      </dimitem>
    </leveldesc>
  </nleveldim>
  ...
</keyindicatorconfig>
```

The **Sales** dimension is displayed as follows in the PPM user interface:




Multi-byte character sets are used in the same way as for one-level dimensions (see chapter **One-level dimensions** (page 167).

#### 9.3.2.3 Import dimension values

For one, two, and n-level dimensions, you can use the **rundimdata** command line program to import data before the actual PPM import takes place. In this case, the values are imported as pairs in the form **<key-description>** for each dimension level. Note that a key must be specified, while the description is optional.

You will find detailed information about importing dimension values for text dimensions in advance of the actual PPM import in the PPM Data Import manual.

### 9.3.3 Floating point dimensions

Floating point dimensions (**floatingdim** XML element) are configured in the same way as one-level dimensions except that the attributed referenced using the **dimitem** XML element must be a numerical data type (for example, **DOUBLE**, **LONG, TIMESPAN**, **FACTORYTIMESPAN, FREQUENCY, PERCENTAGE**, and all user-defined data types such as **COST**).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <floatingdim name="ORDERVOL" dimtype="PROCESS"
                              importmode="OPTIONAL">
    <description language="de" name="Auftragsvolumen"/>
    <description language="en" name="Order size"/>
    <dimitem attrname="AT_ORDERVOL" colname="ORDERVOL"
                                 calculated="FALSE">
      <description language="de" name="Umsatz"/>
      <description language="en" name="Sales revenues"/>
    </dimitem>
  </floatingdim>
  ...
</keyindicatorconfig>
```

| XML tag | Description |
|---|---|
| **name** | Internal name of the dimension. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| **dimtype** | Dimension type: Valid values:<br>**PROCESS** (process dimension)<br>**FUNCTION** (function dimension, obsolete, only to be used for compatibility reasons)<br>**OT_FUNC** (function dimension)<br>**RELATION** (relation dimension)<br>**OT_ORG** (organizational dimension) |

| XML tag | Description |
|---|---|
| **dimitem location** (optional) | Only for **dimtype="RELATION"**<br>Valid values: **SOURCE** (attribute placement on source reference object of relation)<br>**TARGET** (attribute placement on target reference object of relation)<br>**THIS** (default value: attribute is placed at the relation itself) |
| **importmode** (optional) | Output of error messages when calculating dimension values<br>**OPTIONAL**: Calculation errors are not output.<br>**MANDATORY**: Calculation errors are output<br>Default value: **OPTIONAL** |

## 9.3.4 Time dimensions

The configuration characteristics of a time dimension are summarized in the **timedim** XML element.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <timedim name="DAY" dimtype="PROCESS" precision="DAY"
              attrname="AT_DAY" calculated="TRUE"
                                importmode="OPTIONAL">
    <description language="de" name="Tag"/>
    <description language="en" name="Day"/>
  </timedim>
  ...
</keyindicatorconfig>
```

| XML tag | Description |
|---|---|
| **name** | Internal name of the dimension, displayed in paramset. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| **dimtype** | Dimension type: Valid values:<br>**PROCESS** (process dimension)<br>**FUNCTION** (function dimension, obsolete, only to be used for compatibility reasons)<br>**OT_FUNC** (function dimension)<br>**RELATION** (relation dimension)<br>**OT_ORG** (organizational dimension) |

| XML tag | Description |
|---|---|
| **location** (optional) | Only for **dimtype="RELATION"**<br>Valid values: **SOURCE** (attribute placement on source reference object of relation)<br>**TARGET** (attribute placement on target reference object of relation)<br>**THIS** (default value: attribute is placed at the relation itself) |
| **attrname** | Internal name of the referenced attribute |
| **precision** (optional) | Most detailed step width of the dimension (**DAY**, **HOUR**, **MINUTE**) |
| **calculated** | **TRUE**: Attribute value must be calculated.<br>Default value: **FALSE**. |
| **internal** | Mark as internal dimension with **yes**. Default value: **no** |
| **earlyalert** | Mark as critical dimension in Early alert system with **yes**. Default value: **no** |
| **importmode** (optional) | Output of error messages when calculating dimension values<br>**OPTIONAL**: Calculation errors are not output.<br>**MANDATORY**: Calculation errors are output<br>Default value: **OPTIONAL** |
| **dimgroup** (optional) | Name of the dimension group to which the time dimension is to be assigned. |
| **deleteon compression** | The internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** must be specified (**Configure the internal aggregation attribute** (page 213)).<br>Only for **dimtype="PROCESS"**<br>**TRUE**: Identical and differing dimension values are deleted when permanently aggregating using the command prompt (**runppmcompress**) (see **PPM Operation Guide**) (**Change aggregation behavior** (page 213)).<br>**FALSE**: When permanently aggregating via command prompt, identical dimension values are transferred to the aggregated EPC, while differing values are deleted.<br>Default setting: **FALSE** |

### 9.3.4.1 Time dimensions for the Early alert system

To monitor critical times in individual process instances as part of the Early alert system, you can define special time dimensions in the measure configuration. These time dimensions are identified by the **earlyalert** attribute having the value **yes** (CTK: Enable the **Early alert** check box). For all other time dimensions, this attribute has the value **no** by default.

- For better differentiation, the name of the dimension (**name**) should begin with the prefix **CRIT_**.
- As critical time dimensions are only calculated internally at the process instance level, the **internal** attribute must have the value **yes** (CTK: enable the corresponding check box in the **Internal** column) and the **dimtype** attribute must have the value **PROCESS** (CTK: Select the value **PROCESS** in the **Usage** column). Otherwise, an error message is output when importing the configuration.

Time dimensions from the Early alert system are internal dimensions and are therefore not displayed in the dimension list in the navigation structure. For this reason, the assignment to a dimension group is ignored.

### 9.3.4.2 Special feature for calculation of critical time attributes

In the calculation rule (**calcattr**), for the critical time attribute, you must assign the **delete** XML attribute the value **yes** (CTK: Enable the **Delete attribute value** check box).

**Example**

Extract from the calculation rule for the **Critical goods issue date** attribute

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                        "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
...
   <calcattr name="AT_CRITICAL_WAUS_DATE"
                        type="PROCESS" delete="yes">
...
</keyindicatorconfig>
```

Use PPM Customizing Toolkit to define, calculate, and register critical time dimensions.

In the **Measures and dimensions** module, you can use the **Dimensions** component to conveniently enter the required definition information. You can create the definitions of the calculation rules for the critical times in the **Calculated attribute types** component.

Register critical time dimensions to the preferred process type groups or process types using the **Process tree** component on the **Process analysis dimensions** tab in the **Processes** module.

### Example

The following example calculates the **Critical goods issue date**. Instances in which no goods issue has been posted within four days of the **Create delivery** function (**SAP.LIEF**) being executed are classed as critical, that is, the **Post goods issue** function (**SAP.WAUS**) does not occur in the process instance. The calculation rule for the **AT_CRITICAL_WAUS_DATE** attribute calculates the critical time by adding a time span of four days (**354600 seconds**) to the time of the earliest occurrence of the **Create delivery** function.
The calculation is made to the nearest hour (**precision="hour"** XML attribute for the **CRT_TIME_WAUS** dimension). For example, if the earliest reference time (**AT_TIME**) for the **Create delivery** function in a process instance is **13.07.03 20:26:55**, based on the above calculation rule the critical goods issue date, correct to the nearest hour, is thus calculated as **17.07.03 20:00**.

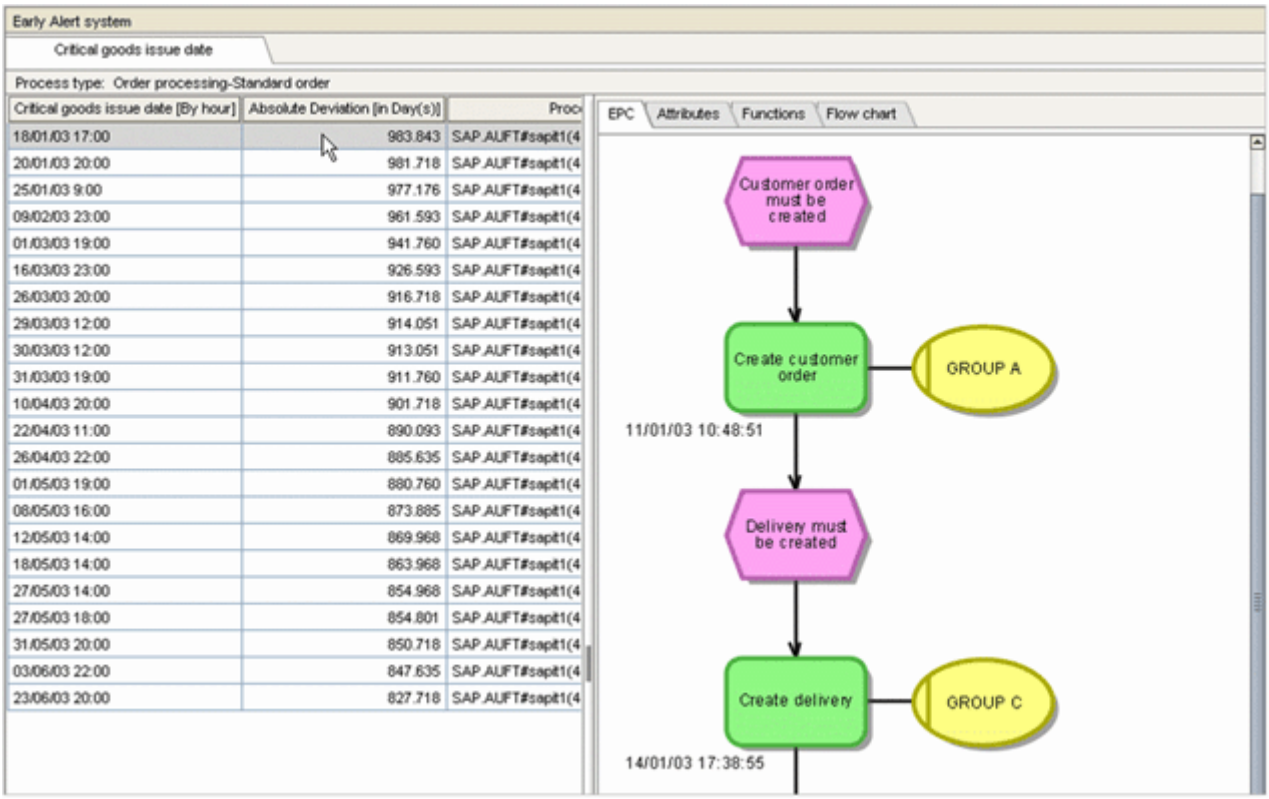
The result of a calculated critical time attribute must always be available in a time stamp format (**TIME** data type).

This extract from the measure configuration shows the definition of the critical time dimension **CRIT_TIME_WAUS**. The attribute and value combination **earlyalert="yes"** identifies it as an Early alert system time dimension.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      "KeyindicatorConfiguration.dtd">
<keyindicatorconfig>
  ...
  <calcattr name="AT_CRITICAL_WAUS_DATE"
                        type="PROCESS" delete="yes">
    <calculation>
      <if>
        <not>
          <exists>
            <attribute name="AT_OBJNAME_INTERN" nodetype=
                       "OT_FUNC" objectname="SAP.WAUS"
                                 onerror="CONTINUE"/>
          </exists>
        </not>
      <then>
        <addtimespan>
          <min>
            <attribute name="AT_TIME" nodetype="OT_FUNC"
                                objectname="SAP.LIEF"
                         onerror="EXIT_NO_WARNING"/>
          </min>
          <constant>
            <dataitem value="345600.0">
              4,000
              <datatype name="TIMESPAN">Time span
              </datatype>
              <scale name="DAY" factor="86400.0">Day(s)
              </scale>
            </dataitem>
          </constant>
        </addtimespan>
      </then>
      </if>
    </calculation>
```

```
  </calcattr>
  ...
  <timedim name="CRIT_TIME_WAUS" dimtype="PROCESS"
      precision="HOUR" attrname="AT_CRITICAL_WAUS_DATE"
      calculated="TRUE" internal="yes" earlyalert="yes"
      importmode="OPTIONAL">
  <description name="kritischer Warenausgangstermin"
                                    language="de"/>
  </timedim>
  ...
</keyindicatorconfig>
```

The **Early alert system** component of the **Instance controlling** module in the PPM front-end checks whether critical goods issue dates have been exceeded. In the example, it is assumed that the critical dimension **CRIT_TIME_WAUS** is registered at the **Order processing\Standard order** process type.




The current deviation from the critical goods issue date at the time of execution is specified in the analysis area in the **Absolute deviation [Days]** column. The execution time for the early alert check is the current system time. In the example, after executing the early alert check for the **Critical goods issue date** dimension, all process instances of the **Standard order** process type in which the critical goods issue date has been exceeded are displayed in a process instance table. For the process instance selected in the illustration the critical goods issue date has currently been exceeded by 983.843 days. This is the absolute deviation in days from the critical goods issue date at the current execution time (in the example 10.08.05 15:31).

Alternatively, you can identify critical process instances using the **runppmanalytics** command line program using the **-earlyalert** option (see **PPM Operation Guide**).

## 9.3.5 Time range dimensions

Time range dimensions are special time dimensions. They enable users to observe process states based on a past period (start time to end time).

Three variants exist.

- Due date-related time range dimension, based on the start time
- Due date-related time range dimension, based on the end time
- Interval-based time range dimension

Time range dimensions are defined by the XML element **timerangedim**.

**Example**

A time range dimension is configured using the following syntax, for example.

```
<timerangedim name="RANGEDIM_KEYWORD" reference="END" dimtype="PROCESS">
    <startattribute name="AT_START_TIME" calculated="TRUE"/>
    <endattribute name="AT_END_TIME" calculated="TRUE"/>
    <description name="Display name" language="en"></description>
</timerangedim>
```

| XML tag | Description |
|---|---|
| **timerangedim** | |
| ▪ name | Internal name of the dimension, displayed in paramset. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| ▪ dimtype | Dimension type: Valid values:<br>**PROCESS** (process dimension)<br>**FUNCTION** (function dimension, obsolete, only to be used for compatibility reasons)<br>**OT_FUNC** (function dimension)<br>**RELATION** (relation dimension)<br>**OT_ORG** (organizational dimension) |
| ▪ reference | Defines whether it is a due date-related time range dimension with a start time (**value = "START"**) or end time (**value="END"**), or an interval-based time range dimension (**value="RANGE"**).<br>Specification = Optional<br>Default value = **"END"**<br>Changing the type **START**, **END**, or **RANGE** at a later time is not allowed and prevented by the configuration import. |

| XML tag | Description |
|---|---|
| ▪ internal | Mark as internal dimension with **yes**. Default value: **no** |
| ▪ importmode | Output of error messages when calculating dimension values<br>**OPTIONAL**: Calculation errors are not output.<br>**MANDATORY**: Calculation errors are output<br>Default value: **OPTIONAL** |
| ▪ dimgroup | Name of the dimension group to which the time dimension is to be assigned.<br>Specification: Optional |
| **startattribute/ endattribute** | |
| ▪ name | Specify the EPC attribute based on which the dimension value of the start or end time is to be calculated.<br>Attribute type = TIME<br>Specification = Mandatory |
| ▪ calculated | **TRUE**: Attribute value must be calculated.<br>Default value: **FALSE**. |
| ▪ location | Only for **dimtype="RELATION"**<br>Valid values: **SOURCE** (attribute placement on source reference object of relation)<br>**TARGET** (attribute placement on target reference object of relation)<br>**THIS** (default value: attribute is placed at the relation itself)<br>Specification: Optional |
| **description** | |
| ▪ name | Language-specific interface name of the dimension in the PPM front-end. |
| ▪ language | Language in which the interface name is displayed |

## 9.3.6 Time of day dimensions

The configuration characteristics of a time of day dimension are summarized in the **hourdim** XML element.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <hourdim name="TIMEOFDAY" dimtype="PROCESS"
          attrname="AT_TIME_OF_DAY" tablename="DAYTIME"
          precision="SECOND" calculated="TRUE"
                              importmode="OPTIONAL">
    <description language="de" name="Uhrzeit"/>
    <description language="en" name="Time of day"/>
  </hourdim>
  ...
</keyindicatorconfig>
```

For each step width of a time of day dimension (hour, minute, second), a separate column is created in the database table for the dimension.

| XML tag | Description |
|---|---|
| **name** | Name of the dimension. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| **dimtype** | Dimension type: Valid values:<br>**PROCESS** (process dimension)<br>**FUNCTION** (function dimension, obsolete, only to be used for compatibility reasons)<br>**OT_FUNC** (function dimension)<br>**RELATION** (relation dimension)<br>**OT_ORG** (organizational dimension) |
| **location** (optional) | Only for **dimtype="RELATION"**<br>Valid values: **SOURCE** (attribute placement on source reference object of relation)<br>**TARGET** (attribute placement on target reference object of relation)<br>**THIS** (default value: attribute is placed at the relation itself) |
| **attrname** | Name of the referenced attribute |
| **precision** | Most detailed step width of the dimension (**HOUR**, **MINUTE**, **SECOND**). Default value: **HOUR** |
| **calculated** | **TRUE**: Attribute value must be calculated. |

| XML tag | Description |
|---|---|
| | Default value: **FALSE** |
| **importmode** | Output of error messages when calculating dimension values<br>**OPTIONAL**: Calculation errors are not output.<br>**MANDATORY**: Calculation errors are output<br>Default value: **OPTIONAL** |
| **deleteon compression** | The internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** must be specified (**Configure the internal aggregation attribute** (page 213)).<br>Only for **dimtype="PROCESS"**<br>**TRUE**: Identical and differing dimension values are deleted when permanently aggregating using the command prompt (**runppmcompress**) (see **PPM Operation Guide**) (**Change aggregation behavior** (page 213)).<br>**FALSE**: When permanently aggregating via command prompt, identical dimension values are transferred to the aggregated EPC, while differing values are deleted.<br>Default setting: **FALSE** |

## 9.3.7 Search dimensions

You can use this special dimension type to search for process instances using particular values for a search attribute. The search dimension acts like a filter on the set of currently available process instances. As for the other dimension types, the set filter expression can be edited or removed. Several search dimension filters can be used simultaneously in an analysis.

