DATA IMPORT

ARIS Process Performance Manager 10

October 2026

This document applies to ARIS Process Performance Manager Version 10 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 © 2000-2026 SAG Aris GmbH, Saarbrücken, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name ARIS and all SAG Aris GmbH product names are either trademarks or registered trademarks of SAG Aris GmbH 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.

The Software may contain or be distributed with certain commercial third party components (“Third Party Components”), which may be subject to special license terms and conditions (“Third Party Terms”) located at https://aris.com/terms.

Third Party Terms include important licensing and warranty information and disclaimers of third party licensors. For the avoidance of doubt, if Customer uses the Software in accordance with the Documentation, the Third Party Terms do not restrict the rights granted under this Agreement. In the event that a third party manufacturer no longer provides active support for any Third Party Component, Supplier will use commercially reasonable efforts to respond to any issues with the Software related to the Third Party Components.

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with SAG Aris GmbH and/or its subsidiaries and/or its affiliates and/or their licensors.

Table of Contents

1. General 5
2. XML 6
2.1. What is XML? 6
2.2. Structure of an XML document 6
3. XML data import 7
3.1. Graph format 7
3.1.1. Object types 9
3.1.2. Connections 9
3.1.3. Relations (optional) 10
3.1.4. Guidelines for the graph structure 10
3.1.5. Attributes 10
3.1.6. XML example graph 11
3.2. System event format 14
3.2.1. Definition of process fragments 15
3.2.2. Definition of mapping 16
3.2.2.1. Definition of process fragment mapping 17
3.2.2.2. Definition of attribute mapping 20
3.2.2.3. Organizational units 27
3.2.2.4. Special case of attribute mapping 28
3.2.3. Create fragment definitions in ARIS 30
3.2.3.1. Modeling the overall process 30
3.2.3.2. Modeling the process fragment definitions 31
3.2.3.3. Format of system event file 34
3.2.3.4. Run the ARIS report 35
3.2.4. Generating the XML output file 36
3.2.5. Summary 37
3.3. Data formats 42
3.3.1. Special characters in XML documents 43
3.4. Generating the process instance fragments 43
3.4.1. Extending the attribute configuration 44
3.4.1.1. Specify the data type of unknown attributes 45
3.4.2. Extending the mapping configuration 49
3.4.3. Multi-valued system event attributes 50
3.4.4. Direct import of process attributes 51
3.4.5. Special case of scaled system 52
3.4.6. Archiving of XML import files 55
3.5. runxmlimport command line program 55
3.5.1. runxmlimport arguments 56
3.6. Import multiple data sources 58
3.7. Re-importing the same data 59
3.7.1. Graph format 59
3.7.2. System event format 60
4. Import of process instance-independent data 61
4.1. Process instance-independent measures 61
4.1.1. Data import formats 61
4.1.1.1. XML format 62
4.1.1.2. CSV format 65
4.1.1.3. XLS format 66
4.1.2. Data reimport 68
4.1.3. Export of values of process instance-independent data series 69
4.1.4. Deletion of values of process instance-independent data series 69
4.1.5. runpikidata command line program 70
4.2. Dimension values 72

4.2.1. XML format 73
4.2.2. CSV format 74
4.2.3. Default and replacement values 75
4.2.4. Data reimport 75
4.2.5. Delete dimension values 75
4.2.6. rundimdata command line program 75
4.3. Data analytics 77
5. How to handle large EPCs 78
5.1. Import large EPCs 78
5.2. Delete large EPCs 78
6. Appendix 79
6.1. Design of a Process Warehouse 79
6.1.1. Generate process fragments 81
6.1.2. Merge process fragments 82
6.1.2.1. Copying the process instance attributes 85
6.1.2.2. Making organizational units anonymous 85
6.1.3. Typify processes 85
6.1.4. Calculate measures 86
6.1.5. Checking planned values 86
6.2. Import scenarios 87
6.2.1. Parameter values 87
7. Support and legal information 89
7.1. Documentation scope 89
7.2. Support 89
7.3. Legal notices 90

1. General

PPM uses **XML** as a universal data format, which means that the entire PPM system can be configured using XML files.

This technical reference describes PPM's XML interfaces that are used to import source system data into the PPM system as XML files. For importing data, PPM uses its own data formats, namely **system event format** and **graph format**.

The source system data is first extracted from almost any source system type (for example, SAP, JDBC, CSV) using the PPM process extractors (system event format) or other adapters.

The **Design of a Process Warehouse [79]** chapter provides a design-based view of how extracted source system data can be used to generate process instances, which reflect the sequences of the source system processes and are available for further measure analysis in PPM.

2. XML

This chapter contains basic information about XML, which is necessary to understand the subsequent chapters.

2.1. What is XML?

The abbreviation XML stands for e**X**tensible **M**arkup **L**anguage. XML is a meta-language for the description of display languages such as HTML. Meta-languages provide the rules required for the definition of document types. Display languages allow documents to be output correctly.

2.2. Structure of an XML document

An XML document is a text file and consists of two character types: the actual data and the so-called tags or markups. Tags are XML instructions, which describe the division of the document into storage units and its logical structure. The structure itself is saved in a document type definition (DTD).

Tags are always written in pairs in angle brackets. Every start tag always has a corresponding end tag.

XML attributes are used within the tags. An attribute may only occur once within a tag.

XML documents consist of elements. An element is made up of two XML tags and the enclosed text. Blank elements consist of only one tag and always end with a slash (/) before the final bracket.

You can create simple XML documents with a text editor. In the following example, the DTD is specified in square brackets in the XML file directly:

```
<?xml version="1.0"?>
<!DOCTYPE memberlist
[
<!ELEMENT memberlist (no, name, age)>
<!ELEMENT no (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
]>
<memberlist>
<no>001</no>
<name>Doe, John</name>
<age>27</age>
</memberlist>
```

If you save this document under the name of your choice with the extension **.xml**, Internet Explorer can display the document in a structured form.

3. XML data import

This chapter describes the XML-based import of process instance data.

The instance data for the actual processes completed is extracted from the operational application system (source system) by special software and saved in XML output files. These output files are imported into PPM using the XML import interface. The internal structure of the XML files is specified by a DTD (Document Type Definition).

?The PPM XML import interface supports two different import formats, PPM graph format and PPM system event format.

PPM graph format

PPM graph format is used to import already structured process data from process-oriented application systems (for example, workflow systems). The application-specific adapter generates XML files, in which process instances including their procedural logic are described in PPM graph format. In contrast to PPM system event format, complete process instances can be imported. A merge operation is not necessary. When importing complete process instances, for a new import of the instance data, complete process instances must always be imported.

Graph format is used within the PPM system for the universal exchange of EPC-based data.

PPM system event format

PPM system event format is used for all activity-oriented application systems, in which the information making up the process (procedural logic) cannot be extracted.

When importing data in system event format, system events are logged in an XML file. All types of system events, which are to be imported to PPM, must be defined in process fragment models before importing. Rules are also defined for how these process fragment models are merged into an overall process.

PPM generates process instance fragments by mapping the system events to process fragment models. These are then linked to form process instances.

System event format allows process instances already imported to be extended and modified by importing delta data.

3.1. Graph format

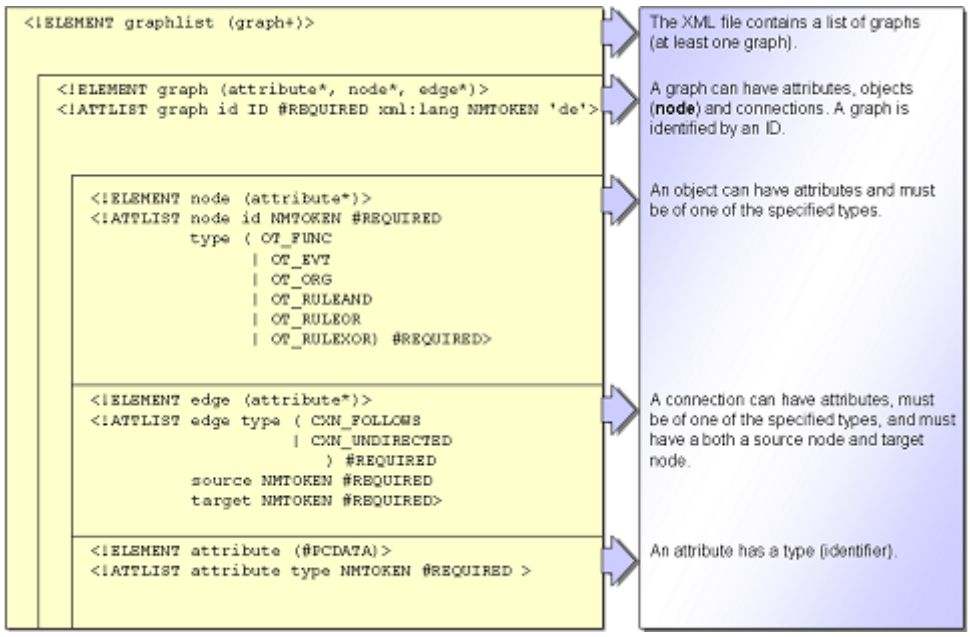
An XML file in PPM graph format contains a list of graphs (EPCs). Each graph represents a process instance or a process instance fragment. A graph is made up of different types of objects, connections and any object relations. The graph and the objects, connections and relations can have attributes.

The XML file below contains a simple graph, which is made up of three linked objects (event – function – event):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE graphlist SYSTEM "graph.dtd">
<graphlist>
<graph id="00093862" xml:lang="en">
<attribute type="AT_ID">XMLGraph-Job-00093862</attribute>
<attribute type="AT_EPK_KEY">00093862</attribute>
```

```
<attribute type="AT_PROCTYPE">Standard order</attribute>
<attribute type="AT_PROCTYPEGROUP">Order processing</attribute>

<node id="Start" type="OT_EVT">
<attribute type="AT_OBJNAME_INTERN">AUFTRAG_ANZU</attribute>
<attribute type="AT_OBJNAME">Customer order to be created</attribute>
</node>

<node id="Function" type="OT_FUNC">
<attribute type="AT_OBJNAME_INTERN">AUFTRAG</attribute>
<attribute type="AT_OBJNAME">Create customer order</attribute>
<attribute type="AT_START_TIME">14.2.2000 13:12:57</attribute>
<attribute type="AT_END_TIME">14.02.2000 13:22:57</attribute>
</node>

<node id="Processor" type="OT_ORG">
<attribute type="AT_OBJNAME">Ms. Smith</attribute>
</node>

<node id="End" type="OT_EVT">
<attribute type="AT_OBJNAME_INTERN">LIEFERUNG_ANZU</attribute>
<attribute type="AT_OBJNAME">Delivery to be created</attribute>
<attribute type="AT_ID">XMLGraph-Job-Evt2</attribute>
</node>