A search criterion is specified in the front-end using the **Edit filter** pop-up menu for the search dimension. Alternatively, search criteria can be specified in the Process Instance Search Wizard.

A search criterion consists of a string. Optionally, a placeholder **?** or ***** can be used at the end of the string. The ***** symbol stands for any sequence of characters, while the **?** symbol stands for any single character.

The configuration of search dimensions includes the following simplifications compared to the other dimension types:

- Search dimensions are based exclusively on process attributes.
- Search attributes must be of the **TEXT** type.

The configuration characteristics of a search dimension are summarized in the **searchdim** XML element:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <searchdim name="BELEGNR">
    <description language="de" name="Suche Belegnummer"/>
    <description language="en"
                 name="Searching for document number"/>
    <dimitem attrname="AT_SAP_BELEGNR"
                  colname="BELEGNR" calculated="TRUE">
      <description language="de" name="Belegnummer"/>
      <description language="en" name="Document number"/>
    </dimitem>
  </searchdim>
  ...
</keyindicatorconfig>
```

A separate column is created in the database table for each search dimension.

| XML tag | Description |
|---|---|
| **name** | Internal name of the dimension. A table is created in the database under this name. For the specified name, the guidelines described in the **Table name** chapter are applicable. |
| **description name** | Language-specific interface name of the dimension in the PPM front-end. |
| **dimitem** | Definition of a column in the database table. Each search dimension has a single **dimitem** element, in which the dimension is described. |
| **attrname** | Name of the attribute used as a basis for the search dimension |
| **colname** | Name of the data column in the info cube. If this is not specified, the dimension name is used.<br>For the specified name, the guidelines described in the **Table name** chapter apply. |
| **calculated** | **TRUE**: Attribute value must be calculated.<br>Default value: **FALSE** |
| **compression value** (optional sub-element for **dimitem**) | The internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** must be specified (**Configure the internal aggregation attribute** (page 213)).<br>Only for **dimtype="PROCESS"**<br>Identical and differing dimension values are deleted when permanently aggregating using the command prompt (**runppmcompress**) (see **PPM Operation** |

| XML tag | Description |
|---|---|
| | **Guide**) and they are replaced by the specified aggregation value (**Change aggregation behavior** (page 213)). |

Search dimensions are written to the info cube for the process type group. If you are using several search dimensions in a process type group, you need to give the data columns different names.

## 9.3.8 Variant dimension

With PPM 10.1 the new **VARIANTDIM** dimension type has been introduced.

Variants classify process instances according to their structure. The relevant structure is the sequence of functions in a process instance. **Variant** has two dimension levels, **Combined variant** (rough step width) and **Precise variant** (refined step width), and the **Name** as dimension value.

More basic information about variants, a list with affected functionalities, and how to add the variant feature to PPM can be found in the documentation **PPM Customizing Toolkit**.

### 9.3.8.1 Attribute configuration

The following attribute types are defined as attributes for the **VARIANT** dimension type. They are of type **Text** and are part of the attribute configuration.

#### ATTRIBUTE TYPES

```
<attributedefinition key="AT_INTERNAL_FUNCTION_FLOW_VARIANT" type="TEXT"
group="AG_INTERNAL" />
<attributedefinition key="AT_INTERNAL_PRECISE_VARIANT" type="TEXT"
group="AG_INTERNAL" />
```

#### ATTRIBUTE NAMES

```
<attribute key="AT_INTERNAL_FUNCTION_FLOW_VARIANT" name="Internal combined
variant" />
<attribute key="AT_INTERNAL_PRECISE_VARIANT" name="Internal precise variant" />
```

Because of a specific semantic it is not recommended to import values or to define an attribute calculator. See chapter Usage of variant attributes during import (page 186).

### 9.3.8.2 Measure configuration - dimension type

The **variantdim** element is used to define the variant dimension. It has just three attributes and nested **leveldesc** elements for configuring the attributes used to feed the two levels.

Attributes of the **variantdim** element are:

| XML tag | Description |
|---|---|
| name | keyword as used in paramset (required) |
| comment | comment for dimension (optional) |
| dimgroup | name of the group this dimension is assigned to (optional) |

Each dimension level (leveldesc) has exactly one **dimitem** as value and a description is not allowed for both dimension level. Only the **attrname** element is required for the **dimitem** element.

| XML tag | Description |
|---|---|
| **attrname** | Name of the referenced attribute, containing the dimension data. Only the **TEXT** data type is permitted. |

Typically variant dimension is defined as follows.

```
<variantdim name="D_EPC_VARIANT">
<description language="en" name="Variant" />
        <description language="de" name="Variante" />
        <leveldesc>
                <dimitem attrname="AT_INTERNAL_FUNCTION_FLOW_VARIANT">
                        <description language="en" name="Combined variant" />
                        <description language="de" name="Kombinierte Variante" />
                </dimitem>
        </leveldesc>
        <leveldesc>
                <dimitem attrname="AT_INTERNAL_PRECISE_VARIANT">
                        <description language="en" name="Precise variant" />
                        <description language="de" name="Präzise Variante" />
                </dimitem>
        </leveldesc>
</variantdim>
```

### 9.3.8.3 Process tree configuration

Variant dimensions can be assigned to process tree nodes similar to all other dimension types by using the **<usedim>** element.

```
<usedim name="D_EPC_VARIANT" />
```

Optionally, the <usedim> assignment can contain a default refinement. If it is not the case, the coarsest level is used as refinement similar to all other **Text** dimension types.

#### 9.3.8.4 Usage of variant attributes during import

The use of the variant attributes during merge and attribute mapping is not forbidden but strongly discouraged. The values of variant process attributes are overwritten by the variant calculation, which is only happening after all other calculations have already been processed. So neither one can use the results of the variant calculation nor the fully processed EPCs contain the values set outside the variant calculation.

There is a special danger in filling the dimension attributes by calculation rules or through mapping: When you do that while the variant dimension is not registered at the process type of the EPC, these values will not be overwritten. When you then register the dimension, re-initialize the analysis server, and do not recalculate the EPC, it will contain invalid dimension IDs, for example, a variant corresponding to the values would be shown on the GUI, but it would not correspond to any real variant in the database. In order to rectify the situation, you can runppmimport with **–keyindicator new**. (Using the command line parameter **–ps** to specify a suitable query, you can restrict the recalculation to the EPCs of the process type.)

Detailed information on how to use runppmimport can be found in the PPM Operation Guide.

### 9.3.9 Shared function dimension

By default, shared functions are transferred to the system through a one-time import of a shared fragment. Applying the shared fragment rules merges the shared functions that are contained in the imported shared fragments with the normal process instance fragments. All objects of the shared fragment are copied to the fragment instance. This automatically ensures the uniqueness of a shared function and you can use the function ID of a shared function to differentiate between shared functions.

If you directly import shared functions as normal fragment instances using event format, a unique ID is created for each imported function. The function ID cannot be used as characteristic of a shared function. You can, however, define a shared function dimension where identical dimension values combine functions into shared functions. Dimension values not specified are not included.

A shared function dimension has the following properties:

- Only one shared function can be defined per client.
- The shared function dimension must be registered at the process tree root.
- The shared function dimension is invisible on the interface.
- The shared function dimension cannot be customized using CTK.

**Example**

The following example supplies excerpts from measure and process tree configuration files. For functions with an **AT_IS_SHARED_FUNCTION** attribute having the value **true**, the calculation rule of the **AT_SHARED_FUNCTION_ID** function attribute used for the shared function dimension concatenates the internal function name with the time stamp of the function execution.

**keyindicator.xml file extract**

```
...
```

```
<calcattr name="AT_SHARED_FUNCTION_ID" type="OT_FUNC" delete="yes">
  <calculation>
    <if>
      <and mode="PPM4">
        <exists mode="PPM4">
          <filteredattribute name="AT_IS_SHARED_FUNCTION"↵
            nodetype="OT_FUNC" objectname="this"↵
            onerror="CONTINUE" filter="EARLY" />
        </exists>
        <eq mode="PPM4">
          <filteredattribute name="AT_IS_SHARED_FUNCTION"↵
            nodetype="OT_FUNC" objectname="this"↵
            onerror="CONTINUE" filter="EARLY" />
          <constant>
            <dataitem value="TRUE">
              TRUE
              <datatype↵
                name="BOOLEAN">Logical value</datatype>
            </dataitem>
          </constant>
        </eq>
      </and>
      <then>
        <concat mode="PPM4">
          <set mode="PPM4">
            <filteredattribute name="AT_OBJNAME_INTERN"↵
              nodetype="OT_FUNC" objectname="this"↵
              onerror="EXIT_WARNING" filter="EARLY" />
            <convert datatype="TEXT">
              <filteredattribute name="AT_END_TIME"↵
                nodetype="OT_FUNC" objectname="this"↵
                onerror="EXIT_WARNING" filter="EARLY" />
            </convert>
          </set>
        </concat>
      </then>
    </if>
  </calculation>
</calcattr>
...
<sharedfunctiondim name="SHARED_FUNCTION">
  <description name="Shared Function" language="de" />
  <description name="Shared Function" language="en" />
  <dimitem attrname="AT_SHARED_FUNCTION_ID"↵
    colname="SHARED_FUNCTION" calculated="TRUE">
    <description language="de" name="SHARED_FUNCTION" />
    <description language="en" name="SHARED_FUNCTION" />
  </dimitem>
</sharedfunctiondim>
...
```

**processtree.xml file extract**

```
...
<usesfdim name="SHARED_FUNCTTION" />
<processtypegroup name="Standard order"↵
  dbtablename="CUBE1">
  <processtype name="Order processing" autovisible="FALSE" />
</processtypegroup>
...
```

### 9.3.10 Using organizational units as dimensions

Using an organizational unit as a dimension does not represent a separate dimension type. Specifying a special calculation rule copies the name of the organizational unit to the relevant functions of the process instance as an attribute.

**Example**

The following file extracts from the measure configuration illustrate the creation of a dimension from organizational units:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE keyindicatorconfig SYSTEM
                      'KeyindicatorConfiguration.dtd'>
<keyindicatorconfig>
  ...
  <calcattr name="AT_ORGUNIT" type="OT_FUNC">
    <calcclass name= "com.idsscheer.ppm.server.
                  keyindicator.attributecalculator
                   .ZAttributeCalculatorOriginator"/>
  </calcattr>
  ...
  <oneleveldim name="ORGUNIT" dimtype="FUNCTION"
                                    internal="no">
    <description language="de" name="Processor"/>
    <leveldesc>
      <dimitem attrname="AT_ORGUNIT" colname="FIRST_ID"
                                calculated="TRUE">
        <description language="de" name="Processor"/>
      </dimitem>
    </leveldesc>
  </oneleveldim>
  ...
</keyindicatorconfig>
```

The **AT_ORGUNIT** attribute is created for each function instance and is assigned the name of the organizational unit as its value. This attribute is used to create the one-level function dimension **ORGUNIT**.

The **AT_ORGUNIT** attribute is one of the default attributes in the PPM system and does not need to be defined.

## 9.4 Definition of data access dimensions

By configuring data access dimensions, you can assign data access privileges that, in addition to process access privileges, enable you to control access to PPM data.

Data access privileges are assigned to user groups and are inherited by the users assigned to that group. The administrator (PPM user with **User management** function privilege) defines the data access privileges by specifying particular filters on dimensions that cannot be edited by the user. These dimensions are called data access dimensions and specified in the configuration of the process tree through the **roledim** XML element. The **roledim** element must reference an already

configured text dimension (chapter Text dimensions (page 162)) that must be registered at the root of the process tree. This ensures that data access dimensions can be used throughout the entire process tree. Only one- and two-level text dimensions are allowed for the **roledim** element.

If you do not want a data access dimension to be displayed in the PPM user interface, specify the **internal="yes"** XML attribute in the definition of the dimension.

**Example**

In the process tree configuration file, the **Sold-to party** and **Sales organization**data access dimensions are specified as follows:

```
...
<roledim name="VKORG"/>
<roledim name="PRINCIPAL" refinement="BY_LEVEL1"/>
...
<usedim name="VKORG"/>
<usedim name="PRINCIPAL" refinement="BY_LEVEL1"
                         scale="LEVEL1SCALE"/>
...
```

The two dimensions are available as data access dimensions in privilege management.

PPM users inherit the data access privileges for all user groups they are assigned to. The data access privileges are linked as follows:

- Different data access privileges for the same dimensions are linked by an OR rule.
- Data access privileges for different dimensions are linked by an AND rule. If the user is assigned to a user group that has the data access privilege None, None is ignored.
- If a user belongs to at least one user group that has the data access privilege All, this data access privilege is not restricted by the data access privileges of other groups to which the user belongs.

A user who is not assigned to any user groups has no data access privileges.

## SPECIAL CASE

To link data access privileges for different dimensions with an OR rule, combine the values of these dimensions into a new, invisible dimension using the attribute calculator and specify the calculated dimension as the data access dimension.

**Example**

You want to assign data access privileges for the two dimensions **Location 1** and **Location 2**in such a way that a user can view data if the **Munich** plant appears in one of the **Location 1** or **Location 2** dimensions.
All dimension values for the two dimensions are combined in the calculated dimension **Location 3**. This is specified as a data access dimension in the process tree configuration.

| Location 1 | Location 2 | Location 3 |
|---|---|---|
| Munich | Berlin | Munich_Berlin |
| Stuttgart | Leipzig | Stuttgart_Leipzig |

| Location 1 | Location 2 | Location 3 |
|---|---|---|
| Hamburg | Munich | Hamburg_Munich |
| Saarbrücken | Hamburg | Saarbrücken_Hamburg |

Using the filter expression ***Munich*** creates the relevant data access privilege.

## 9.4.1 Using data access dimensions

A PPM user who logs in using restricting data access privileges can only view data that is released for him. From an overall system perspective, the use of data access dimensions has the following effect:

### PROCESS ACCESS PRIVILEGES

Process access privileges specified for a user are evaluated independent on data access privileges. Within the process types released for him, a user can only analyze the data corresponding to his data access privileges.

### DATA ANALYSIS

Every analysis inquiry is automatically supplemented by the filter for the access dimensions applicable for the user logged in. The filters for multiple data access dimensions are linked by an AND rule.

### FILTER DIALOGS

If the data access privileges for a user are restricted, in the filter dialog for the corresponding data access dimension, only the dimension values released for that user are displayed for selection.

### PLANNED VALUES

A user can only create planned values for data for which he has data access privileges. As with the data analysis, the filter for the access dimensions applicable to the user logged in is automatically added to the planned value definition.
If planned values are defined for which the data access dimension filter, valid for the user logged in, is only part of the filter valid for the planned value, the planned value is displayed but cannot be edited by the user.

### AGGREGATION AND DELETING

A user can only aggregate and delete the data for which he has data access privileges. As with the data analysis, the filter for the access dimensions applicable to the user logged in is automatically used.

For persistent aggregation, iteration automatically uses the data access dimension so that the assignment of the aggregated process instances to the data access dimensions is retained.

### DATA IMPORT

Data can be imported with no restrictions. If a user has the data import privilege, he can also import data for which he will not actually have access privileges after completing the import operation.

### PROCESS INSTANCE-INDEPENDENT DATA

The behavior of data access privileges applies to process instance-independent measures and dimensions with no restrictions. In this case, you can use any dimensions of the imported process instance-independent data as data access dimensions.

## 9.5 Process tree definition

The definition of the process tree is specified in an XML file. This file has the following structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE processtree SYSTEM
                     "KeyindicatorProcesstree.dtd">
<processtree name="...">
<!-- ROOT - Definition -->
    <processparamset>
        <paramset>
        ...
        </paramset>
    </processparamset>
    <functionparamset>
        <paramset>
        ...
        </paramset>
    </functionparamset>
<!-- Standard measures -->
    <useki ...  >
      <usepidim ... />
    </useki>
<!-- Default dimensions -->
    <usedim ... "/>
<!-- Start of process tree definition -->
    <processtypegroup name="Auftragsabwicklung">
        <processparamset>
        ...
        </processparamset>
        <functionparamset>
        ...
        </functionparamset>
        <useki ... >
          <usepidim ... />
        </useki>
        <usedim ... "/>
        <processtype name="Other orders">
        </processtype>
        ...
    </processtypegroup>
    <processtypegroup name="...">
        <processparamset>
        ...
        </processparamset>
        <useki ... />
```

```
        <usedim ... "/>
        <processtype name="..." autovisible="...">
        </processtype>
        ...
    </processtypegroup>
    ...
</processtree>
```

| XML element | Description |
|---|---|
| **processtree** | Name of process tree. Displayed as the root. The definition of the process tree contains the following details:<br>- Default query for processes and functions<br>- Measures and dimensions available in the entire tree<br>- At least one process type group |
| **processparamset** | Specifies one default query each (default paramset) for the root of the process tree, each process type group and each process type. The default query is shown when starting the analysis component of the PPM front-end. The default query can be called up at any time using the **Display default query** pop-up menu in the process tree. |
| **functionparamset** | Specifies a default query (default paramset) for each function type. |
| **paramset** | Describes the presentation of the default queries as an analysis in XML notation. |
| **useki** | Assigns the specified measure to the relevant elements of the process tree (**processtree**, **processtypegroup**, **processtype** XML elements). |
| **usedim** | Assigns the specified dimension to the relevant elements of the process tree (**processtree**, **processtypegroup**, **processtype** XML elements). |
| **userelki** | Assigns the specified relation measure to the relevant elements of the process tree (**processtree**, **processtypegroup**, **processtype** XML elements). |

| XML element | Description |
|---|---|
| **usereldim** | Assigns the specified relation dimension to the relevant elements of the process tree (**processtree**, **processtypegroup**, **processtype** XML elements). |
| **usepidim (optional)** | Registers the specified dimension to the same process tree element to which the process instance-independent measure specified by **useki** is assigned. This is only necessary if the dimension for the process tree element is not already available due to being assigned or passed on. |
| **processtypegroup** | Defines a process type group. The definition of a process type group contains the following information:<br>- Name of process type group<br>- Default query for processes and functions in the process type group<br>- Measures and dimensions assigned to the process type group in addition to the global measures and dimensions<br>- At least one process type |
| **processtype** | Defines a process type. The definition of a process type contains the following details:<br>- Name of the process type<br>- Default query for processes and functions in the process type<br>- Measures and dimensions that are assigned to the process type in addition to those from the process type group<br>The optional **autovisible="TRUE"** gives newly created PPM users automatic access privileges for this process type. The default setting is **FALSE**, that is, newly created PPM users initially have no access privileges for this process type. |

The names of process types and process type groups in the process tree must be unique and correspond to the names used in the process type definition. They are therefore specified in only one language, that is, the language of the source system.

## 9.5.1 Registration of measures and dimensions at the PPM system

The measures and dimensions defined in the client-specific measure configuration (**KeyindicatorConfiguration.xml**) must be registered in the process tree configuration (**ProcessTree.xml**) to be known to the PPM system. Then they are available after a successful configuration import for analyses and calculations in the PPM interface.

Measures are assigned to individual function types, individual process types, individual process type groups, or all process type groups (global measures and dimensions).

Measures and dimensions for a process type group are automatically passed on to subordinate process types. If the process tree root is selected, only the measures and dimensions that are assigned to all process type groups are displayed.

### 9.5.1.1 Register measure

A measure is registered at the PPM system in the process tree configuration file using the following XML element:

```
<useki name="..." assessment="..." scale="..." refinement="..."/>
```

| XML tag | Description |
|---|---|
| **name** | Internal name of the measure. The measure itself is defined in the **kidef** XML element in the measure configuration. |
| **assessment** | Assessment of a measure. **POS** specifies that high measure values are assessed positively. **NEG** specifies that low measure values are assessed positively. |
| **scale** (optional) | Default scaling of the measure. A unit of the attribute data type on which the measure is based (for example, unit **HOUR** when using the **TIMESPAN** data type). The scaling can be changed in the analysis.<br>If no scaling is set, PPM automatically determines a value for optimum representation. |
| **refinement** (optional) | Default step size for the measure when used as an iteration. The value must be specified with a unit (for example, 2.5 **PER_DAY**). The step size can be changed using the measure pop-up menu in the analysis. |

### 9.5.1.1.1 Register relation measure

A relation measure is registered at the PPM system in the process tree configuration file using the following XML element:

```
<userelki name="..." relname="..." assessment="..." scale="..." refinement="..."/>
```

If necessary, the same relation measure is registered individually for each relation.

| XML tag | Description |
|---|---|
| **name** | Internal name of the relation measure. The measure itself is defined in the **kidef** XML element in the measure configuration. |
| **relname** | Internal name of the relation for which the relation measure is to be available |
| **assessment** | Chapter **Register measure** (page 194) |
| **scale** (optional) | Chapter **Register measure** (page 194) |
| **refinement** (optional) | Chapter **Register measure** (page 194) |

**Example (extract from process tree configuration):**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE processtree SYSTEM
                     "KeyindicatorProcesstree.dtd">
<processtree name="Processes">
  <processparamset>
  ...
  </processparamset>
  <functionparamset>
  ...
  </functionparamset>
  ...
  <processtypegroup name="Customer Services"
                              dbtablename="CUBE6">
  <processparamset>
  ...
  </processparamset>
  <userelki name="RNUMA" relname="REL_CARRY_OUT"
                                assessment="POS"/>
  <userelki name="RNUMA" relname="REL_PING_PONG"
                                assessment="POS"/>
  ...
  <processtype name="..." autovisible="TRUE">
  <typifierrule function="..." priority="..."/>
  ...
  <userelki name="..." relname="..."/>
  ...
  </processtype>
  ...
  </processtypegroup>
  ...
</processtree>
```

### 9.5.1.1.2 Register measures and dimensions of process instance-independent data series

Process instance-independent measures and referenced dimensions of process instance-independent data series must be registered at the process tree before they can be used in analyses.

Like process instance-dependent measures, process instance-independent measures are registered at the process tree by the **useki** XML element.
A process instance-independent measure can be registered at multiple process type groups or process types.

**Example**

The process instance-independent measure **OVERHEAD_COSTS** is registered to the **Shipping** process type group in the process tree. The measure base unit is **EUR**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE processtree SYSTEM
                    "KeyindicatorProcesstree.dtd">
<processtree name="Processes">
  ...
  <processtypegroup name="Shipping">
    ...
    <useki name="OVERHEAD_COSTS" scale="EUR" assessment="NEG"/>
    ...
  </processtypegroup>
  ...
</processtree>
```

#### 9.5.1.1.2.1 Special case: Register referenced dimensions

If a process instance-independent measure relates to dimensions that are not available at the same process tree element as the process instance-independent measure, the dimensions must be registered using the **usepidim** XML element within the **useki** element at the process tree element of the process instance-independent measure.

**Example (previous example continued):**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE processtree SYSTEM
             "KeyindicatorProcesstree.dtd">
<processtree name="Processes">
  ...
  <processtypegroup name="Shipping">
    ...
    <useki name="OVERHEAD_COSTS" scale="EUR" assessment="NEG">
      <usepidim name="PRINCIPAL"/>
    </useki>
    <usedim name="D_COLOR" />
    <usedim name="D_EQUIPMENT" />
    <usedim name="D_PRODUCT" />
    <processtype name="...">
      ...
    </processtype>
  </processtypegroup>
```

```
  ...
</processtree>
```

The dimension principal (**PRINCIPAL**) is registered together with the process instance-independent measure **OVERHEAD_COSTS** at the **Shipping** process type group, at which the dimension itself is not registered via the **usedim** element.
The dimension referenced by the process instance-independent data series and thus registered serves only analysis purposes pertaining to the process instance-independent measure **OVERHEAD_COSTS**.

Dimensions registered using the **usepidim** XML element cannot be used for measure calculation at process instance level.

### 9.5.1.2 Register dimension

A dimension is registered at the PPM system in the process tree configuration file using the following XML element:

```
<usedim name="..." scale="..." refinement="..." variance="..."/>
```

| XML tag | Description |
|---|---|
| **name** | Internal name of the dimension. The dimension itself is defined in one of the **oneleveldim**, **twoleveldim**, **floatingdim**, **timedim**, **hourdim** or **searchdim** XML elements in the measure configuration. |
| **scale** (optional) | Default scaling of the dimension. The scaling can be changed using the dimension pop-up menu in the analysis.<br>If no scaling is set, PPM automatically determines a value for optimum representation. |
| **refinement** (optional) | Default step size of the dimension. The possible options depend on the attribute data type on which the dimension is based:<br>- **Numerical attribute**: Value with unit<br>- **Alphanumeric attribute** (also data type specific):<br>- **Text**: No entries possible. The iteration steps are stipulated by the different attribute values.<br>- **Text pair BY_LEVEL1**, **BY_LEVEL2** (rough, detailed): For each dimension level, the iteration steps are stipulated by the attribute values.<br>- **Time**: **BY_YEAR**, **BY_QUARTER**, **BY_MONTH**, **BY_WEEK**, **BY_DAY**, **BY_HOUR**, **BY_MINUTE**<br>If no scaling is set, PPM automatically determines a value for optimum representation. |

| XML tag | Description |
|---|---|
| **Variance** (optional) | Has been defined for future expansions and is not currently in use. |

### 9.5.1.2.1 Register reference dimension

The reference dimensions used to define relations are registered at the process tree root so that they are available in all process type groups and process types. They are registered separately for each relation using the **usereldim** XML element.

| XML tag | Description |
|---|---|
| **name** | Internal name of the reference dimension. The dimension itself is defined in one of the **oneleveldim** or **twoleveldim** XML elements in the measure configuration. |
| **relname** | Internal name of the relation (see chapter on **Definition of relations** (page 202)) for which the reference dimension is to be available |
| **scale** (optional) | Chapter **Register dimension** (page 197) |
| **refinement** (optional) | Chapter **Register dimension** (page 197) |
| **Variance** (optional) | Chapter **Register dimension** (page 197) |

**Example (extract from process tree configuration):**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE processtree SYSTEM
                       "KeyindicatorProcesstree.dtd">
<processtree name="Processes">
  <processparamset>
  ...
  </processparamset>
  <functionparamset>
  ...
  </functionparamset>
  <roledim name="..."/>
  ...
  <useki name="..." assessment="..."/>
  ...
  <usedim name="..." refinement="..." scale="..."/>
  ...
  <usereldim name="FROMORG" relname="REL_CARRY_OUT"/>
  <usereldim name="FUNCTION" relname="REL_CARRY_OUT"/>
  <usereldim name="FROMORG"
                     relname="REL_WORKS_TOGETHER"/>
  <usereldim name="TOORG" relname="REL_WORKS_TOGETHER"/>
  <usereldim name="FROMORG" relname="REL_PING_PONG"/>
```

```
  <usereldim name="TOORG" relname="REL_PING_PONG"/>
  ...
</processtree>
```

#### 9.5.1.2.2 Register relation dimension

Relation dimensions are registered in the same way as reference dimensions (see chapter **Register reference dimension** (page 198)) except that relation dimensions can be registered at different process tree elements (process tree root, process type groups, or process types). For the registration at superordinate process tree elements, the same inheritance mechanisms apply as described in the introduction to the process tree configuration (see chapter Registration of measures and dimensions at the PPM system (page 194)).

If necessary, the same relation dimension is registered individually at the corresponding process tree element for each relation for which it is to be available.

**Example (extract from process tree configuration):**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE processtree SYSTEM
                       "KeyindicatorProcesstree.dtd">
<processtree name="Processes">
  <processparamset>
  ...
  </processparamset>
  <functionparamset>
  ...
  </functionparamset>
  ...
  <processtypegroup name="Customer Services"
                              dbtablename="CUBE6">
  <processparamset>
  ...
  </processparamset>
  <usereldim name="SOURCEFUNC"
                      relname ="REL_WORKS_TOGETHER"/>
  <usereldim name="TARGETFUNC"
                      relname ="REL_WORKS_TOGETHER"/>
  ...
  <processtype name="..." autovisible="TRUE">
  <typifierrule function="..." priority="..."/>
  ...
  <usereldim name="..." relname="..."/>
  ...
  </processtype>
  ...
  </processtypegroup>
  ...
</processtree>
```

### 9.5.2 Automatic process tree expansion

If process instances whose process type and process type group do not exist in the process tree are edited during data import, the process tree is automatically expanded to include the missing elements. A distinction is made between the following cases:

- New process type and new process type group
  The new process type group and the new process type are created under the root of the process tree. They inherit all measures and dimensions from the tree root.
- New process type in existing process type group
  The new process type is created under the existing process type group. The process type inherits all measures and dimensions from the process type group.

**Warning**

An individual process type cannot be assigned to multiple process type groups. In such a case, the measures and dimensions assigned to this process type in the new group would not be defined. This can occur if you transfer process typification information directly from the source system to the **AT_PROCTYPEGROUP** and **AT_PROCTYPE** process instance attributes.

Automatically created process type groups and process types inherit the measures and dimensions assigned to the higher-level elements. They cannot be assigned any further measures and dimensions.

## 9.5.3 Manual process tree expansion

During operation of your PPM system, the continuous importing of data can result in automatic expansion of the existing process tree (see chapter **Automatic process tree expansion** (page 199)).

If you want to expand the process tree configuration manually, you should first back up the current process tree with the automatically created expansions in a configuration file.

To do this, export the current process tree to a local XML file using **runppmconfig** (see **PPM Operation Guide**).

Specify the relevant expansions in the XML file exported.

Then import the file with the expansions back into the PPM system using **runppmconfig**.

## 9.5.4 Definition of process tree in multi-byte character sets

The following extract from the process tree configuration file shows an example of the definition options for process type groups or process types when using a multi-byte character set:

### DEFINITION OF PROCESS TYPE GROUP AND PROCESS TYPE WITH DEFAULT QUERY:

```
...
  <processtypegroup name="διεκπεραίωση εντολής"
                              dbtablename="CUBE10">
    <processparamset>
      ...
    </processparamset>
    <functionparamset>
      ...
    </functionparamset>
    <processtype name="πώληση τοις μετρητοίς"
                              autovisible="FALSE">
      <processparamset>
```

```
      <paramset>
        ...
        <kiquery showzero="auto">
          <keyindicator>
            <criterion name="PNUM">   Number of processes
</criterion>
          </keyindicator>
          <iteration nullvalue="no">
            <criterion name="PROCESSTYPE">Process type
            </criterion>
              <refinement name="BY_LEVEL1">Rough
              </refinement>
          </iteration>
          <filter>
            <criterion name="PROCESSTYPE">Process type
            </criterion>
              <filteritem operator="or">
                <dataitem>
                 διεκπεραίωση εντολής\πώληση
                 τοις μετρητοίς
                  <datatype name="TEXTPAIR">
                    Text pair
                  </datatype>
                  <scale name="LEVEL2SCALE"
                                  factor="2.0">
                    Detailed
                  </scale>
                </dataitem>
              </filteritem>
          </filter>
        </kiquery>
        ...
      </paramset>
    </processparamset>
    <functionparamset>
      <paramset>
        ...
        <kiquery showzero="auto">
          ...
          <filter>
            <criterion name="PROCESSTYPE">Process type
            </criterion>
            <filteritem operator="or">
              <dataitem>
                  διεκπεραίωση εντολής\πώληση
                 τοις μετρητοίς
                <datatype name="TEXTPAIR">Text pair
                </datatype>
                  <scale name="LEVEL2SCALE"
                                  factor="2.0">
                    Detailed
                  </scale>
                      </dataitem>
            </filteritem>
          </filter>
          ...
        </kiquery>
      </paramset>
    </functionparamset>
  </processtype>
</processtypegroup>
```

...

In the example, the process type group **διεκπεραίωση εντολής** is defined and assigned the process type **πώληση τοις μετρητοίς**.

## 9.6 Relations

In the **Interaction analysis** module, relations between different objects can be analyzed at process instance level.

A relation is a link between two object instances in a process instance.

Organizational units and functions can be used as the reference objects. The calculator (see chapter on **Definition of relation calculations** (page 204)) of a relation determines the object instances between which the relation exists.

These relations (relation occurrences) can be thought of as an invisible connection between the object instances in the process instance. Specific measures and dimensions can be defined for each relation (see **Definition of relation measures** (page 209) and **Definition of relation and organizational dimensions** (page 211) chapters) and will then be calculated for each relation occurrence at process instance level.

Example configurations relating to **Interaction analysis** are located in your PPM Customizing Toolkit installation in the directory **<PPM installation directory\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ctk\examples\custom\organalysis**.

### 9.6.1 Definition of relations

Relations are the basis for the definition of relation measures and relation dimensions. They are defined in the measure configuration.

A relation exists between a source reference dimension and a target reference dimension (see **Reference dimensions** (page 203) chapter) and has a name and a relation calculator assigned using the **calcrel** XML element.

| XML tag | Description |
|---|---|
| **relation** | Relation definition |
| **name** | Unique key word for the relation for internal referencing |
| **id** | Unique integer between 0 and 999 under which the corresponding database table is created |
| **description** | Language-specific user interface name. This must be specified in the default language. |
| **sourcedim** | Source reference dimension. Only one or two-level dimensions of **OT_FUNC** or **OT_ORG** type (see chapter on **Reference dimensions** (page 203)) |

| XML tag | Description |
|---|---|
| **targetdim** | Target reference dimension. Only one or two-level dimensions of **OT_FUNC** or **OT_ORG** type (see chapter on **Reference dimensions** (page 203)) |
| **refki** | At least one measure of the **RELATION** type |
| **refdim (optional)** | Dimension of **RELATION** or **PROCESS** type.<br>Each referenced dimension can only be evaluated in the context of the specified relation. |

**Example**

(extract from **Keyindicator.xml**)

```
...
<relation name="REL_CARRY_OUT" id="0">
  <description name="executes" language="de" />
  <sourcedim name="FROMORG" />
  <targetdim name="FUNCTION" />
  <refki name="REL_CO_CORATE" />
  <refki name="RNUMA" />
  <refki name="REL_CO_DLZ" />
  <refki name="ORGCAPA" />
  <refki name="REL_CO_COST" />
  <refdim name="REL_CO_TIME" />
</relation>
...
```

**Warning**

Do not use the same dimension as the source and target reference dimension of a relation. This leads to the measure configuration import being aborted with a corresponding error message.

You cannot reference any cardinality measures in the **refki** XML element.

### 9.6.1.1 Reference dimensions

In PPM a relation always exists between a source object and a target object, known as reference objects. To specify the reference objects of a relation, you must define reference dimensions (**sourcedim**, **targetdim**) for these objects in the measure configuration. For each relation, you define a source reference dimension and a target reference dimension of the **TEXT** type (one, two, or n-level dimensions). The individual dimension values are used to reference particular organizational units or functions. The required attribute mapping is carried out in the configuration file for the organizational units and/or in the mapping information file.

The **dimtype** XML attribute is used to specify the object type for which the reference dimension is defined. Valid values are **OT_FUNC** for function dimensions and **OT_ORG** for organizational dimensions.

**Example**

(extract from **Keyindicator.xml**)

```
...
```

```
<twoleveldim name="FROMORG" dimtype="OT_ORG"
                  dimgroup="DIM_GROUP_CRITERIA"
                  internal="no" importmode="OPTIONAL">
  <description language="de"
                 name="Organisationseinheit (Start)"/>
  <leveldesc>
    <dimitem attrname="AT_ORGGRP" colname="GRP"
                               calculated="FALSE">
      <description language="de" name="Group" />
    </dimitem>
  </leveldesc>
  <leveldesc>
    <dimitem attrname="AT_OBJNAME"
                  colname="NAME" calculated="FALSE">
      <description language="de" name="Name" />
    </dimitem>
  </leveldesc>
</twoleveldim>
...
```

**Warning**

You need to register the source and target reference dimensions used at the process tree root in the process tree configuration (see chapter **Register reference dimension** (page 198)). Otherwise, an error message is output when importing the process tree configuration.