<edge type="CXN_FOLLOWS" source="Start" target="Function" />
<edge type="CXN_FOLLOWS" source="Function" target="End" />
<relationtype name="REL_CARRY_OUT">
<relation source="Processor" target="Function">
<attribute type="AT_KI_PK_R">7.5 EUR</attribute>
<attribute type="AT_KI_RNUM">1</attribute>
</relation>
</relationtype>
</graph>
</graphlist>
```

When imported into PPM, the structure of XML files in graph format is verified against the following DTD:



3.1.1. Object types

The table below shows all object types used in PPM graph format:

Object type	Identifier	Description
Function	OT_FUNC	Functions describe activities in the process. Function attributes are used to apply actual values for measure calculations.
Event	OT_EVT	Events are process statuses and describe the status triggering a function and the result of executing a function. Process fragments are merged into process instances using events of the same type.
Organizational unit (optional)	OT_ORG	Processors of a function can be assigned to organizational units by being made anonymous. The attributes of the organizational units are the basis for process cost accounting.
AND rule (optional)	OT_RULEAND	Splits or consolidates a process flow. The two process paths that follow the AND rule are both run through.
OR rule (optional)	OT_RULEOR	Splits or consolidates a process flow. At least one of the process paths that follow the OR rule is run through.
XOR rule (optional)	OT_RULEXOR	Splits or consolidates a process flow. Only one of the process paths that follow the XOR rule is run through.

Within a graph, an object is uniquely identified by its **node id**. The **node id** is found in the **id** XML attribute for the **node** XML element. Objects with the same **node id** are not allowed and are combined into a single object.

3.1.2. Connections

The table below shows all connections allowed in PPM:

Connection type	Identifier	Description
Flow connection	CNX_FOLLOWS	Links objects forming the structure of the process (events, functions, rules) in the graph.
Comment connection	CXN_UNDIRECTED	Assigns an organizational unit to a function that it executes.

3.1.3. Relations (optional)

The table below shows all relation types allowed in PPM:

Relation type	Identifier	Description
executes	REL_CARRY_OUT	Creates a relation between a function and the executing organizational unit.
cooperates with (without gaps)	REL_WORKS_ TOGETHER	Creates a relation between organizational units as the source reference object and organizational units as the target reference object. A relation calculation is only performed between function instances that directly follow one another and at which organizational units are specified. There must not be any other function instances without organizational units between them.
cooperates with (with gaps)	REL_WORKS_ TOGETHER_LONG_DISTANCE	Creates a relation between organizational units as the source reference object and organizational units as the target reference object. Between the function instances at which the organizational units are specified additional function instances without organizational units may occur. These are ignored in the relation calculation, that is, the function instances with the organizational units do not have to follow one another directly.
Ping pong	REL_PING_PONG	Creates a relation between organizational units as the source reference object and organizational units as the target reference object.

The actual relation occurrences have the measure attributes that are used as a basis for calculation of the relation measures defined in the measure configuration – if necessary, for each individual relation type.

For details of how relations are defined and calculated, refer to the **PPM Customizing** Technical Reference.

3.1.4. Guidelines for the graph structure

The general EPC conventions (sequence in connection flow **Event - Function - Event**) result in the following guidelines for creating the graph for a process instance:

- A process instance must begin and end with one or more events and contain at least one function.
- As rules, process instances can contain exclusively AND rules, as they represent actual working processes.
- An event may not be followed by a branching rule (OR or XOR).
- A function may be followed by another function. The linking event between two functions may be removed.
- An event may only be followed by a function or a joining rule.

3.1.5. Attributes

The following attributes must be specified for graphs of a process instance and its objects:

Process attributes

Attribute name	Identifier	Data type	Description
Process ID (optional)	AT_EPK_KEY	TEXT	Unique identifier of the process instance. The attribute value should match the graph ID (**id** tag of the **<graph>** XML element). This attribute identifies an instance as completed.
Process identification	AT_ID	TEXT	Process identification. The attribute value is displayed in the process instance list and EPC view of the respective process instance. The value of the **AT_EPK_KEY** attribute is a useful attribute value for completed instances.
Process type group	AT_PROCTYPE GROUP	TEXT	Name of the process type group to which the process instance belongs. If the attribute is not specified it is created by the typifier.
Process type	AT_PROCTYPE	TEXT	Name of the process instance's process type. If the attribute is not specified it is created by the typifier.
Instance	AT_IS_PROC INSTANCE	BOOLEAN	Specifies whether it is the graph for an individual process instance (not specified or value = TRUE) or an aggregated process instance (value = FALSE).

Object attributes

Attribute name	Identifier	Data type	Description
Name	AT_OBJNAME	TEXT	Object name. Is used for EPC view.
Internal name	AT_OBJNAME_ INTERN	TEXT	Internal language-independent name of the object. Is used for referencing the object.
Start time	AT_START_ TIME	TIME STAMP	Specifies the start time for execution of a function. Is optional if the **End time** attribute is specified at a function.
End time	AT_END_TIME	TIME STAMP	Specifies the end time for execution of a function. Is optional if the **Start time** attribute is specified at a function.

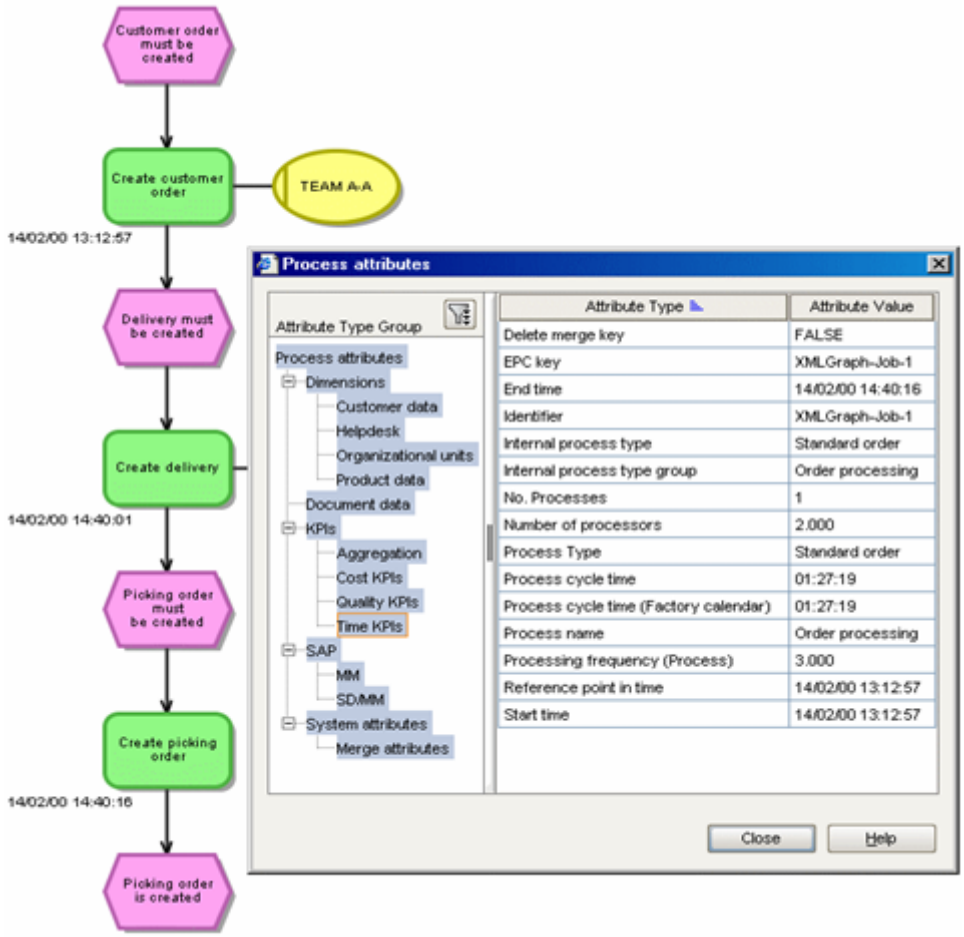
Optional object attributes

The following table gives an example of various object attributes that can be used to calculate measures.

Attribute name	Identifier	Type	Description
Number of executions	AT_COUNT_ PROCESSINGS	LONG	Specifies the frequency of execution of a function.
Performance standard	AT_LS	TIME SPAN	Is used to calculate the average duration of the execution of an instance.
Batch user	AT_IS_BATCH USER	BOOLEAN	Specifies whether an organizational unit is a batch user (program).

3.1.6. XML example graph

The illustration below shows a graph with created process instance attributes (calculated measures and process type) as an EPC view with active process attribute dialog:



The associated XML file looks like this (the lines representing objects have a colored background according to the EPC objects):

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE graphlist SYSTEM "graph.dtd">
<graphlist>
<graph id="XMLGraph-Job-1" xml:lang="en">
<attribute type="AT_EPK_KEY">XMLGraph-Job-1</attribute>
<attribute type="AT_TIME">14.2.2000 13:12:57</attribute>
<attribute type="AT_PROCTYPE">Standard order</attribute>
<attribute type="AT_PROCTYPEGROUP">Order processing</attribute>
<attribute type="AT_ID">XMLGraph-Job-1</attribute>
<node id="XMLGraph-Job-Evt1" type="OT_EVT">
<attribute type="AT_OBJNAME_INTERN">AUFTRAG_ANZU</attribute>
<attribute type="AT_OBJNAME">Customer order to be created</attribute>
</node>
<node id="XMLGraph-Job-Func1" type="OT_FUNC">
<attribute type="AT_TIME">14.2.2000 13:12:57</attribute>
<attribute type="AT_OBJNAME_INTERN">AUFTRAG</attribute>
<attribute type="AT_OBJNAME">Create customer order</attribute>
<attribute type="AT_END_TIME">14.02.2000 01:12:57 PM</attribute>
</node>
<node id="HDMXMLGraph-Job-Func1" type="OT_ORG">
<attribute type="AT_OBJNAME">HDM</attribute>
</node>
```

```
<node id="XMLGraph-Job-Evt2" type="OT_EVT">
<attribute type="AT_OBJNAME_INTERN">LIEFERUNG_ANZU</attribute>
<attribute type="AT_OBJNAME">Delivery to be created</attribute>
<attribute type="AT_ID">XMLGraph-Job-Evt2</attribute>
</node>
<node id="XMLGraph-Job-Func2" type="OT_FUNC">
<attribute type="AT_TIME">14.2.2000 02:40:01 PM</attribute>
<attribute type="AT_OBJNAME_INTERN">LIEFERUNG</attribute>
<attribute type="AT_OBJNAME">Create delivery</attribute>
<attribute type="AT_END_TIME">14.02.2000 02:40:01 PM</attribute>
</node>
<node id="HDMXMLGraph-Job-Func2" type="OT_ORG">
<attribute type="AT_OBJNAME">HDM</attribute>
</node>
<node id="XMLGraph-Job-Evt3" type="OT_EVT">
<attribute type="AT_OBJNAME_INTERN">KOM_AUFTRAG_ANZU</attribute>
<attribute type="AT_OBJNAME">Pick order must be
created</attribute>
<attribute type="AT_ID">XMLGraph-Job-Evt3</attribute>
</node>
<node id="XMLGraph-Job-Func3" type="OT_FUNC">
<attribute type="AT_TIME">14.2.2000 02:40:16 PM</attribute>
<attribute type="AT_OBJNAME_INTERN">KOM_AUFTRAG</attribute>
<attribute type="AT_OBJNAME">Create pick order</attribute>
<attribute type="AT_END_TIME">14.02.2000 02:40:16 PM</attribute>
</node>
<node id="XMLGraph-Job-Evt4" type="OT_EVT">
<attribute type="AT_OBJNAME_INTERN">KOM_AUFTRAG_ALGT</attribute>
<attribute type="AT_OBJNAME">Pick order created</attribute>
</node>
<edge type="CXN_FOLLOWS" source="XMLGraph-Job-Evt1"
target="XMLGraph-Job-Func1"/>
<edge type="CXN_FOLLOWS" source="XMLGraph-Job-Func1"
target="XMLGraph-Job-Evt2" />
<edge type="CXN_UNDIRECTED" source="HDMXMLGraph-Job-Func1"
target="XMLGraph-Job-Func1">
<attribute type="AT_COUNT_PROCESSINGS">1</attribute>
</edge>
<edge type="CXN_FOLLOWS" source="XMLGraph-Job-Func2"
target="XMLGraph-Job-Evt3" />
<edge type="CXN_UNDIRECTED" source="HDMXMLGraph-Job-Func2"
target="XMLGraph-Job-Func2">
<attribute type="AT_COUNT_PROCESSINGS">1</attribute>
</edge>
<edge type="CXN_FOLLOWS" source="XMLGraph-Job-Func3"
target="XMLGraph-Job-Evt4" />
<edge type="CXN_FOLLOWS" source="XMLGraph-Job-Evt2"
target="XMLGraph-Job-Func2" />
<edge type="CXN_FOLLOWS" source="XMLGraph-Job-Evt3"
target="XMLGraph-Job-Func3" />
</graph>
</graphlist>
```

3.2. System event format

Many source systems log the processing of an operation by documenting status changes or particular system statuses. These are saved in the form of system events (source system events) with supplementary information. A typical example is a system for order processing, in which the creation of a new order or the invoicing of an order are saved. A further example is the SAP R/3 SD module, which documents the statuses and progress of an order in individual transactions.

Importing process instance fragments in PPM system event format has three stages:

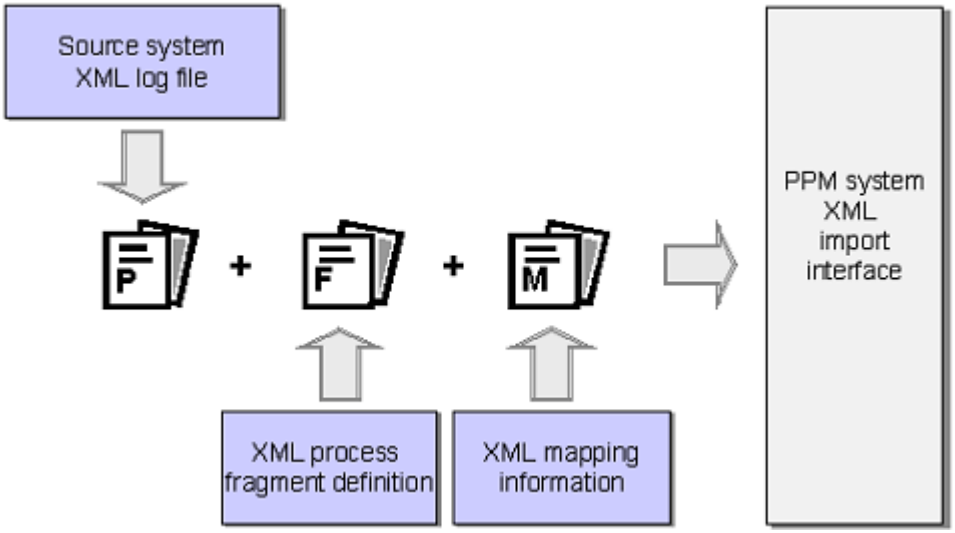
1. Define process fragments and mapping

   All system event types occurring in the source system data, which are to be imported to PPM (for example, Create order, Invoice order), must first be created as process fragment definitions in the form of an EPC in the fragment file. In addition, the mapping file must specify which system event types are to be assigned to which fragment definitions when imported and which attributes are to be transferred to PPM. In the mapping file, you must take into account attributes, which allow the calculation of the measures and the merging of process instance fragments into a process instance (for example, time stamp, sequential system event number, order number).
2. Generating a source system XML output file

   Individual system events with additional information (attributes with real data) are extracted from the application system into an XML output file according to the activity flow.
3. Generate the process instance fragments

   When importing data, a search is made for the fragment definition assigned (type level) for each system event in the XML output file, and this is then copied to the PPM database. Attributes of the system event with real data (for example, execution time, processor, order and customer number) are transferred to objects in the copy of the fragment definition, generating a process instance fragment (instance level) in the PPM database that corresponds to this system event.

The illustration below highlights the process of generating process instance fragments:



3.2.1. Definition of process fragments

In order to be able to generate process instances for the PPM system from the system events, each system event to be imported to PPM must be linked to a process fragment definition. Each system event is assigned to a system event type. A fragment definition must be created for each system event type.

Each system event type is assigned to an end event in a separate EPC. It is assumed that in the process, a function must have preceded and triggered the end event. Adding a start event before the function results in a complete process fragment complying with the modeling system. The start event of a process fragment can correspond to the end event of another process fragment.

All process fragment definitions are saved in an XML file as a graph list. The fragment definition XML file uses the graph format DTD for its format description.

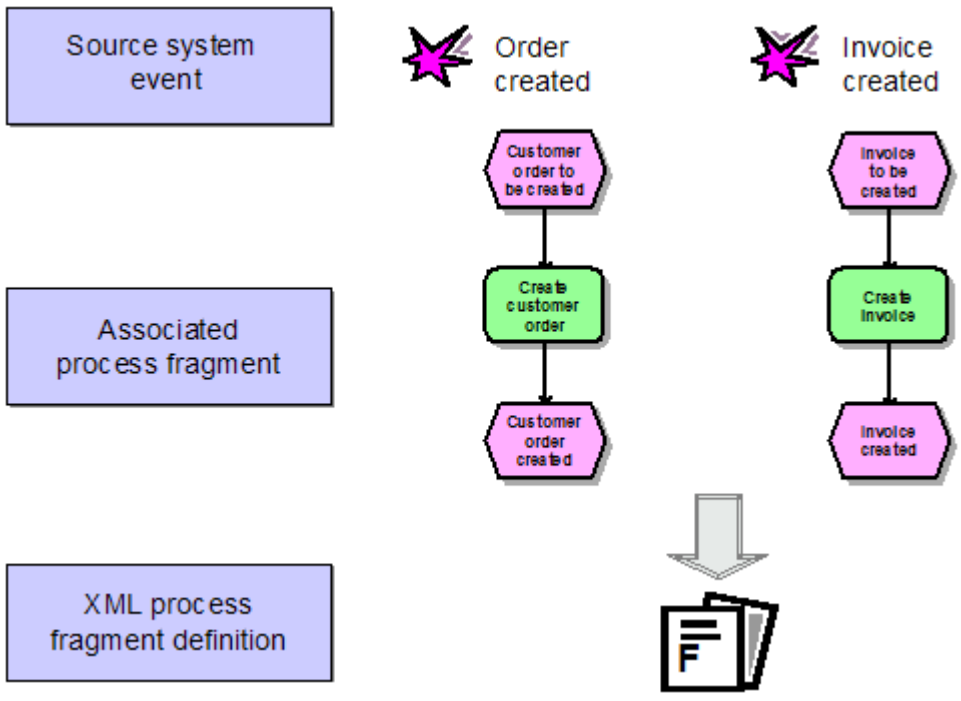
Imported process instance fragments are linked to form a process instance by way of the start and end events of the individual fragments. These events are known as merge events.

It is not absolutely necessary for the system event and the end event of the process instance fragment to be the same, for example, if the process for the fragment definition ends with two end events. What is important is that all system descriptions contained in the system event are represented by the process instance fragment created.

Example 1.

The source system contains the two system events **Order created** and **Invoice created**. Each of these is transferred to an EPC as an end event and interpreted as the result of the **Create customer order** or **Create invoice** function. You must use further knowledge of the source system to determine the events preceding these two functions. In the example, these are the **Customer order to be created** and **Invoice to be created** events.

The illustration below shows the process fragment description for the two system events.