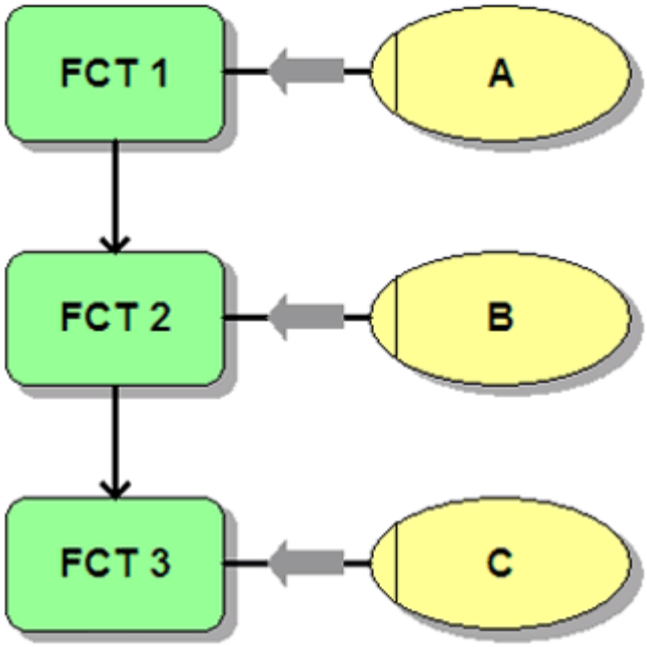
## 9.6.2 Definition of relation calculations

The defined relations are created using relation calculators in the process instance. For each relation, the corresponding calculation class is specified in the **calcrel** XML element in the measure configuration. By default, these are the following four classes (the fixed part of the class name is omitted in each case):

- ZRelationCalculatorCarriesOut for the **executes** relation

In the corresponding process instances, that is, those for whose process type or process type group the relevant relation measures or relation dimensions are registered, this creates a relation from each instance of an organizational unit to the associated function instance, which is assigned to the organizational unit by the **executes** connection (**CXN_UNDIRECTED**).

**Example**

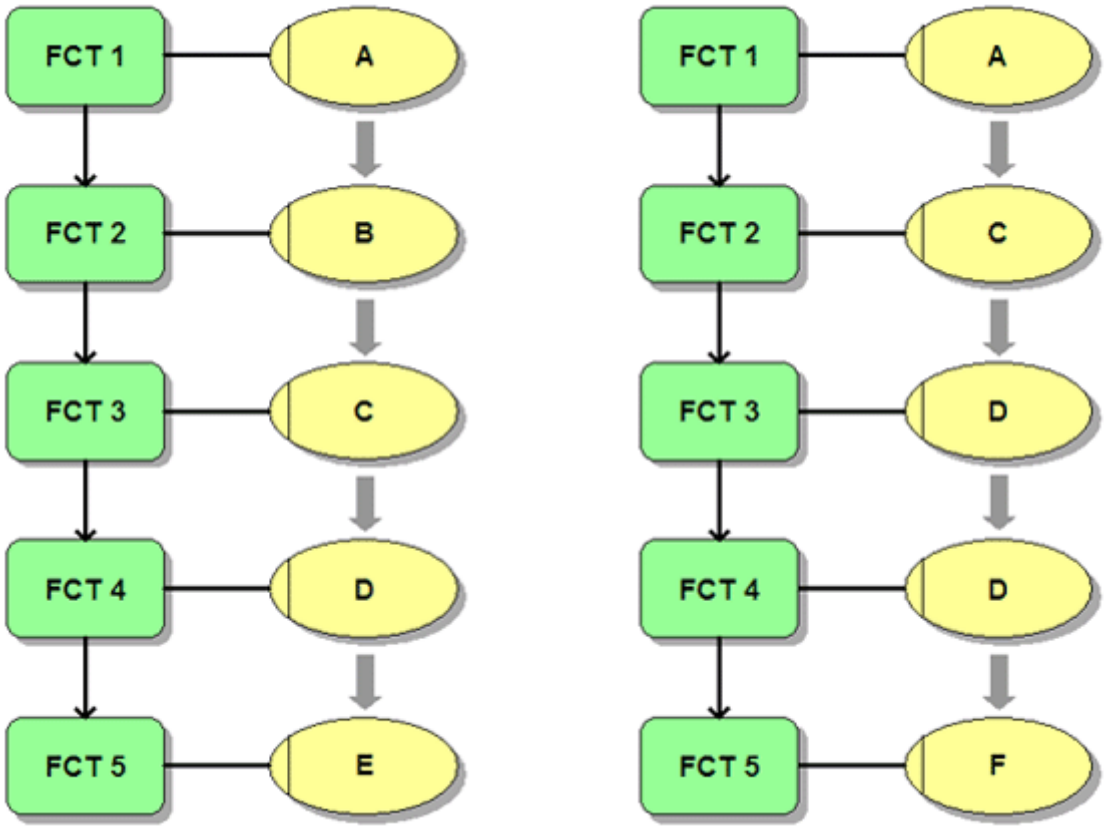



In the process instance, the relation calculator creates the **executes** relation (gray arrows) between each organizational unit and function. The organizational unit is the source object and the function the target object.

- ZRelationCalculatorWorksTogether for the **co-operates with with (without gaps)** relation

In the corresponding process instances, this creates a relation from the instance of an organizational unit to each instance of the organizational unit that executes the directly succeeding function instance. These can be identical organizational units, that is, organizational units with the same name (**AT_OBJNAME**). By selecting appropriate filters, this relation can be used to evaluate organizational structures within an organizational unit.

**Example**

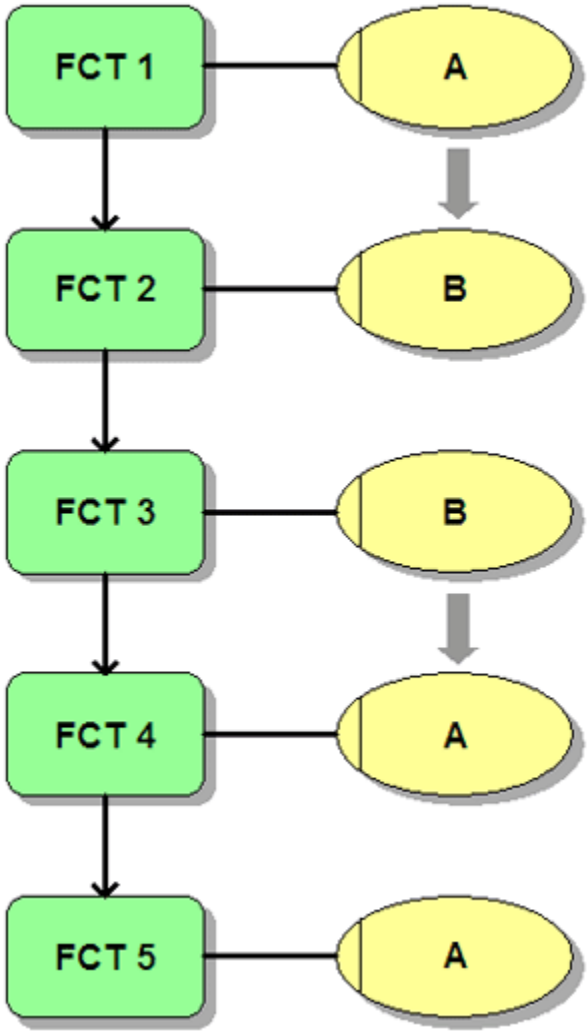



In both process instances, the relation calculator creates the **cooperates with** relation (gray arrows) between each organizational unit and the organizational unit that executes the directly succeeding function. For example, the graphic shows that organizational unit **D** cooperates with organizational units **D**, **E**, and **F**.

- ZRelationCalculatorWorksTogetherLongDistance for the relation **cooperates with (with gaps)**; the function instances including the organizational units do not have to be sequential, unlike the previously described relation **cooperates with (without gaps)**.
- ZRelationCalculatorOrgBreak for the **Organizational break** relation

Behaves in exactly the same way as the co-operates with relation except that the relation is only created between different organizational units, that is, organizational units with different names (**AT_OBJNAME**). This relation is used to evaluate organizational structures between different organizational units.

**Example**

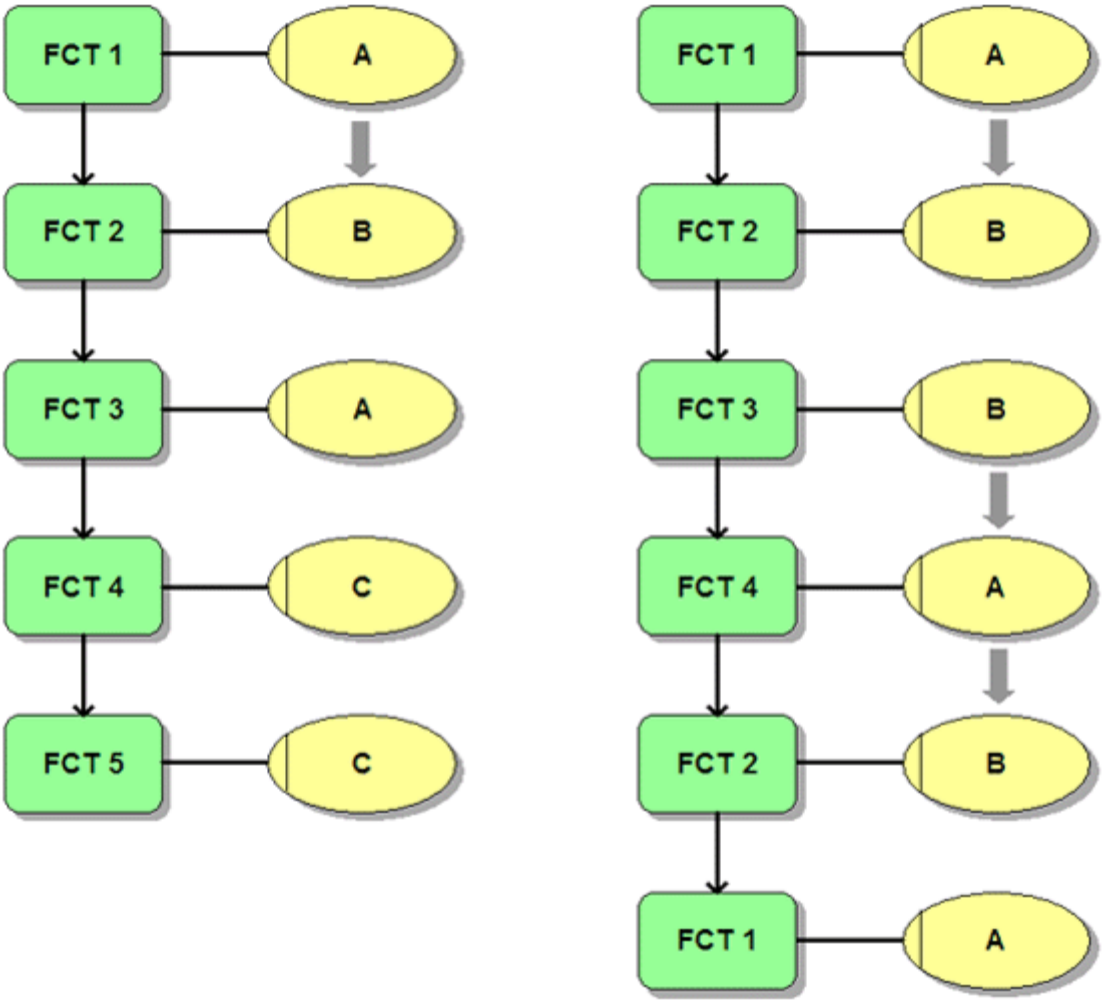



In the process instance, the relation calculator creates the **Organizational break** relation (gray arrows) between two organizational units, whenever execution of the directly succeeding function results in an organizational change.

- ZRelationCalculatorPingPong for the **Ping pong** relation

In the corresponding process instances, this creates a relation between two organizational units with different names, which switch directly at least once in the subsequent process flow without any additional organizational units being involved. This can involve the execution of different functions or the same function (**AT_OBJNAME**).

### Example




In the two process instances, the relation calculator creates the **Ping pong** relation (gray arrows) between organizational unit **A** and organizational unit **B**.

### Example (definition of a relation calculator)

(extract from **Keyindicator.xml**)

```
...
<calcrel name="REL_CARRY_OUT">
  <calcclass name="com.idsscheer.ppm.server.keyindicator.
     relation.calculator.ZRelationCalculatorCarriesOut"/>
  <calcparam key="..." value="..."/>
</calcrel>
...
```

| XML tag | Description |
|---|---|
| **calcrel** | Relation calculator |
| **name** | Internal name of the relation to be calculated |

| XML tag | Description |
|---|---|
| **calcclass** | Name of the calculation class. Any optional **calcparam** XML elements transfer calculation parameters when the class is called up (see chapter on **Definition of attribute calculations** (page 45)). |
| **depends** (optional) | Name and type of an attribute (**PROCESS**, **OT_FUNC**, **OT_EVT**, **OT_ORG**, or **RELATION**), which must exist for the calculation to be executed. If the specified attribute is a calculated attribute, this is calculated first. The **relname** attribute specifies the relation on which there is a dependency (only for **type="RELATION"**).<br>Several **depends** elements can be specified simultaneously.<br>Not to be used in conjunction with **dependsrel**. |
| **dependsrel** (optional) | Name of the relation of which there is a dependency. Several **dependsrel** elements can be specified simultaneously. Not to be used in conjunction with **depends**. |

## 9.6.3 Definition of relation measures

Relation measures are assigned to a particular relation in the measure configuration using the **refki** XML element. Relation measures can only be evaluated in the **Interaction analysis** module with the corresponding relation. Relation measures are indicated by yellow symbols in the user interface.

Relation measures are configured using the following XML elements and XML attributes in the measure configuration (see chapter on **Definition of standard measures** (page 136)):

| XML tag | Description |
|---|---|
| **name** | Unique key word for measure. Referenced in the **refki** XML element in the relation definition (see chapter on **Definition of relations** (page 202)). Recommended prefix: **REL_** |
| **type** | **RELATION** (relation measure) |

| XML tag | Description |
|---|---|
| **location** (optional) | Only for **type="RELATION"**<br>Valid values: **SOURCE** (the attribute from which the measure value is taken, search performed on the source reference object of the relation.)<br>**TARGET** (the attribute from which the measure value is taken, search performed on the target reference object of the relation.)<br>**THIS** (default value: The search for the attribute from which the measure value is taken is performed on the relation connection itself) |

### Example (extracts from measure configuration)

Measure definition:

```
...
<kidef name="REL_CO_DLZ" type="RELATION"
      attrname="AT_APX_PROCESSINGTIME" calculated="TRUE"
      location="TARGET" distribution="TRUE"
      standarddeviation="TRUE" sharedfunctionki="FALSE"
      functionspanki="FALSE" retrievertype="KEYINDICATOR"
              dimreferring="LOOSE" importmode="OPTIONAL">
  <description language="de" name="Average working time" />
</kidef>
...
```

Associated calculation rule:

```
...
<calcattr name="AT_APX_PROCESSINGTIME"
                          type="OT_FUNC" delete="no">
  <calculation>
    <max>
      <set>
        <constant>
          <dataitem value="0.0">
            0.000
            <datatype name="DOUBLE">Floating point number
            </datatype>
          </dataitem>
        </constant>
        <max>
          <attribute name="AT_APX_PROCESSINGTIME"
                 nodetype="OT_FUNC" onerror="CONTINUE" />
        </max>
      </set>
    </max>
  </calculation>
</calcattr>
...
```

## 9.6.4 Definition of relation and organizational dimensions

All dimension types (except search dimensions) can be defined as relation or organizational dimensions using the **dimtype** XML attribute (see chapter on **Definition of dimensions** (page 159)). These dimension types are only available for evaluations in the **Interaction analysis** module. These dimensions are indicated by yellow symbols in the user interface. For relation dimensions, the **location** attribute also specifies which object of the relation is searched for the corresponding dimension values (attributes). By default, this is the relation itself.

Relation and organizational dimensions are configured using the following XML elements and XML attributes in the measure configuration:

| XML tag | Description |
|---|---|
| **name** | Unique key word for dimension. Referenced in the **refdim** XML element in the relation definition (see **Definition of relations** (page 202) chapter). Recommended prefix: **REL_** |
| **dimtype** | **RELATION** (Relation dimension)<br>**OT_ORG** (Organizational dimension) |
| **location** (optional) | Only for **type="RELATION"**<br>Valid values: **SOURCE** (the attribute from which the dimension value is taken, search performed on the source reference object of the relation.)<br>**TARGET** (the attribute from which the dimension value is taken, search performed on the target reference object of the relation.)<br>**THIS** (default value: The search for the attribute from which the dimension value is taken is performed on the relation itself) |

#### Examples (extracts from measure configuration)

Time dimension as relation dimension:

```
...
<timedim name="REL_CO_TIME" dimtype="RELATION"
   attrname="AT_END_TIME" location="TARGET"
   tablename="FUNC_ENDTIME" precision="HOUR"
   dimgroup="DIM_GROUP_TIME" calculated="FALSE" internal="no"
   earlyalert="no" importmode="OPTIONAL">
  <description language="de" name="Time" />
</timedim>
...
```

Two-level dimension as organizational dimension:

```
...
<twoleveldim name="FROMORG" dimtype="OT_ORG"
           dimgroup="DIM_GROUP_CRITERIA" internal="no"
                              importmode="OPTIONAL">
  <description language="de"
                name="Organisationseinheit (Start)"/>
  <leveldesc>
    <dimitem attrname="AT_ORGGRP"
                     colname="GRP" calculated="FALSE">
      <description language="de" name="Group" />
    </dimitem>
  </leveldesc>
  <leveldesc>
    <dimitem attrname="AT_OBJNAME"
                    colname="NAME" calculated="FALSE">
      <description language="de" name="Name" />
    </dimitem>
  </leveldesc>
</twoleveldim>
...
```

**Warning**

Search dimensions (**searchdim** XML element) cannot be defined as either relation dimensions or organizational dimensions.