The file extract below shows a possible fragment definition file for the process fragments illustrated:

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<!DOCTYPE graphlist SYSTEM "graph.dtd">
<graphlist>
<graph id="FRG_ORD_CREATED">
<node id="EVT_ORD_TOBECREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">Customer order to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_ORD_TOBECREATED</attribute>
</node>
<node id="FCT_CREATE_ORDER" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create customer order</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_CREATE_ORDER</attribute>
</node>
<node id="EVT_ORD_CREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">Customer order created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_ORD_CREATED</attribute>
</node>
<edge type="CXN_FOLLOWS" source="EVT_ORD_TOBECREATED"
target="FCT_CREATE_ORDER"/>
<edge type="CXN_FOLLOWS" source="FCT_CREATE_ORDER"

  target="EVT_ORD_CREATED"/>
</graph>
<graph id="FRG_INVOICED">
<node id="EVT_TOBE_INVOICED" type="OT_EVT">
<attribute type="AT_OBJNAME">Invoice to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_TOBE_INVOICED</attribute>
</node>
<node id="FCT_INVOICE" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create invoice</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_INVOICE</attribute>
</node>
<node id="EVT_INVOICED" type="OT_EVT">
<attribute type="AT_OBJNAME">Invoice created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_INVOICED</attribute>
</node>
<edge type="CXN_FOLLOWS" source="EVT_TOBE_INVOICED"
target="FCT_INVOICE"/>
<edge type="CXN_FOLLOWS" source="FCT_INVOICE"
target="EVT_INVOICED"/>
</graph>
</graphlist>
```

Fragment definition graphs should not contain any attributes specified. When the process instance fragments are subsequently merged, only object attributes are taken into account by default. Process instance attributes to be retained when merging can be specified in the merge configuration.

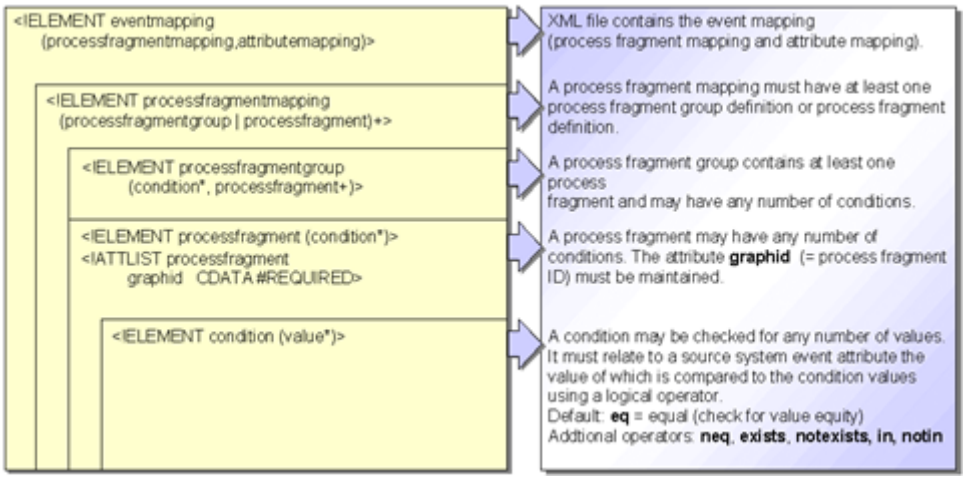
3.2.2. Definition of mapping

The mapping file contains the assignment of the system event types to process fragment definitions and determines the attributes of the system events, which are copied to the process fragments for which an instance is created in the PPM system.

3.2.2.1. Definition of process fragment mapping

Process fragment mapping defines which process fragment definitions are used to instantiate the system event types. It can be controlled by any number of conditions (**condition** XML element) linked to one another by AND rules.

The rules for the structure of process fragment mapping in the XML mapping file are specified in the following extract from the file **eventmapping.dtd**:




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE eventmapping SYSTEM "eventmapping.dtd">
<eventmapping>
<processfragmentmapping>
<processfragment graphid="FRG_ORD_CREATED">
<condition eventattributetype="AUFTR_TYP">
<value>C</value>
</condition>
<condition eventattributetype="MAT_NR" logicaloperator="in">
<value>123456</value>
<value>56789</value>
<value>78901</value>
</condition>
</processfragment>
...
</processfragmentmapping>
<attributemapping>
...
</attributemapping>
</eventmapping>
```

The **FRG_ORD_CREATED** process fragment is created if the following two conditions are met:

- The relevant system event represents an order document (**AUFTR_TYP** attribute has the value **C**).
- The value of the **MAT_NR** system event attribute corresponds to one of the specified material numbers.

A process fragment definition must not contain a condition. In this case, the same specified fragment definition is used for each imported system event. The objects in the instanced fragment are specified in the subsequent attribute mapping.

Definition of a process fragment group

Process fragment definitions can be summarized into groups. This results in the following advantages:

- Performance increase
- Simplified creation of process fragment mapping definitions
- Improved clarity

The two process fragment definitions below are contained in an XML mapping file:

```
<processfragment graphid="AUFTRAG_ANLEGEN">
<condition eventattributetype="AUFTR_TYP">
<value>C</value>
</condition>
<condition eventattributetype="CHARGEN_PFL" logicaloperator="neq">
<value>X</value>
</condition>
</processfragment>
<processfragment graphid="CHPLICHT_AUFTRAG_ANLEGEN">
<condition eventattributetype="AUFTR_TYP">
<value>C</value>
</condition>
<condition eventattributetype="CHARGEN_PFL">
<value>X</value>
</condition>
</processfragment>
```

Example 2.

The first process fragment **AUFTRAG_ANLEGEN** is created for a system event of the **Order document** type (attribute value of **AUFTR_TYP** is **C**), which is not subject to management in batches (attribute value of **CHARGEN_PFL** does not equal **X**).

The second process fragment **CHPFLICH_AUFTRAG_ANLEGEN** is created for a system event of the **Order document subject to management in batches** type (attribute value of **AUFTR_TYP** is **C** and attribute value of **CHARGEN_PFL** is **X**).

The two process fragments shown can be summarized in a process fragment group:

```
<processfragmentgroup>
<condition eventattributetype="AUFTR_TYP">
<value>C</value>
</condition>

<processfragment graphid="AUFTRAG_ANLEGEN">
<condition eventattributetype="CHARGEN_PFL" logicaloperator="neq">
<value>X</value>
</condition>
</processfragment>

<processfragment graphid="CHPLICHT_AUFTRAG_ANLEGEN">
<condition eventattributetype="CHARGEN_PFL">
<value>X</value>
```

```
</condition>
</processfragment>

</processfragmentgroup>
```

Summary into a process fragment group means that only one check is made as to whether the **AUFTR_TYP** source system attribute has the value **C** when importing. If this is not the case, neither of the two process fragments in the process fragment group is instantiated.

Use process fragment groups to improve the clarity of process fragment mapping definitions and the performance of the import process.

Suppress output of warnings

If you do not want to import certain system events in your customizing and if you have not defined any process mapping for these system events you can suppress the error message to be expected when importing. To do this, you need to specify conditions in the **ignoreevent** XML element, which suppress the output of an error message relating to particular fragments when these cannot be imported.

For system events specified with **ignoreevent** mapping, error message output is suppressed only if the system event cannot be imported. This means that when you import system events having both process fragment mapping and **ignoreevent** mapping, these system events are imported.

Example 3.

You want to import process fragments if the **EKKO_BSTYP** system event attribute exists and the **MSEG_SHKZG** system event attribute has the value **S** for "Post goods receipt" or **H** for "Cancel goods receipt". Other values of the **MSEG_SHKZG** attribute result in a warning being output.

If the **EKKO_BSTYP** system event attribute does not exist, no process fragments are imported and no warning is output.

The following process mapping meets the above requirements:

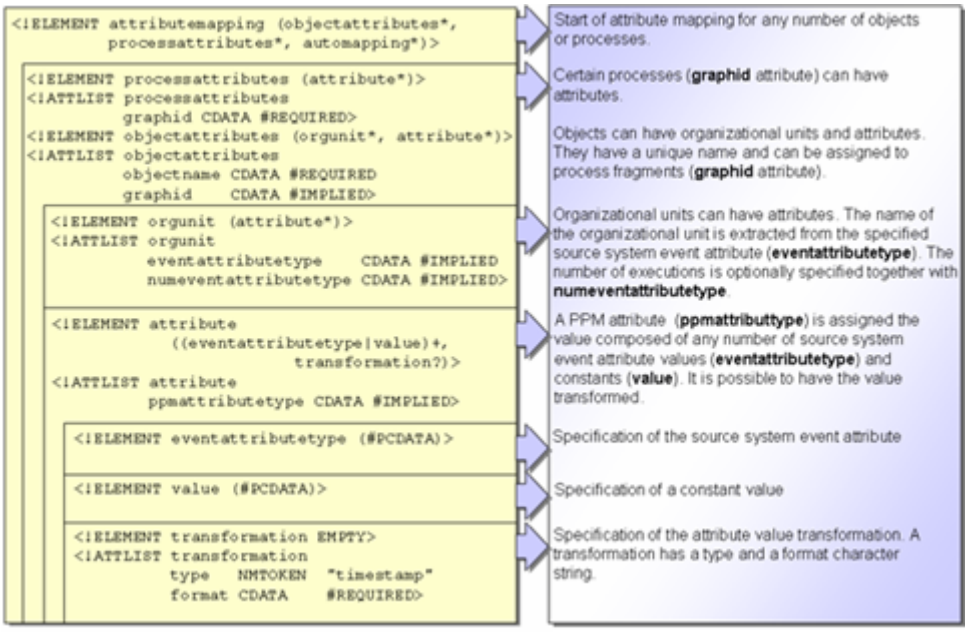
```
...
<processfragmentgroup>
<!-- Import goods receipts with MM predecessor documents only
-->
<condition eventattributetype="EKKO-BSTYP" logicaloperator="exists"/>
<processfragment graphid="GWEOF">
<!-- Post goods receipt
-->
<condition eventattributetype="MSEG-SHKZG" logicaloperator="eq">
<value>S</value>
</condition>
</processfragment>

<processfragment graphid="GWSOF">
<!-- Cancel goods receipt
-->
<condition eventattributetype="MSEG-SHKZG" logicaloperator="eq">
<value>H</value>
```

```
</condition>
</processfragment>
</processfragmentgroup>

<ignoreevent>
<!-- Do not output warning if no MM predecessor document exists
because these system events are not to be imported
-->
<condition eventattributetype="EKKO-BSTYP" logicaloperator="notexists"/>
</ignoreevent>
...
```

3.2.2.2. Definition of attribute mapping

This chapter describes the configuration of attribute mapping. Attribute mapping copies source system attributes to object and process attributes of the fragment instance (PPM attributes).

The rules for the structure of attribute mapping in the XML mapping file are specified in the following extract from the file **eventmapping.dtd**:



PPM attribute values can be made up of any combination of attribute values of the system event and unalterable texts. The optional specification of the internal PPM attribute name allows source system attributes to be copied to any PPM attributes. PPM attributes can be assigned constant values (**value** XML element).

```
...
<attribute>

  <eventattributetype>MATERIAL_CLASS</eventattributetype>

  </attribute>
...
```

Example 4.

The value of the **MATERIAL_CLASS** system event attribute is copied to the PPM attribute of the same name but including the prefix (for example, **AT_**) specified in the **attributeprefix** XML attribute of the data source, that is, **AT_MATERIAL_CLASS**.

```
...
<attribute ppmattributetype="AT_MATERIAL">

  <eventattributetype>MAT_NR</eventattributetype>

  </attribute>
...
```

Example 5.

The value of the **MAT_NR** system event attribute is copied to the **AT_MATERIAL** PPM attribute.

```
...
<attribute ppmattributetype="AT_IS_SHARED_FUNCTION">

  <value>TRUE</value>

  </attribute>
...
```

Example 6.

The **AT_IS_SHARED_FUNCTION** PPM attribute is assigned the constant **TRUE**.

```
...
<attribute ppmattributetype="AT_ID">

  <eventattributetype>AUFTRAGS_SYSTEM</eventattributetype>

  <value>-</value>

  <eventattributetype>SYSTEM_NR</eventattributetype>

  <value>#</value>

  <eventattributetype>AUFTRAGS_NR</eventattributetype>

  <value>-</value>

  <eventattributetype>POSITIONS_NR</eventattributetype>

  <value>-</value>

  <eventattributetype>AUFTRAGS_TYP</eventattributetype>

  </attribute>
...
```

Example 7.

The **AT_ID** PPM attribute is assigned the value **XYZ-401#4711-10-C**.

> **Warning**
> If a specified source system attribute is not available when importing data from the XML output file, the corresponding PPM attribute is not created.

Assign attributes to objects

Created PPM attributes are assigned to particular objects in the fragment instance using the **objectattributes** XML element. The **objname** XML attribute specifies the identifier of the relevant object (**AT_OBJNAME_INTERN** object attribute). Optionally, the object specification can be refined by specifying the graph ID (**graphid** XML attribute).

Example 8.

The example below copies the value of the **END_TIME** source system attribute to the **AT_END_TIME** PPM object attribute for the function with the identifier **SAP.AUFT_ANLEG** in the **AUFTRAG_ANLEGEN** fragment instance (graph ID of the fragment definition):

```
...
<attributemapping>
<objectattributes objectname="SAP.AUFT_ANLEG" graphid="AUFTRAG_ANLEGEN">
<attribute ppmattributetype="AT_END_TIME">
<eventattributetype>END_TIME</eventattributetype>
</attribute>
</objectattributes>
</attributemapping>
...
```

Assign attributes to processes

Created PPM attributes are assigned to particular fragment instances using the **processattributes** XML element. It is mandatory to specify the ID of the fragment definition graph in the **graphid** XML attribute.

To ensure that attributes transferred directly to the process are retained when merging the fragment instances, you must extend the merge configuration accordingly. The calculation effort required to compare the process attributes results in a slight loss of performance.

The example below copies the value of the **PROCESSNAME** source system attribute to the **AT_PROCTYPE** PPM process attribute for the **AUFTRAG_ANLEGEN** fragment instance (graph ID of the fragment definition):

```
...
<attributemapping>
<processattributes graphid="AUFTRAG_ANLEGEN">
<attribute ppmattributetype="AT_PROCTYPE">
<eventattributetype>PROCESSNAME</eventattributetype>
</attribute>
</processattributes>
</attributemapping>
...
```

3.2.2.2.1. Attribute transformations

If the format of an attribute value for a system event does not match the format required by the PPM system, the value written in the PPM attribute must be transformed.

3.2.2.2.2. Time stamp transformations

PPM provides various attribute transformations of the type Time stamp transformation.

The time stamp transformations available in the PPM system convert source system time stamp values in any format to the valid internal PPM format with the proper PPM target data type. The following table provides you with an overview of available time stamp transformations:

Transformation	PPM target data type
timestamp	TIME
timestamp_epoch	TIME
timeofday	TIMEOFDAY
timeofday_epoch	TIMEOFDAY
day	DAY
day_epoch	DAY
SAGDateTime	TIME

timestamp, timeofday, day time stamp transformations

The standards for the configuration of time stamp transformations are specified in the **eventmapping.dtd** document type definition:

```
...
<!ELEMENT transformation EMPTY>
<!ATTLIST transformation
type NMTOKEN "timestamp"
format CDATA #REQUIRED
>
...
```

In the **type** XML attribute, you specify which of the available time stamp transformations is to be used. The **timestamp** attribute transformation is the default. The **format** XML attribute specifies the time stamp format in the source system attribute.