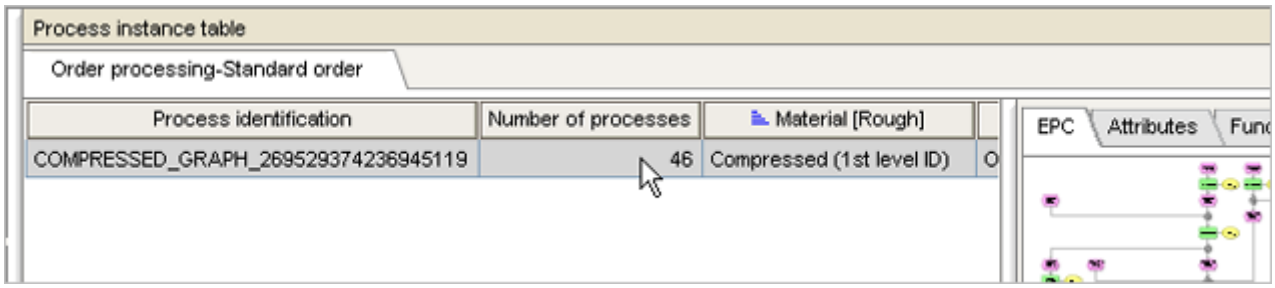
# 10 Change aggregation behavior

By default, when permanently aggregating process instances using the PPM command line aggregation (see **runppmcompress**) (see **PPM Operation Guide**), identical values of a process dimension are transferred as a dimension value to the aggregated EPC, while different values are deleted so that the dimension does not have a value in the aggregated EPC. You can then no longer trace whether dimension values had been specified or not. This only applies to dimensions that do not function as aggregation criteria, that is, dimensions that are not included as iterations in the aggregation paramset.

You can change the behavior of the command line aggregation through specific configuration settings in your PPM system so that both identical and different dimension values are deleted during aggregation if this dimension is not used for iteration in the aggregation paramset. In the aggregated process instance, the deleted values are then rendered visible by a uniform aggregation value. You specify aggregation values in the measure configuration for the relevant process dimension (text dimensions and search dimensions).

For time dimensions (**timedim** and **hourdim**), you can only specify that both identical and different dimension values are to be deleted during aggregation. In the analysis, the dimension can no longer be represented in the aggregated process instance due to the deleted values.

**Text dimension example (aggregation value for the Material process dimension)**




All values of the **Material** process dimension in the selected 46 process instances were deleted via command line during permanent aggregation and replaced by the aggregation value specified in the measure configuration (**Compressed (1st level ID)**). It does not matter whether the dimension values were identical or different.

## 10.1 Configure the internal aggregation attribute

Ensure that the internal aggregation attribute **AT_INTERNAL_COMPRESSCRITERION** is specified in the attribute configuration of your PPM system (see **Definition of attribute types and attribute type groups** (page 15)).

### ATTRIBUTENAMES.XML

```
...
  <attribute key="AT_INTERNAL_COMPRESSCRITERION"
                          name="Compression criteria"/>
...
```

### ATTRIBUTETYPES.XML

```
...
  <attributedefinition key="AT_INTERNAL_COMPRESSCRITERION"
                    type="TEXT" group="AG_KPI_COMPRESS"/>
...
```

In the aggregated process instance in the **AT_INTERNAL_COMPRESSCRITERION** process attribute, all aggregation criteria are listed, that is, all process dimensions included as iterations in the aggregation paramset used as well as all data access dimensions (see **Definition of data access dimensions** (page 188)). If set, the refinement level is specified after the internal name of the dimension in parentheses.

The **PROCESSTYPE** process dimension is always included in the internal aggregation attribute because it is part of each aggregation paramset. The **TIME** process dimension, however, is only included if it is contained as an iteration in the paramset. If only a time filter is contained in the aggregation paramset, the **Time** dimension (as other time dimensions of the **timedim** or **hourdim** type) is not listed as an aggregation criterion.

## 10.2 Assign aggregation values

Aggregation values displayed in an aggregated EPC to make deleted dimension values visible are specified in the measure configuration (see **Definition of dimensions** (page 159)).

### YOU CAN ASSIGN AGGREGATION VALUES TO THE FOLLOWING PROCESS DIMENSION TYPES:

- One-level, two-level, n-level dimensions
  (**oneleveldim** (page 167), **twoleveldim** (page 169), **nleveldim** (page 163))
- Search dimensions
  (**searchdim** (page 182))
- Time dimensions
  (**timedim** (page 174), **hourdim** (page 181))

#### Example 1 (aggregation values for a two-level dimension)

You want to specify aggregation values only for the first level of the two-level **Material** process dimension:

```
...
<twoleveldim name="MATERIAL" dimtype="PROCESS"
                        dimgroup="DIM_GROUP_CRITERIA">
   <description name="Material" language="en"/>
   <leveldesc>
      <dimitem attrname="AT_MATERIAL_KIND" colname="FIRST_ID"
                                  calculated="FALSE">
         <description language="en" name="Material type"/>
         <compressionvalue>
            Compressed (1st level ID)
</compressionvalue>
      </dimitem>
      <dimitem attrname="AT_MATERIALKIND_NAME"
                  colname="FIRST_DESC" calculated="FALSE">
         <description language="en" name="Material type name"/>
```

```
        <compressionvalue>
           Compressed (1st level description)
</compressionvalue>
      </dimitem>
   </leveldesc>
   <leveldesc>
      <dimitem attrname="AT_MATERIAL"
                colname="SECOND_ID" calculated="FALSE">
<description language="en" name="Material"/>
      </dimitem>
      <dimitem attrname="AT_MATERIAL_NAME"
              colname="SECOND_DESC" calculated="FALSE">
        <description language="en" name="Material type name"/>
      </dimitem>
   </leveldesc>
</twoleveldim>
...
```

For each key and description of the first level of the dimension, an aggregation value is specified with the **compressionvalue** XML element. For the permanent aggregation using command line aggregation, all identical and different values of the specified process dimensions are deleted. In the aggregated process instance, the specified aggregation values are displayed as the value of the dimension if the dimension is not included in the aggregation paramset. The default value for the aggregated second level is **Not specified** because no aggregation values are specified for the level items (**dimitem**) of the second level.

Process instance table

Order processing

| Process identification | Time ... | Nu... | Material [Detailed] |
|---|---|---|---|
| COMPRESSED_GRAPH_8263291222575049880 | Jan 07 | 34 | Compressed (1st level ID) (Compressed (1st level description))-Not maintained |

Always specify the aggregation values of the relevant level in pairs (for key and description), otherwise the import of the measure configuration will abort with an error message.

### Example 2 (Time dimension: Delete dimension values when aggregating)

You want to ensure that no values are displayed for the **Process end time** time dimension in permanently aggregated process instances, regardless of whether the dimension values of the process instances to be aggregated are identical or different.

```
...
<timedim name="PROZESSENDZEIT" dimtype="PROCESS" ...
                  attrname="AT_END_TIME" ...
                  calculated="TRUE" ...
                  deleteoncompression="TRUE" ... >
   <description name="Process end time" language="en" />
</timedim>
...
```

If you specify **deleteoncompression="TRUE"** for the **Process end time** process dimension, identical and different dimension values of the process instances to be aggregated are deleted during permanent aggregation via command line if the **Process end time** dimension is not included as an iteration in the aggregation paramset. Dimension values no longer exist in the aggregated EPC.

**Warning**

If you add the **Process end time** dimension in the PPM analysis of the displayed aggregated process instance, data can no longer be displayed due to the deleted dimension values.

You cannot specify aggregation values for the **Process type** (**PROCESSTYPE**) dimension because this dimension is automatically included as an iteration in each aggregation paramset. Similarly, you cannot specify for the **Time** (**TIME**) process dimension that dimension values are to be deleted during aggregation.

You can conveniently specify aggregation values for the relevant process dimension in PPM Customizing Toolkit in the **Dimensions** component of the **Measures and dimensions** module. This is also where you can specify for time dimensions of the **PROCESS** type whether dimension values are to be deleted during aggregation. A prerequisite is that the **AT_INTERNAL_COMPRESSCRITERION** attribute has been specified, that is, created.

# 11 System connections

## 11.1 SAP executables

If configured accordingly, you can use the pop-up menu to start an executable from a process instance selected in the process instance table via a login dialog in the SAP interface that displays data pertaining to the selected process instance.

### 11.1.1 Software requirements

The SAP logon must be installed on the same computer as the PPM front-end.

- The SAP Java Connector (JCo) must be installed on the client computer and the SAP server with the same version number.
- Notes on installation of the SAP Java Connector are available in the PPM Installation Guide.

### 11.1.2 Privileges in the SAP system

The PPM user calling the executable requires an SAP user ID with at least the following privileges:

- Login privilege via SAP GUI
- RFC privilege
- Privilege to execute the ABAP4_CALL_TRANSACTION remote function call
- Privileges to call the SAP executables specified in the configuration

### 11.1.3 Transaction call

If parameters transferred during the transaction call are incorrect, the corresponding error handling takes place in the SAP system itself (see SAP batch programming).

A transaction is called within an independent process. Therefore, several transactions can be open simultaneously.

### 11.1.4 Configuration

The SAP transactions are configured in a separate XML file that can be imported or exported using the **runppmconfig** command line program (see **PPM Operation Guide**). The language-specific descriptions (**description** XML elements) must be specified at least in the default language.

The XML configuration contains information on the pop-up menu, connection data for the available SAP systems and the transaction configurations (optional entries in italics):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE r3transactionconf SYSTEM
                            "mysaptransaction.dtd">
<r3transactionconf>
  <submenu>
    <description language="..." name="..."/>
```