Data in the **format** attribute corresponds to the Java time stamp format:

Symbol	Description	Data type	Example
G	Epoch identifier	Text	AD
y	Year	Figure	2003
MM	Calendar month	Figure	09
MMM	Calendar month	Text	Sep
MMMM	Calendar month	Text	September
d	Calendar day	Figure	26

Symbol	Description	Data type	Example
h	Hour, American notation (1-12)	Figure	12
H	Hour (0-23)	Figure	14
m	Minute	Figure	42
s	Second	Figure	57
S	Millisecond	Figure	978
E	Day of the week	Text	Thursday
F	Recurrence of weekday in month	Figure	2
D	Day of the year	Figure	189
w	Calendar week	Figure	27
W	Week of the month	Figure	2 (2. week of the month)
a	Time of day identifier	Text	PM
k	Hour (1-24)	Figure	24
K	Hour, American notation (0-11)	Figure	7
z	Time zone	Text	GMT
´	Escape character for text	Characters	'Example text'
´´	Simple apostrophe	Characters	'Example' 'Text'

The value **2002-12-24** (<Year>-<Month>-<Day>) for a source system attribute is transformed into PPM format using the format string yyyy-MM-dd:

```
...
<transformation type="timestamp" format="yyyy-MM-dd"/>
...
```

Example 9.

Consolidated source system attribute values:

```
...
<attribute ppmattributetype="AT_END_TIME">
<eventattributetype>ERF_DAT</eventattributetype>
<eventattributetype>ERF_ZEIT</eventattributetype>
<transformation format="yyyyMMddHHmmss"/>
</attribute>
...
```

Example 10.

In the **ERF_DAT** source system attribute, the creation date is present in the format **yyyyMMdd** and in the **ERF_ZEIT** source system attribute the creation time is present in the format **HHmmss**. Extract from the XML output file:

```
<event>
...
<attribute type='ERF_DAT'>20011230</attribute>
...
```

```
</event>
<event>
...
<attribute type='ERF_ZEIT'>120730</attribute>
<attribute type='ERF_DAT'>20011101</attribute>
...
</event>
```

Example 11.

For the first system event, the **AT_END_TIME** attribute is not created, as the **ERF_ZEIT** attribute is not present.

For the second system event, both attributes are present. The attribute values are joined in the specified sequence to give **20011101120730**, then evaluated using the specified format **yyyyMMddHHmmss** and transformed into the PPM-compatible time stamp **01.11.2001 12:07:30**.

In the **ERF_STD** attribute only the hour is recorded in which the system event was created. The creation time is always thirty minutes after the hour.

```
...
<attribute ppmattributetype="AT_END_TIME">
<eventattributetype>ERF_DAT</eventattributetype>
<value>::</value>
<eventattributetype>ERF_STD</eventattributetype>
<value>30</value>
<transformation format="yyyyMMdd::HHmm"/>
</attribute>
...
```

Example 12.

Associated system event from output file:

```
...
<event>
...
<attribute type='ERF_STD'>12</attribute>
<attribute type='ERF_DAT'>20011001</attribute>
...
</event>
...
```

Example 13.

The attribute values and constant strings are combined in the specified sequence into the string **20011001::1230** and transformed using the format **yyyyMMdd::hhmm** to give the PPM compatible time stamp **01.10.2001 12:30:00**, which is written to the PPM attribute **AT_END_TIME**.

timestamp_epoch, timeofday_epoch, and day_epoch time stamp transformations

The **timestamp_epoch** time stamp transformation transforms an integer value indicating the seconds or milliseconds that have passed since January 1, 1970 into the internal PPM format. In the **format** XML

attribute, you specify whether the integer value indicates the number of seconds (**SECOND**) or milliseconds (**MILLISECOND**) that have passed since 01.01.1970 0:00:00 GMT. The system's current time zone is taken into account in the calculation.

Use the attribute transformations **timeofday_epoch** and **day_epoch** the same way.

Example 14.

In the **WORK_ITEM-END_TIME** source system attribute, the number of seconds since January 1, 1970 is indicated.

```
...
<attribute type="WORK_ITEM-END_TIME">1221482578</attribute>
...
```

Example 15.

Use the following mapping rule to assign this attribute value to the value of the PPM **AT_END_TIME** attribute:

```
...
<attribute ppmattributetype="AT_END_TIME">
<eventattributetype>

  WORK_ITEM-END_TIME
</eventattributetype>
<transformation type="timestamp_epoch"
format="SECOND"/>
</attribute>
...
```

3.2.2.2.3. Floating point number transformation

For the PPM data type **DOUBLE**, PPM expects values without a thousands separator and with a period as a decimal separator. If the source system attribute value does not match this format it needs to be transformed accordingly.

When importing floating point numbers, you can specify in the **format** attribute the decimal and thousands separators to be used for parsing a double value if you are using the **double** attribute transformation.

<transformation type="double" format=","/>

<transformation type="double" format="."/>

If only one character is specified, it must be the decimal separator. If two characters are specified, the first one must be the thousands separator and the second the decimal separator.

Example 16.

Attribute values with thousands separator (,) and decimal separator (.)

1.000,00

2.324.213,42

Definition of attribute elements

<attribute ppmattributetype="AT_END_TIME">

<eventattributetype>TIMESTAMP_FIELD</eventattributetype>

<transformation type="double" format="."/>

</attribute>

3.2.2.3. Organizational units

Organizational units are created dynamically for functions in the fragment instance.

Define an organizational unit

The **orgunit** XML element in the mapping file defines an organizational unit. The value of the **eventattributetype** XML attribute specifies the name of the source system attribute, which contains the name of the organizational unit. This name is assigned to the **AT_OBJNAME** and **AT_OBJNAME_INTERN** object attributes for the organizational unit.

The following extract from the mapping file creates two organizational units, whose names are extracted from the **PROCESSOR_1** and **PROCESSOR_2** source system attributes. The number of executions of the functions is read from the **NUM_OF_PROCESSINGS_1** and **NUM_OF_PROCESSINGS_2** source system attributes specified by the **numeventattributtype** XML attribute.

```
...
<attributmapping>
...
<orgunit eventattributetype="PROCESSOR_1"
numeventattributetype="NUM_OF_PROCESSINGS_1"/>
<orgunit eventattributetype="PROCESSOR_2"
numeventattributetype="NUM_OF_PROCESSINGS_2"/>
...
</attributmapping>
...
```

Appropriate attribute mapping allows additional attributes to be created for organizational units or existing attributes to be overwritten.

The file extract below assigns the value **anonymous processor** to the name of the organizational unit displayed in the PPM user interface by overwriting the **AT_OBJNAME** object attribute regardless of the source system attribute value.

```
...
<attributmapping>
...
<orgunit eventattributetype="PROCESSOR_1">
<attribute ppmattributetype="AT_OBJNAME">
<value>anonymous processor</value>
</attribute>
</orgunit>
...
</attributmapping>
...
```

Assign organizational units

Organizational units are assigned to particular functions in the fragment instance using the **objectattributes** XML element. The **objectname** XML attribute specifies the identifier of the relevant function (**AT_OBJNAME_INTERN** object attribute). Optionally, the object specification can be refined by specifying the graph ID (**graphid** XML attribute).

The example below creates an organizational unit for the function with the identifier **SAP.AUFT_ANLEG** from the **AUFTRAG_ANLEGEN** fragment instance (graph ID of fragment definition) from the **VBAP-ERNAM** source system attribute:

```
<attributemapping>
<objectattributes objectname="SAP.AUFT_ANLEG"
graphid="AUFTRAG_ANLEGEN">
<orgunit eventattributetype="VBAP-ERNAM"/>
</objectattributes>
</attributemapping>
```

Organizational units and aggregation

During the temporary aggregation process, the organizational units created during anonymization are retained, whereas users are deleted.

To transfer a real user without anonymization, you have to create the **AT_ISUSERGROUP** attribute with the value **TRUE** in the attribute mapping for the corresponding organizational unit.