```
  </submenu>
  <logoutMenuEntry>
    <description language="..." name="..."/>
  </logoutMenuEntry>
  <r3system systemid="..." client="...">
    <description language="..." name="...">
      Description of the SAP system
    </description>
    <locales>
      <defaultlocale value="..."/>
      <locale value="..."/>
    </locales>
    <applicationserver appserver="..."
                               systemnumber="..."/>
  </r3system>
  <transaction systemid="..." transactionid="..."
           transactioncode="..." skipfirstscreen="..."
           proctypegroup="..." mode="..." update="...">
    <description language="..." name="..."/>
    <batchinputline ... />
    ...
    <batchinputlist ... />
    ...
  </transaction>
  <transaction ...>
    ...
  </transaction>
  ...
</r3transactionconf>
```

For more detailed information on configuring system access, refer to the PPM Process Extractors Technical Reference.

For a transaction call to work, you need to configure not only connection data, but also at least one transaction (**transaction** XML element).

Use the SAP transaction recorder to create transaction configurations and record a corresponding transaction in the ABAP batch input. Please refer to the SAP documentation for more information on how the recorder works and on the ABAP batch input script syntax.

**Warning**

The following instructions on how to create configurations do not replace the SAP documentation, especially not in terms of resolving script errors. Basic knowledge about batch input scripts is a necessary requirement for creating transaction configurations.

In principle, two types of transaction calls can be configured:

- The call is only possible on one selected process instance (single select)
- The call is possible on one or multiple selected process instances (multi select)

### 11.1.4.1 Configuration examples

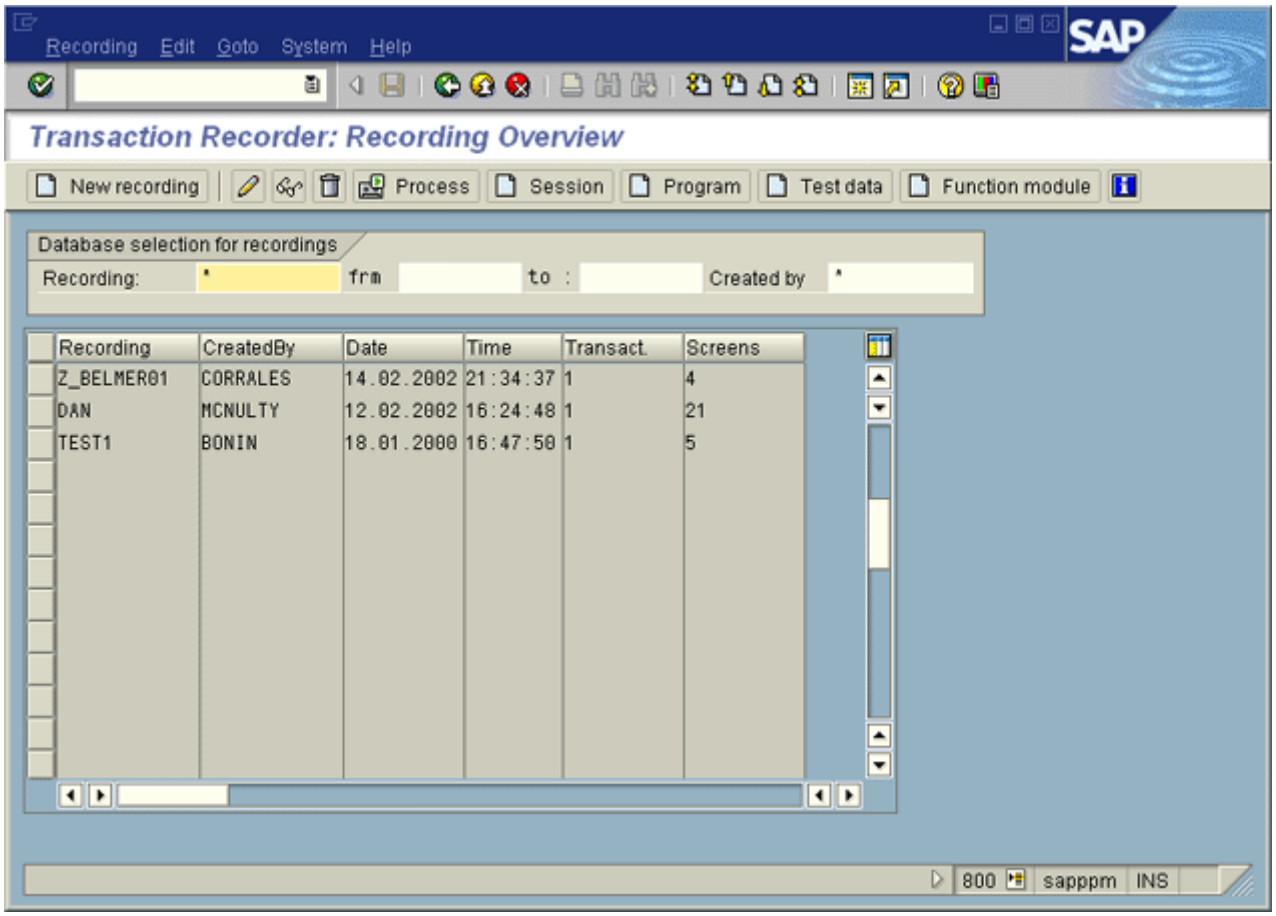
## RECORDING A VA03 SINGLE SELECT TRANSACTION IN THE SAP FRONT-END

### Requirements

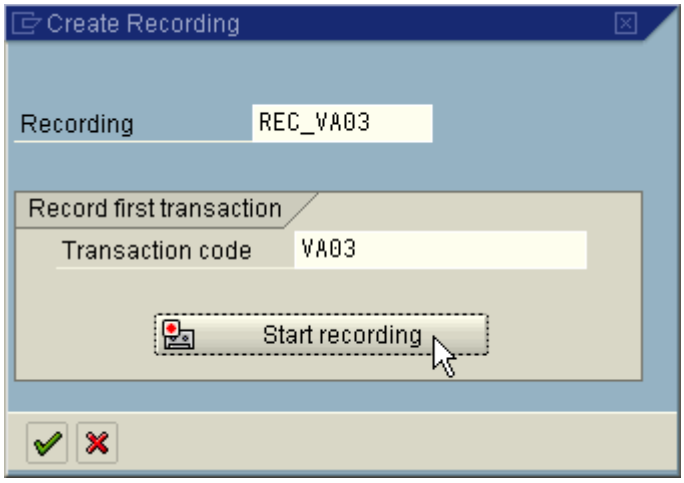
For single selection of a process instance of the Order processing process type group in PPM, the **VA03 transaction (Display order)** should be called in the SAP system using the **sapppm** ID. The transaction is to be assigned the **AT_SALES_ORDER_NUMBER** (order number of the selected process instance) PPM process attribute.

Below you will see how to use the SAP transaction recorder to record the **VA03 transaction (Display order)** taking into account the given requirements.

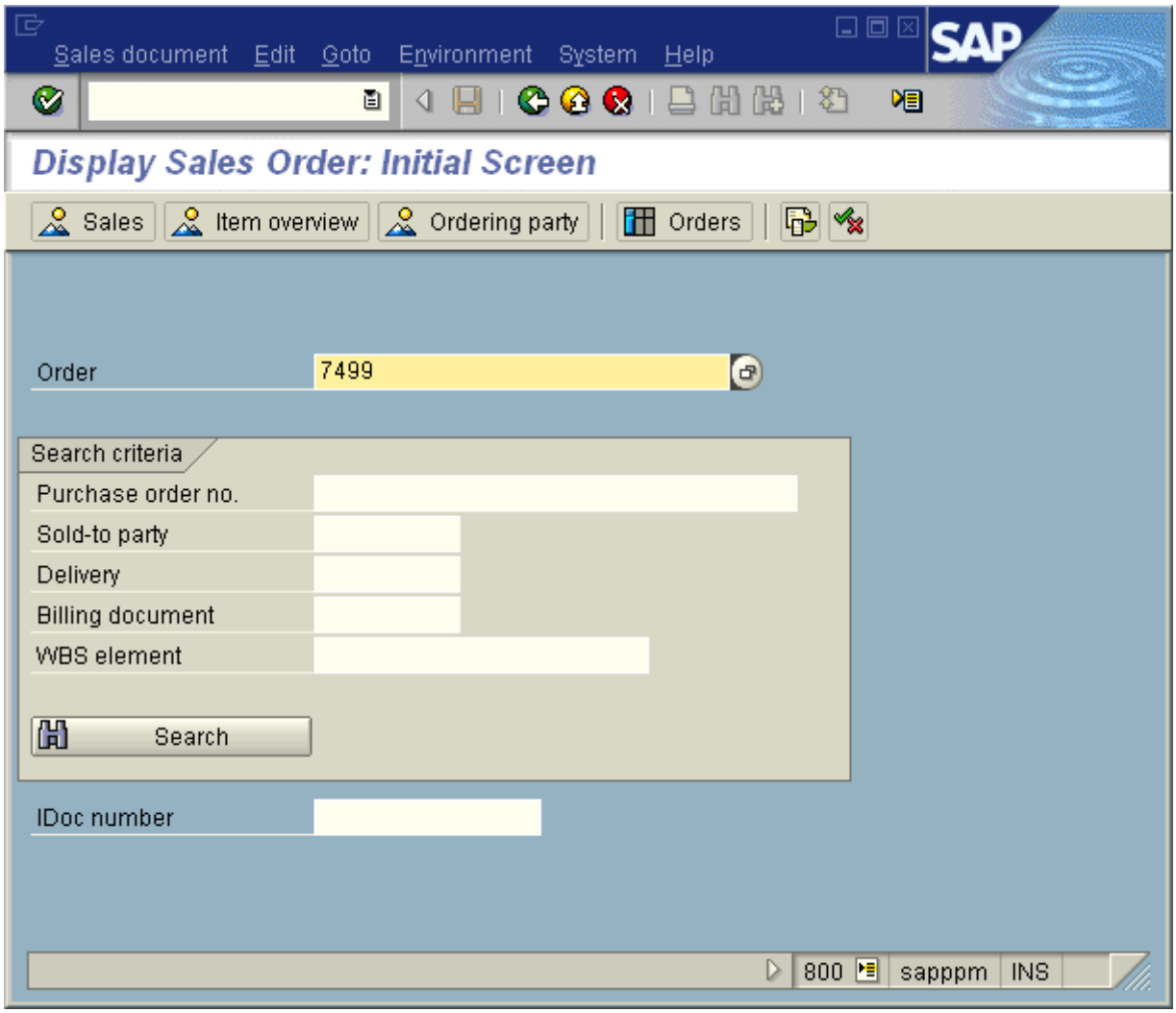
Launch the transaction recorder in the SAP front-end (SHDB transaction). The following screen is displayed:



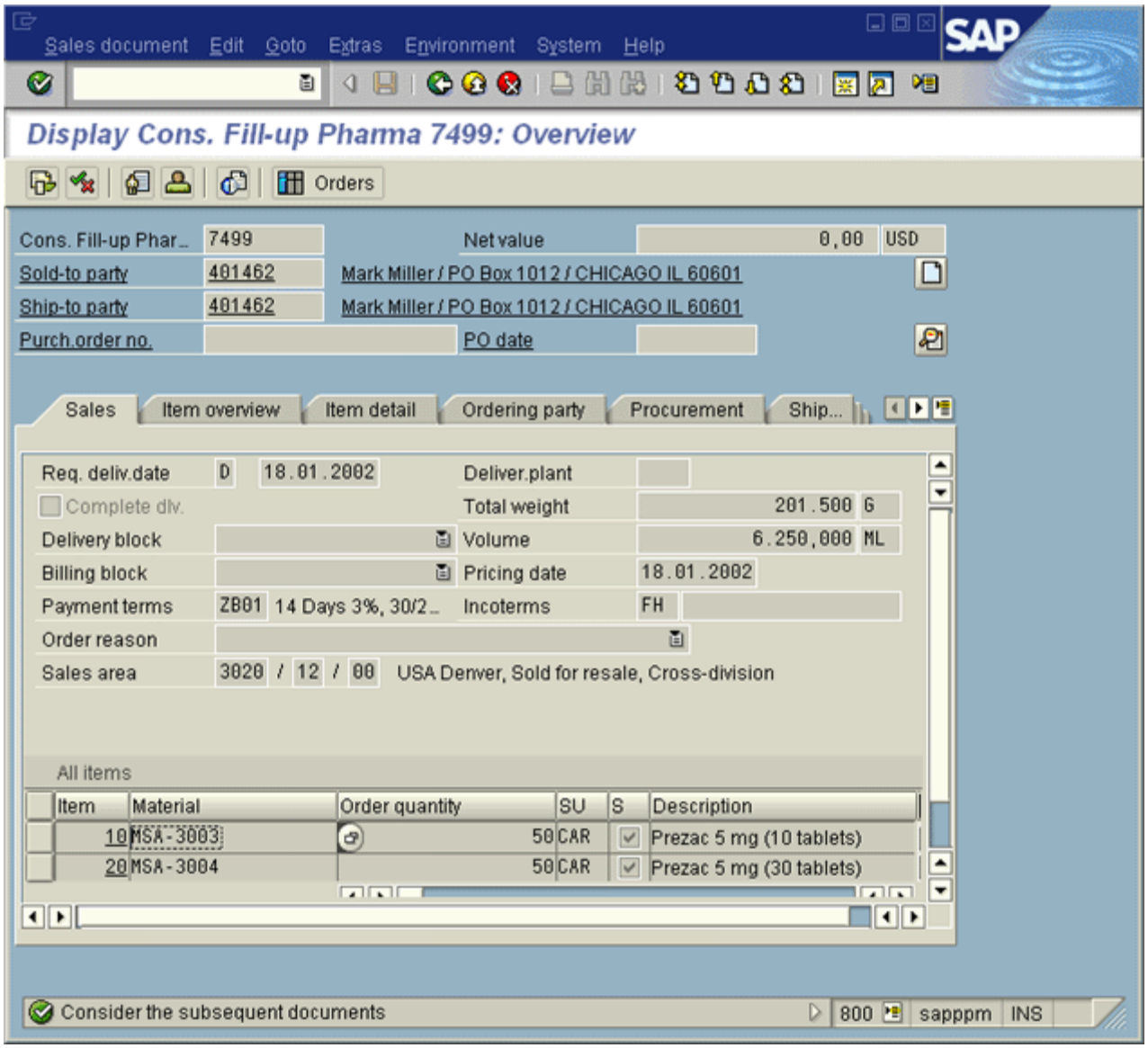
Create a new record and enter any name in the Record box to be used for saving the record. In the Transaction code box, enter the name of the transaction to be recorded:



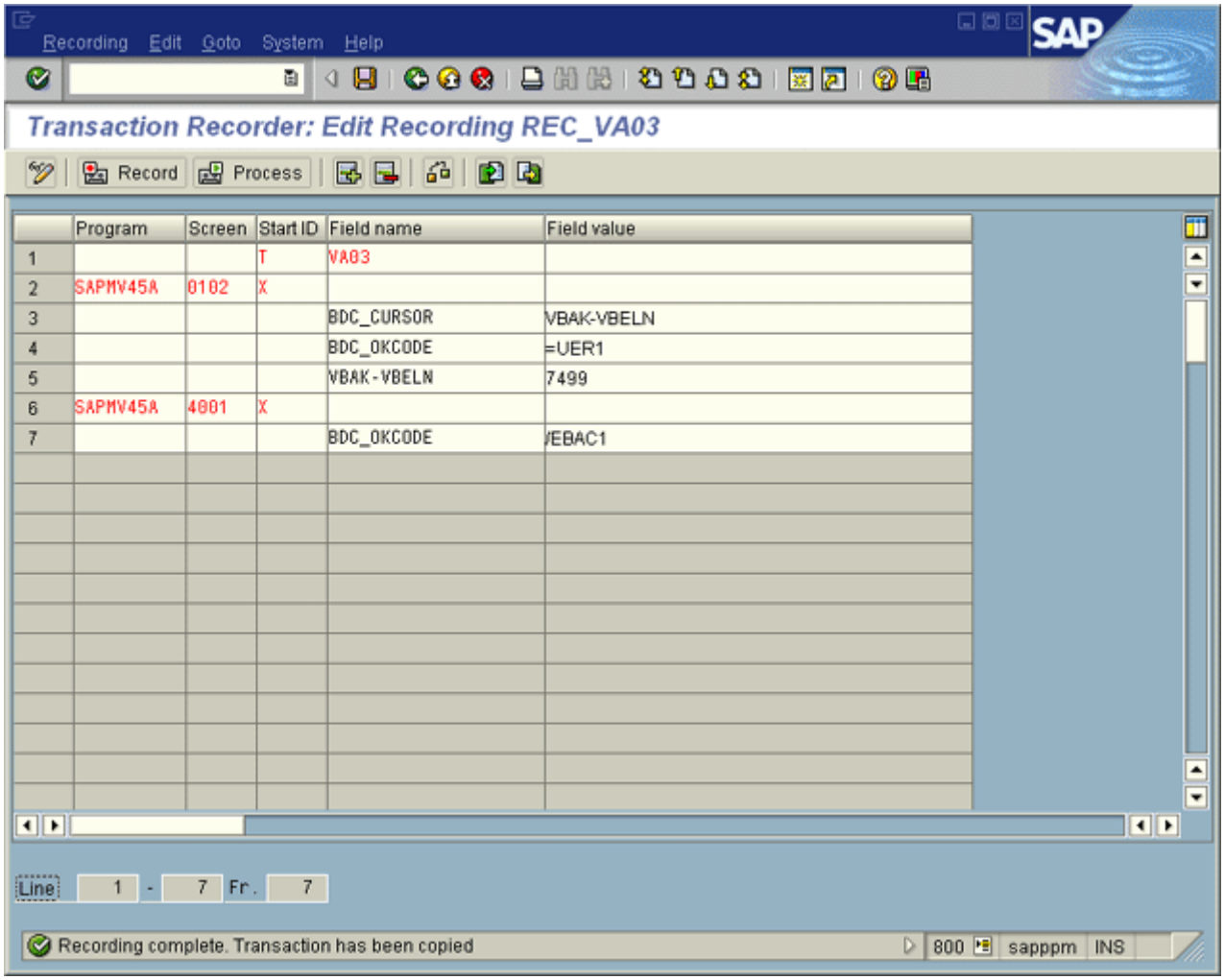
Now start recording and enter the required data in the following screen, that is, in the boxes to be filled with PPM process attributes, you enter the corresponding values and complete the boxes that are to be filled with fixed values when the transaction is called. In this example, there are no fixed values to be preset, only the order number to be transferred from the PPM process instance. Enter an order number that exists in your system in the Order box.



Confirm your entries with the **F5** key and display the data for order **7499**:



Exit the transaction using **Back** (F3). You have returned to the transaction recorder and see your entries in the ABAP batch input format:




The first row of the batch input script refers to the transaction call. The last two rows represent the order display and the use of the **Back** button. These rows can be ignored during the subsequent creation of the XML transaction configuration.

The content of all other rows needs to be transferred into the XML format of the batchinputline elements.

| Column name in ABAP batch input format | XML attribute |
|---|---|
| Program | program |
| Dynpro | dynpro |
| Start indicator | dynprobegin |
| Field name | fieldname |
| Field value | fieldvalue |

Fields with no value do not need to be specified as this corresponds to the default value of fieldvalue. The XML format in this example looks like this:

```
...
<batchinputline program="SAPMV45A" dynpro="0102"
                              dynprobegin="X"/>
<batchinputline fieldname="BDC_CURSOR"
                      fieldvalue="VBAK-VBELN"/>
<batchinputline fieldname="BDC_OKCODE"
                          fieldvalue="=UER1"/>
<batchinputline fieldname="VBAK-VBELN"
                           fieldvalue="7499"/>
...
```

To ensure that the order **7499** is not always displayed, regardless of the process instance in the Order processing process type group from which you call the **VA03** transaction, replace the static value for the order number field in the final batchinputline element (in this case: VBAK-VBELN) with the corresponding PPM process attribute that contains the order number in your process instance, for example, AT_SALES_ORDER_NUMBER.

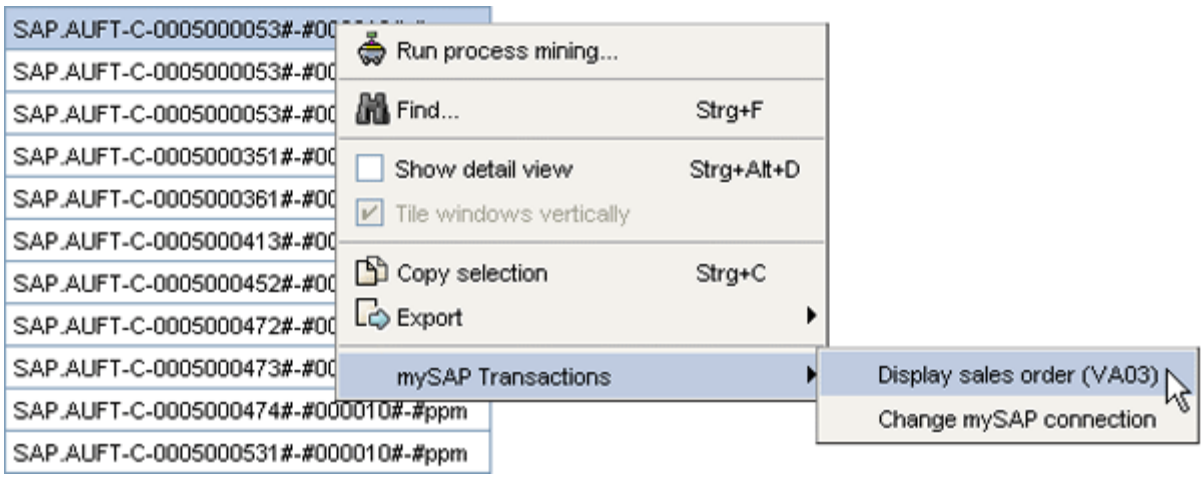
Combined with (sample) connection data, the transaction configuration now looks as follows (ABAP batch input data in bold):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE r3transactionconf SYSTEM
                        "mysaptransaction.dtd">
<r3transactionconf>
  <submenu>
    <description language="de"
                   name="SAP-Transaktionen"/>
    <description language="en"
                    name="SAP transactions"/>
  </submenu>
  <logoutMenuEntry>
    <description language="de"
                 name="SAP-Verbindung ändern"/>
    <description language="en"
                 name="Change SAP connection"/>
  </logoutMenuEntry>
  <r3system systemid="sapppm" client="800">
    <description language="de"
                 name="SAP-System 'sapppm' "/>
    <description language="en"
                 name="SAP system 'sapppm' "/>
    <locales>
      <defaultlocale value="de"/>
      <locale value="en"/>
    </locales>
    <applicationserver appserver="sapppm"
                           systemnumber="00"/>
  </r3system>
  <transaction systemid="sapppm"
       transactioncode="VA03" transactionid="VA03"
               proctypegroup="Order processing">
    <description language="de"
                 name="Auftrag anzeigen (VA03)"/>
    <description language="en"
               name="Display sales order (VA03)"/>
    <batchinputline program="SAPMV45A"
                  dynpro="0102" dynprobegin="X"/>
    <batchinputline fieldname="BDC_CURSOR"
                      fieldvalue="VBAK-VBELN"/>
    <batchinputline fieldname="BDC_OKCODE"
```

```
                                fieldvalue="=UER1"/>
    <batchinputline fieldname="VBAK-VBELN"
            attributname="AT_SALES_ORDER_NUMBER"/>
  </transaction>
</r3transactionconf>
```

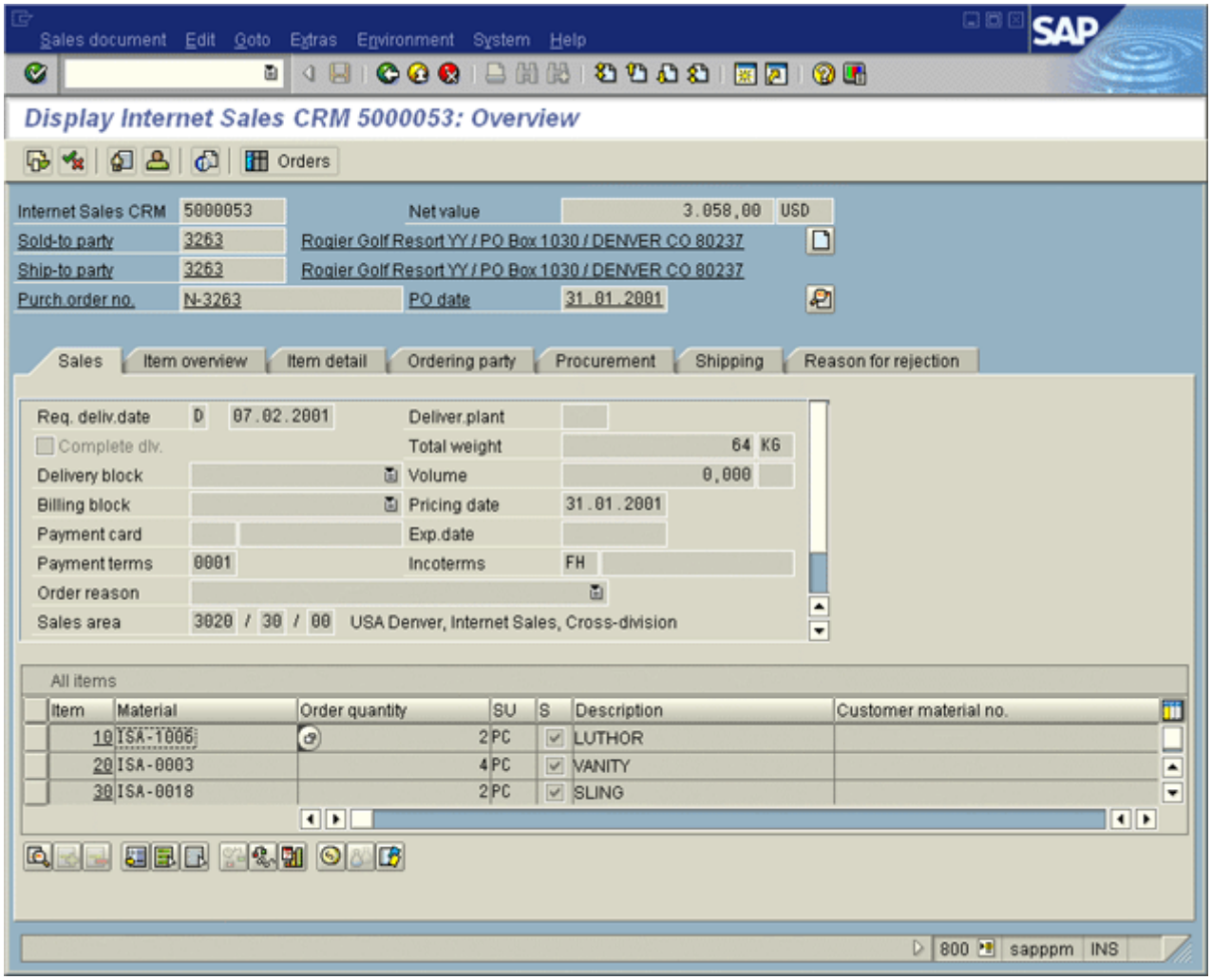
## DISPLAY IN PPM

The pop-up menu for calling a transaction on a selected process instance in the **Order processing\Other orders** process type group in line with the above configuration looks like this:




The SAP login dialog in line with the configuration looks like this:

After the user has successfully been authenticated in the SAP system, the **VA03** transaction is called in the SAP front-end, and the data pertaining to the order number of the selected process instance (here: 5000053) is displayed:




## RECORD ME5F MULTI-SELECT TRANSACTION

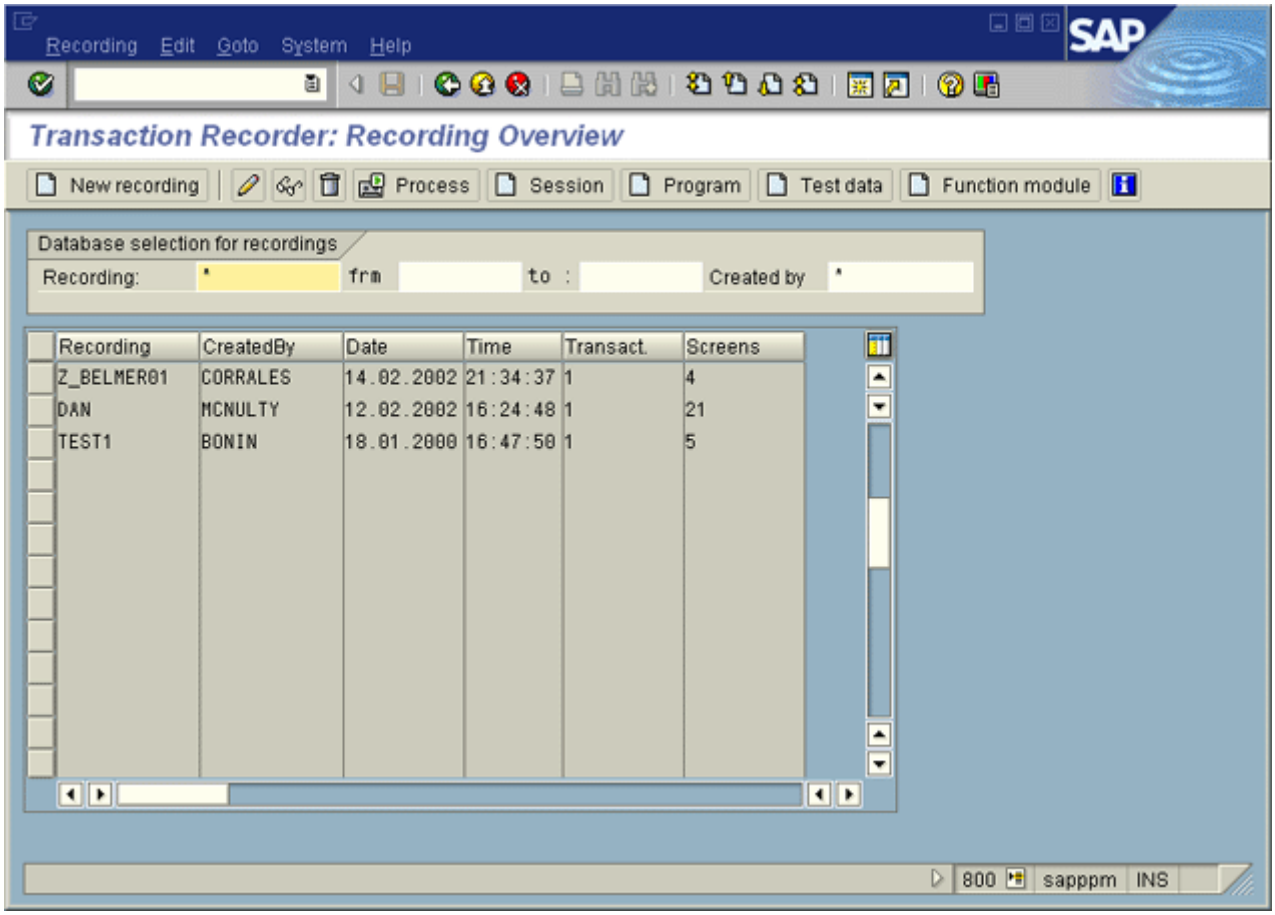
### Requirements

If several process instances from the Purchase requisitions process type group are selected in the process instance table, it should be possible to call the **ME5F** transaction (release reminder: Purchase requisitions) in the SAP system using the **sapppm** ID. Use **KY** as the release code and **01** as the release group for each call of the **ME5F** transaction.

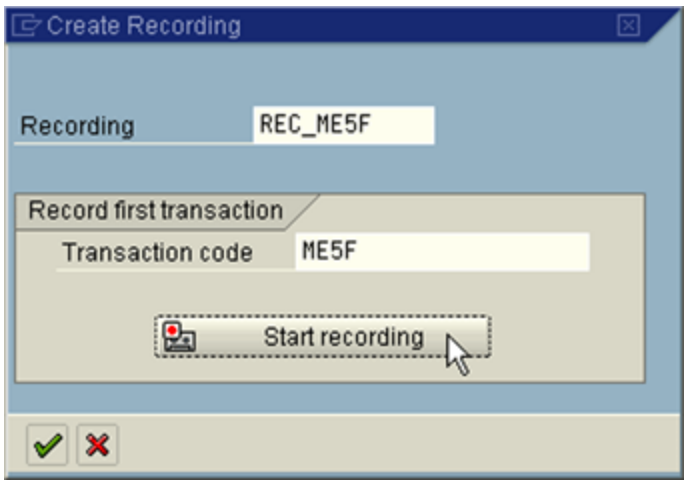
The values of the PPM **AT_BANF_NUMBER** (purchase requisition number) process attribute for the selected process instances should be transferred to the transaction.

Below you will see how to use the SAP transaction recorder to record the **ME5F** transaction (purchase requisition release reminder) taking into account the given requirements.

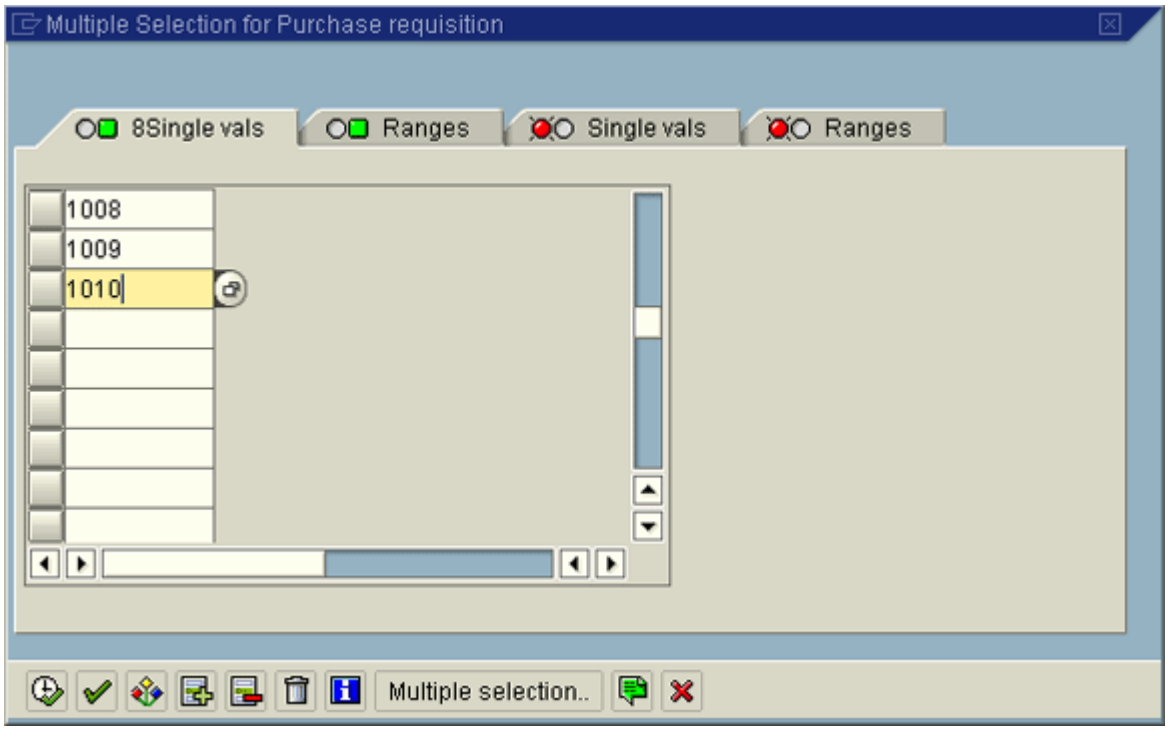
Launch the transaction recorder in the SAP front-end (SHDB transaction). The following screen is displayed:



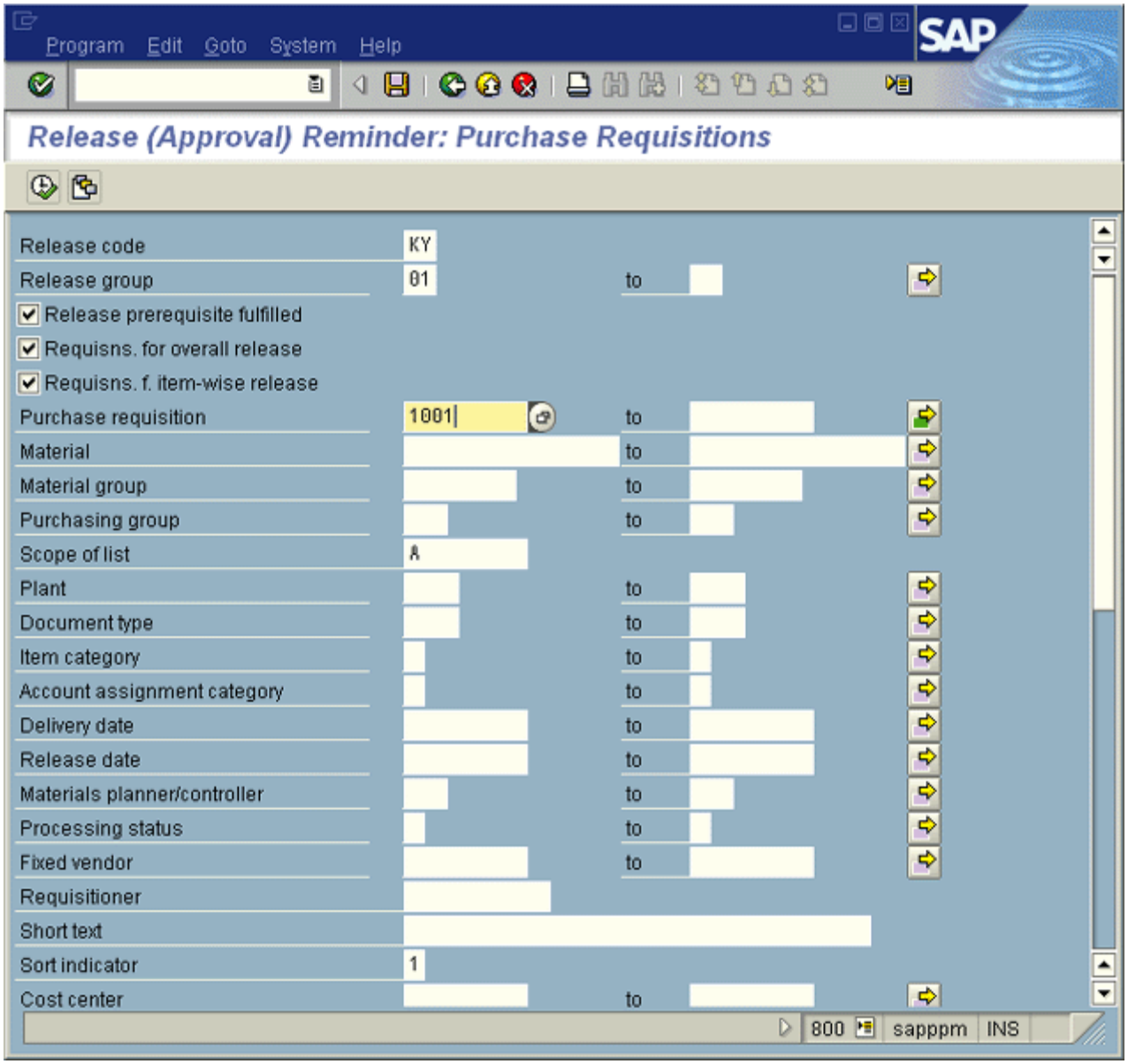

Create a new record and enter any name in the Record box to be used for saving the record. In the **Transaction code** box, enter the name of the transaction to be recorded:



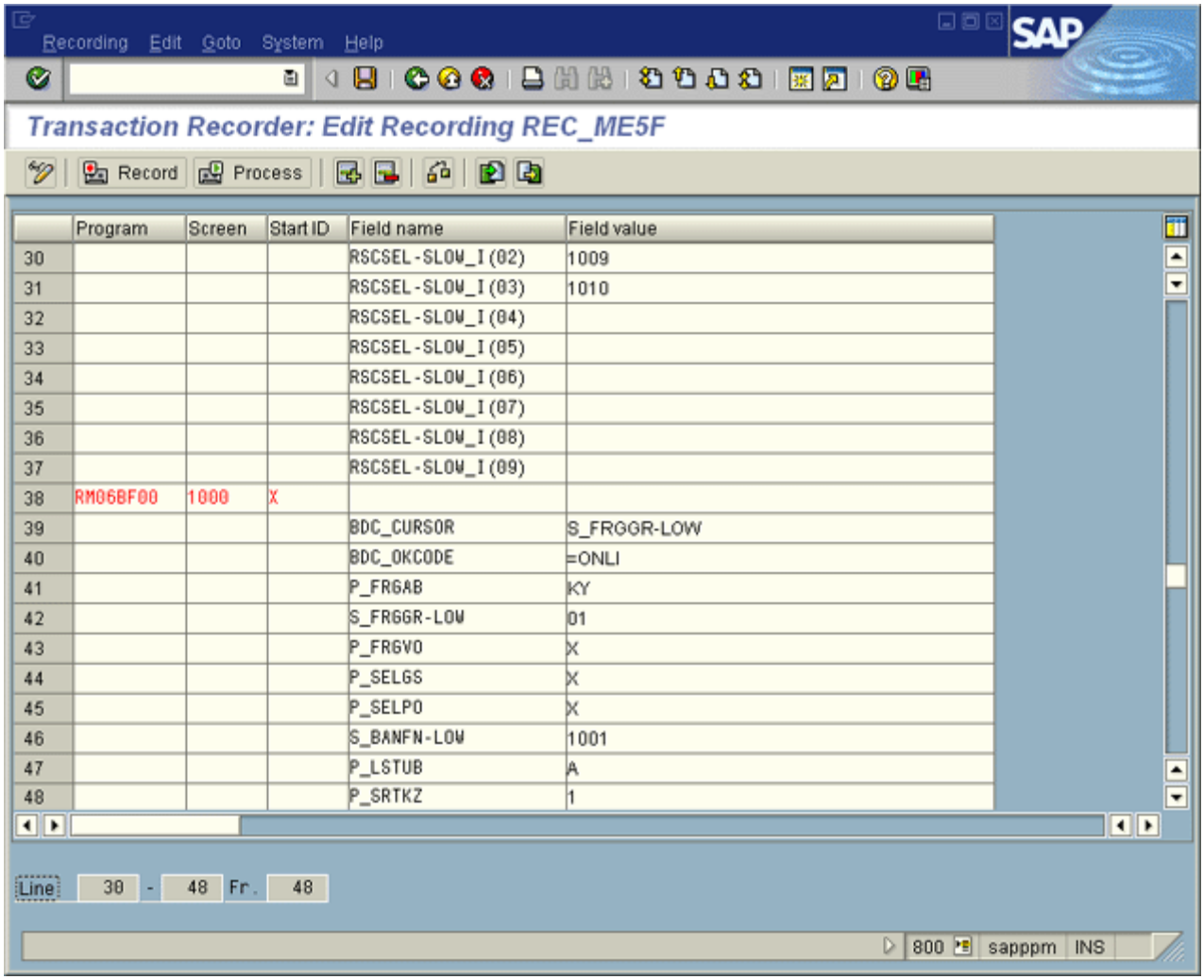
Now start recording and enter the required information in the following screen. For purchase requisition numbers, enter the numbers **1001 - 1010** as individual values. In the dialog box, use the **Down** key to navigate in order to avoid a non-functional OK code in the ABAP batch script using vertical scrolling.



In the **Release code** box, enter the value **KY** and in the **Release group** box, enter the value **01**. Apply your entries by pressing the **F8** key. Your transaction now looks as follows:



Press **F8** again to execute the transaction with the specified values and display the corresponding purchase requirements in ABAP batch script format:




| | Program | Screen | Start ID | Field name | Field value |
|---|---|---|---|---|---|
| 30 | | | | RSCSEL-SLOW_I(02) | 1009 |
| 31 | | | | RSCSEL-SLOW_I(03) | 1010 |
| 32 | | | | RSCSEL-SLOW_I(04) | |
| 33 | | | | RSCSEL-SLOW_I(05) | |
| 34 | | | | RSCSEL-SLOW_I(06) | |
| 35 | | | | RSCSEL-SLOW_I(07) | |
| 36 | | | | RSCSEL-SLOW_I(08) | |
| 37 | | | | RSCSEL-SLOW_I(09) | |
| 38 | RM06BF00 | 1000 | X | | |
| 39 | | | | BDC_CURSOR | S_FRGGR-LOW |
| 40 | | | | BDC_OKCODE | =ONLI |
| 41 | | | | P_FRGAB | KY |
| 42 | | | | S_FRGGR-LOW | 01 |
| 43 | | | | P_FRGVO | X |
| 44 | | | | P_SELGS | X |
| 45 | | | | P_SELPO | X |
| 46 | | | | S_BANFN-LOW | 1001 |
| 47 | | | | P_LSTUB | A |
| 48 | | | | P_SRTKZ | 1 |

Instead of applying the values directly from the SAP front-end, you can also export the script as a DAT file using Shift+F8 and display the contents including line numbers in the editor.

```
 1      0000    T   ME5F
 2  RM06BF00    1000    X
 3      0000        BDC_CURSOR  S_FRGGR-LOW
 4      0000        BDC_OKCODE  =%005
 5      0000        P_FRGAB KY
 6      0000        S_FRGGR-LOW 01
 7      0000        P_FRGVO X
 8      0000        P_SELGS X
 9      0000        P_SELPO X
10      0000        P_LSTUB A
11      0000        P_SRTKZ 1
12  SAPLALDB    3000    X
13      0000        BDC_OKCODE  =P+
14      0000        BDC_SUBSCR  SAPLALDB                                3010SCREEN_HEADER
15      0000        BDC_CURSOR  RSCSEL-SLOW_I(08)
16      0000        RSCSEL-SLOW_I(01)   1001
17      0000        RSCSEL-SLOW_I(02)   1002
18      0000        RSCSEL-SLOW_I(03)   1003
19      0000        RSCSEL-SLOW_I(04)   1004
20      0000        RSCSEL-SLOW_I(05)   1005
21      0000        RSCSEL-SLOW_I(06)   1006
22      0000        RSCSEL-SLOW_I(07)   1007
23      0000        RSCSEL-SLOW_I(08)   1008
24      0000        RSCSEL-SLOW_I(09)   __________
25  SAPLALDB    3000    X
26      0000        BDC_OKCODE  =ACPT
27      0000        BDC_SUBSCR  SAPLALDB                                3010SCREEN_HEADER
28      0000        BDC_CURSOR  RSCSEL-SLOW_I(03)
29      0000        RSCSEL-SLOW_I(01)   1008
30      0000        RSCSEL-SLOW_I(02)   1009
31      0000        RSCSEL-SLOW_I(03)   1010
32      0000        RSCSEL-SLOW_I(04)
33      0000        RSCSEL-SLOW_I(05)
34      0000        RSCSEL-SLOW_I(06)
35      0000        RSCSEL-SLOW_I(07)
36      0000        RSCSEL-SLOW_I(08)
37      0000        RSCSEL-SLOW_I(09)
38  RM06BF00    1000    X
39      0000        BDC_CURSOR  S_FRGGR-LOW
40      0000        BDC_OKCODE  =ONLI
41      0000        P_FRGAB KY
42      0000        S_FRGGR-LOW 01
43      0000        P_FRGVO X
44      0000        P_SELGS X
45      0000        P_SELPO X
46      0000        S_BANFN-LOW 1001
47      0000        P_LSTUB A
48      0000        P_SRTKZ 1
```

The first row of the batch input script refers to the transaction call. Ignore this row and rows **41-48**; they contain repeat entries from rows **5-11** or (as in row **46**) automatically transferred data from the bottom of the multiple selection dialog. Rows named **BDC_SUBSCR** (for example, row **14**) are to be ignored in the subsequent creation of the XML transaction configuration.

The content of all other rows needs to be transferred into the XML format of the batchinputline elements.

| Column name in ABAP batch input format | DAT file row number | XML attribute |
|---|---|---|
| Program | 2, 12, 25, 38 | program |
| Dynpro | 2, 12, 25, 38 | dynpro |
| Start indicator | 2, 12, 25, 38 | dynprobegin |

| Column name in ABAP batch input format | DAT file row number | XML attribute |
|---|---|---|
| Field name | for example, 3-11<br>(1. screen) | fieldname |
| Field value | for example, 3-11<br>(1. screen) | fieldvalue |

You do not indicate fields without value because this corresponds to the fieldvalue default value.

```
...
<!-- 1st screen -->
<batchinputline program="RM06BF00" dynpro="1000"
                                dynprobegin="X"/>
<batchinputline fieldname="BDC_CURSOR"
                         fieldvalue="S_FRGGR-LOW"/>
<batchinputline fieldname="BDC_OKCODE"
                               fieldvalue="=%005"/>
<batchinputline fieldname="P_FRGAB" fieldvalue="KY" />
<batchinputline fieldname="S_FRGGR-LOW" fieldvalue="01" />
<batchinputline fieldname="P_FRGVO" fieldvalue="X" />
<batchinputline fieldname="P_SELGS" fieldvalue="X" />
<batchinputline fieldname="P_SELPO" fieldvalue="X" />
<batchinputline fieldname="P_LSTUB" fieldvalue="A" />
<batchinputline fieldname="P_SRTKZ" fieldvalue="1" />

<!-- 2nd screen -->
<batchinputline program="SAPLALDB" dynpro="3000"
                                dynprobegin="X"/>
<batchinputline fieldname="BDC_OKCODE" fieldvalue="=P+" />
<batchinputline fieldname="BDC_CURSOR"
                     fieldvalue="RSCSEL-SLOW_I(08)"/>
<batchinputline fieldname="RSCSEL-SLOW_I(01)"
                               fieldvalue="1001"/>
<batchinputline fieldname="RSCSEL-SLOW_I(02)"
                               fieldvalue="1002"/>
<batchinputline fieldname="RSCSEL-SLOW_I(03)"
                               fieldvalue="1003"/>
<batchinputline fieldname="RSCSEL-SLOW_I(04)"
                               fieldvalue="1004"/>
<batchinputline fieldname="RSCSEL-SLOW_I(05)"
                               fieldvalue="1005"/>
<batchinputline fieldname="RSCSEL-SLOW_I(06)"
                               fieldvalue="1006"/>
<batchinputline fieldname="RSCSEL-SLOW_I(07)"
                               fieldvalue="1007"/>
<batchinputline fieldname="RSCSEL-SLOW_I(08)"
                               fieldvalue="1008"/>
<batchinputline fieldname="RSCSEL-SLOW_I(09)"
                          fieldvalue="__________"/>

<!-- 3rd screen -->
<batchinputline program="SAPLALDB" dynpro="3000"
                                dynprobegin="X"/>
<batchinputline fieldname="BDC_OKCODE"
                               fieldvalue="=ACPT"/>
<batchinputline fieldname="BDC_CURSOR"
                     fieldvalue="RSCSEL-SLOW_I(03)"/>
```

```
<batchinputline fieldname="RSCSEL-SLOW_I(01)"
                                fieldvalue="1008"/>
<batchinputline fieldname="RSCSEL-SLOW_I(02)"
                                fieldvalue="1009"/>
<batchinputline fieldname="RSCSEL-SLOW_I(03)"
                                fieldvalue="1010"/>
<batchinputline fieldname="RSCSEL-SLOW_I(04)"
                                   fieldvalue=""/>
<batchinputline fieldname="RSCSEL-SLOW_I(05)"
                                   fieldvalue=""/>
<batchinputline fieldname="RSCSEL-SLOW_I(06)"
                                   fieldvalue=""/>
<batchinputline fieldname="RSCSEL-SLOW_I(07)"
                                   fieldvalue=""/>
<batchinputline fieldname="RSCSEL-SLOW_I(08)"
                                   fieldvalue=""/>
<batchinputline fieldname="RSCSEL-SLOW_I(09)"
                                   fieldvalue=""/>

<!-- 4th screen -->
<batchinputline program="RM06BF00" dynpro="1000"
                                 dynprobegin="X"/>
<batchinputline fieldname="BDC_OKCODE"
                               fieldvalue="=ONLI"/>
...
```

This transaction configuration would actually work with the connection data from the previous example. However, no matter which process instances in the **Purchase requisitions** process type group you would select, one transaction pertaining to a purchase requisition with the number **1001-1010** would always be displayed.

In order to display the proper purchase requisition for each process instance, you need to replace both multiple selection screens (all batchinputline XML elements from the second and third screen) in the current configuration with one batchinputlist XML element only. You need the following data:

| XML attribute | Value (description) |
|---|---|
| program | SAPLALDB (program name) |
| dynpro | 3000 (dynpro name) |
| okcodefieldname | BDC_OKCODE (name of the dynpro field that contains the OK code) |
| okcodepagedown | =P+ (OK code value for paging down) |
| okcodeaccept | =ACPT (OK code value to accept the entry in multiple selection) |
| fieldname | RSCSEL-SLOW_I (Name of the field that you want to assign the PPM attribute values to. The field must not contain row indices [(01), (02), etc.].) |

| XML attribute | Value (description) |
|---|---|
| attributname | AT_BANF_NUMBER (internal name of the PPM process attribute whose values are going to be transferred to the called transaction) |
| linesperpage | 9 (number of entry rows on each multiple selection screen) |

Based on these entries, the batchinputlist element now looks like this:

```
<batchinputlist program="SAPLALDB" dynpro="3000"
  okcodefieldname="BDC_OKCODE" okcodepagedown="=P+"
  okcodeaccept="=ACPT" fieldname="RSCSEL-SLOW_I"
  attributname="AT_BANF_NUMBER" linesperpage="9" />
```

In combination with the connection data from the single select example, the transaction configuration now looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE r3transactionconf SYSTEM
                              "mysaptransaction.dtd">
<r3transactionconf>
  <submenu>
    <description language="de"
                         name="SAP-Transaktionen"/>
    <description language="en"
                          name="SAP transactions"/>
  </submenu>
  <logoutMenuEntry>
    <description language="de"
                     name="SAP-Verbindung ändern"/>
    <description language="en"
                     name="Change SAP connection"/>
  </logoutMenuEntry>
  <r3system systemid="sapppm" client="800">
    <description language="de"
                      name="SAP-System 'sapppm' "/>
    <description language="en"
                      name="SAP system 'sapppm' "/>
    <locales>
      <defaultlocale value="de"/>
      <locale value="en"/>
    </locales>
    <applicationserver appserver="sapppm"
                                systemnumber="00"/>
  </r3system>
  <transaction
    systemid="sapppm" transactioncode="ME5F"
                              transactionid="ME5F">
    <description language="de"
                name="Freigabeerinnerung BANF (ME5F)"/>
    <description language="en"
                  name="Release (approval) reminder:
                      Purchase Requisitions (ME5F)"/>
    <!-- 1st screen -->
```

```
    <batchinputline program="RM06BF00" dynpro="1000"
                                  dynprobegin="X"/>
    <batchinputline fieldname="BDC_CURSOR"
                          fieldvalue="S_FRGGR-LOW"/>
    <batchinputline fieldname="BDC_OKCODE"
                               fieldvalue="=%005"/>
    <batchinputline fieldname="P_FRGAB" fieldvalue="KY" />
    <batchinputline fieldname="S_FRGGR-LOW"
                                  fieldvalue="01"/>
    <batchinputline fieldname="P_FRGVO" fieldvalue="X" />
    <batchinputline fieldname="P_SELGS" fieldvalue="X" />
    <batchinputline fieldname="P_SELPO" fieldvalue="X" />
    <batchinputline fieldname="P_LSTUB" fieldvalue="A" />
    <batchinputline fieldname="P_SRTKZ" fieldvalue="1" />
    <!-- Multiple selection screens -->
    <batchinputlist program="SAPLALDB" dynpro="3000"
       okcodefieldname="BDC_OKCODE" okcodepagedown="=P+"
       okcodeaccept="=ACPT" fieldname="RSCSEL-SLOW_I"
       attributname="AT_BANF_NUMBER" linesperpage="9" />
    <!-- 1st screen -->
    <batchinputline program="RM06BF00" dynpro="1000"
                                  dynprobegin="X"/>
    <batchinputline fieldname="BDC_OKCODE"
                               fieldvalue="=ONLI"/>
  </transaction>
</r3transactionconf>
```

### 11.1.4.2 Explanations regarding the DTD

The file **mysaptransaction.dtd** determines the configuration options:

#### POP-UP MENU CONFIGURATION

| XML tag | Description |
|---|---|
| **submenu** (optional) | Name of the submenu. Default value: **SAP transactions**. If only one submenu entry exists, it is displayed directly in the pop-up menu without submenu entry (e. g., initial login with only one configured transaction) |
| **logoutMenuEntry** (optional) | Menu entry for resetting all connection parameters after at least one successful SAP login. Default value: **Reset connection parameters** of all **SAP systems** |

## SYSTEM ACCESS CONFIGURATION

| XML tag | Description |
|---|---|
| **r3system** | Description and connection data of the available SAP system. Data of any number of SAP systems can be specified. The configuration of system access is described in detail in the **PPM Process Extractors** technical reference. |
| **description** | At least the description in the default language must be specified. |
| **name** | Name of the SAP system in the SAP logon dialog in the PPM front-end |
| **systemid** | Unique name of an SAP system. Is referenced by the corresponding transaction. |
| **client** | Name of the SAP client. |
| **locales** | Languages available in the SAP system |
| **defaultlocale** | Default language. Is preselected in the SAP login dialog |
| **locale** (optional) | Additional language(s) available in the SAP system |
| **appserver** | Computer name or IP address of the SAP source system computer |
| **systemnumber** | SAP system number |
| **mshost** | Name of the SAP message host |
| **r3name** | R/3 system name |
| **group** | Name of application server group |
| **appserver** | Name of application server |
| **systemnumber** | SAP system number |
| **gwhost** | Computer name of R/3 gateway |
| **gwserv** | Service number of the R/3 gateway |

## TRANSACTION CONFIGURATION

| XML tag | Description |
|---|---|
| **transaction** | Transaction configuration |

| XML tag | Description |
|---|---|
| **systemid** | ID of the SAP system in which the transaction is to be called. Must correspond to the value of a **systemid** of the specified SAP systems (**r3system** XML elements). |
| **transactionid** | Transaction ID |
| **transactioncode** | Transaction code of the transaction to be called (see SAP documentation) |
| **skipfirstscreen** (optional) | Skips the transaction start page if all mandatory fields are completed (see SAP documentation on the CALL_TRANSACTION function module).<br>Valid values: **yes** \| **no**<br>Default value: **yes** |
| **mode** (optional) | Execution mode of the ABAP batch input (see SAP documentation on the CALL_TRANSACTION function module). Valid values:<br>**SHOW_DYNPROS**<br>(Dynpros are displayed during execution)<br>**SHOW_DYNPROS_ONLY_ON_ERRORS**<br>(Dynpros are displayed only if an error occurs or when the end of the batch script is reached)<br>**DONT_SHOW_DYNPROS**<br>(Dynpros are not displayed)<br>Default value:<br>**SHOW_DYNPROS_ONLY_ON_ERRORS** |
| **update** (optional) | Update type in the SAP system (see SAP documentation on the CALL_TRANSACTION function module). Valid values:<br>**SYNCHRONOUS** (synchronous update)<br>**ASYNCHRONOUS** (asynchronous update)<br>**LOCAL** (local update)<br>Default value: **ASYNCHRONOUS** |
| **proctypegroup** (optional) | Process type group in which the transaction is available. The transaction is automatically available in all process types of the specified process type group. If this entry is missing, the transaction is available in the entire process tree. |
| **description** | Language-specific interface name of the transaction |
| **name** | Pop-up menu entry of the transaction |

| XML tag | Description |
|---|---|
| **batchinputline** | Line in ABAP batch input format. If a transaction configuration contains only **batchinputline** XML elements, the transaction can normally be called for single selection only. Multiple selection is possible only if the PPM process attribute specified with **attributname** has the same value in all selected process instances. |
| **program** | Name of the program |
| **dynpro** | Dynpro name |
| **dynprobegin** | Start of a dynpro |
| **fieldname** | Name of the dynpro field |
| **attributname** (optional) | Internal name of a PPM process attribute whose value is to be determined by the selected process instance. The value is assigned to the dynpro field, **fieldvalue** is ignored. |
| **fieldvalue** (optional) | A constant value to be assigned to the dynpro field. If **attributname** is specified **fieldvalue** is ignored. |
| **batchinputlist** | Multiple lines in ABAP batch input format. If a transaction configuration contains at least one **batchinputlist** XML element, the transaction can be called with both single and multiple selection. |
| **program** | Name of the program |
| **dynpro** | Dynpro name |
| **okcodefieldname** (optional) | Name of the dynpro field that contains the OK code. Default value: **BDC_OKCODE** |
| **okcodepagedown** (optional) | OK code value for paging down. Default value: **=P+** |
| **okcodeaccept** (optional) | OK code value to accept the entry. Default value: **=ACPT** |
| **fieldname** | Name of the dynpro field |
| **attributname** | Internal name of a PPM process attribute whose value is to be determined by the selected process instance. The value is assigned to the dynpro field, **fieldvalue** is ignored. |

| XML tag | Description |
| --- | --- |
| **linesperpage** (optional) | Number of visible value lines on the dynpro. When this number of lines is reached, the system pages down. Default value: **9** |

# 12 Legal information

## 12.1 Documentation scope

The information provided describes the settings and features as they were at the time of publishing. Since documentation and software are subject to different production cycles, the description of settings and features may differ from actual settings and features. Information about discrepancies is provided in the Release Notes that accompany the product. Please read the Release Notes and take the information into account when installing, setting up, and using the product.

If you want to install technical and/or business system functions without using the consulting services provided by Software AG, you require extensive knowledge of the system to be installed, its intended purpose, the target systems, and their various dependencies. Due to the number of platforms and interdependent hardware and software configurations, we can describe only specific installations. It is not possible to document all settings and dependencies.

When you combine various technologies, please observe the manufacturers' instructions, particularly announcements concerning releases on their Internet pages. We cannot guarantee proper functioning and installation of approved third-party systems and do not support them. Always follow the instructions provided in the installation manuals of the relevant manufacturers. If you experience difficulties, please contact the relevant manufacturer.

If you need help installing third-party systems, contact your local Software AG sales organization. Please note that this type of manufacturer-specific or customer-specific customization is not covered by the standard Software AG software maintenance agreement and can be performed only on special request and agreement.

If a description refers to a specific ARIS product, the product is named. If this is not the case, names for ARIS products are used as follows:

| Name | Includes |
|---|---|
| ARIS products | Refers to all products to which the license regulations of Software AG standard software apply. |
| ARIS Client | Refers to all programs that access shared databases by using ARIS Server. |
| ARIS Download Client | Refers to an ARIS Client that can be accessed using a browser. |

## 12.2 Data protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR).

Where applicable, appropriate steps are documented in the respective administration documentation.