In the example below, an organizational unit whose content is extracted from the **UNIT_GROUP_LABEL** source system attribute is created for all functions with the internal object name **FCT_A** in the process instance fragment with the graph ID **FRG_B**.

```
<eventmapping>
<processfragmentmapping>
...
</processfragmentmapping>
<attributemapping>
...
<objectattributes objectname="FCT_A" graphid="FRG_B">
<orgunit eventattributetype="UNIT_GROUP_LABEL">
<attribute ppmattributetype="AT_ISUSERGROUP">
<value>TRUE</value>
</attribute>
</orgunit>
...
</objectattributes>
...
</attributemapping>
</eventmapping>
```

3.2.2.4. Special case of attribute mapping

If several identical objects exist within a fragment definition graph (identical attribute value for **AT_OBJNAME_INTERN**), you need to specify a unique **AT_NODE_ID** object attribute in the fragment definition to differentiate between the objects. In the attribute mapping file, instead of the value of the **AT_OBJNAME_INTERN** object attribute you should specify the value of the **AT_NODE_ID** object attribute.

Example 17.

Fragment definition:

The fragment definition contains two **Order is created** end events.

```
...
<graph id="CHPFLICHT_AUFTRAG_ANLEGEN">
<node id="4" type="OT_EVT">
<attribute type="AT_OBJNAME">Order to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">SAP.AUFT_ANZU</attribute>
</node>
<node id="5" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create order</attribute>
<attribute type="AT_OBJNAME_INTERN">SAP.AUFT_ANLEG</attribute>
</node>
<node id="42" type="OT_RULEOR"/>
<node id="6" type="OT_EVT">
<attribute type="AT_OBJNAME">Order is created</attribute>
<attribute type="AT_OBJNAME_INTERN">SAP.AUFT_ANGELEGT</attribute>
<attribute type="AT_NODE_ID">End event 1</attribute>
</node>
<node id="7" type="OT_EVT">
<attribute type="AT_OBJNAME">Order is created</attribute>
<attribute type="AT_OBJNAME_INTERN">SAP.AUFT_ANGELEGT</attribute>
<attribute type="AT_NODE_ID">End event 2</attribute>
</node>
<edge type="CXN_FOLLOWS" source="4" target="5"/>
<edge type="CXN_FOLLOWS" source="5" target="42"/>
<edge type="CXN_FOLLOWS" source="42" target="6"/>
<edge type="CXN_FOLLOWS" source="42" target="7"/>
</graph>
...
```

Example 18.

Attribute mapping:

At the two **Order is created** end events, attribute mapping generates a PPM attribute **AT_ID** with a different value:

```
...
<attributemapping>
<objectattributes objectname="End event 1" graphid="AUFTRAG_ANLEGEN">
<attribute ppmattributetype="AT_ID">
<value>This is the first end event</value>
</attribute>
</objectattributes>
<objectattributes objectname="End event 2" graphid="AUFTRAG_ANLEGEN">
<attribute ppmattributetype="AT_ID">
<value>This is the second end event</value>
</attribute>
</objectattributes>
</attributemapping>
```

```
...
```

3.2.3. Create fragment definitions in ARIS

To correctly import the data, is does not matter how the fragment definition file and the mapping file have been created. Creating the fragment definition file with ARIS or the mapping file with PPM Customizing Toolkit is easier than directly editing the XML files.

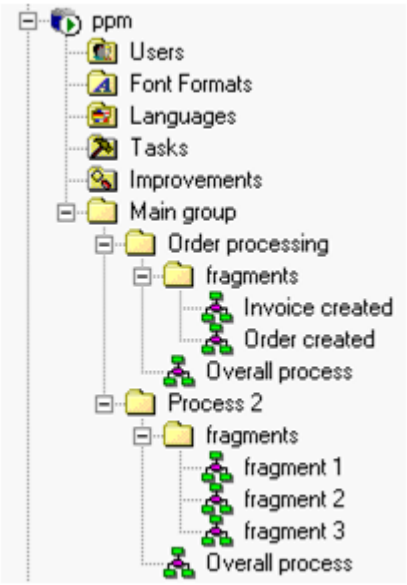
The process fragment definitions are modeled as EPCs. A special ARIS method filter is available for modeling. Often, the overall process to be analyzed by PPM is already modeled using ARIS and can be used as a basis for the creating the fragments.

The EPCs for the process fragment definitions can also be modeled directly. However, in practice the creation of an overall process has proved to be more effective for reasons of clarity.

3.2.3.1. Modeling the overall process

First of all, the entire process to be monitored by PPM is modeled as an EPC in the application system using the real activity flow. When creating the process, ensure that all system events occurring are modeled in the EPC as events of a preceding function. Organizational units should be specified at the functions.

For each overall process, create a separate group with a sub-group for the process fragment definitions. Model an EPC for the overall process and a further EPC for each fragment in the sub-group.



Modeling guidelines for the overall process

- A process begins with an event and ends with at least one event.
- Events may not precede branching OR and XOR rules.
- Specify the ARIS attributes **Name** and **Identifier** for all events and functions. When the ARIS report is executed, the name is transferred to the fragment definition as the language-specific PPM attribute **AT_OBJNAME**, and the identifier as the language-independent PPM attribute **AT_OBJNAME_INTERN**.

3.2.3.2. Modeling the process fragment definitions

First of all, generate a model assignment for all functions of the overall process in the group, which is to contain the fragment definition. The identifier of the assigned model (process fragment definition) specifies the name of the fragment definition. Next copy all events, which adjoin a function in the procedural logic, into the model which is assigned to the function.

Observe the following guidelines when modeling the fragments, regardless of whether you are using an overall process or creating the fragments without an overall process.

Modeling guidelines for fragment definition

Use the special PPM method filter **FragmentXML_ARISToolSet_Filter.amc** from the directory **<PPM installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ARIS**. This limits the number of modeling elements to the permissible object and connection types and thus makes modeling easier.

- Each process fragment must be modeled in a separate model of the **EPC** type. The identifier of the model is transferred to the XML fragment definition file as the **graph id** and specifies the name of the process fragment.

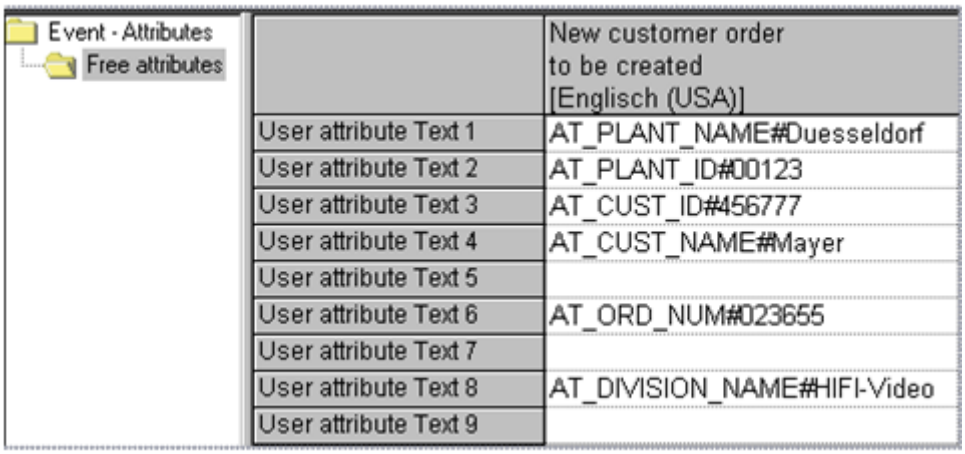
If you are using an overall process, when modeling the fragments generate occurrence copies of the objects in the overall process.

- Avoid cyclic process paths (loops). A fragment definition that contains cycles is identified and ignored by the model report and a corresponding message is displayed in the ARIS output window.
- All objects (events, functions, rules) in the fragment model must have a unique identifier. This is transferred to PPM as the language-independent attribute **AT_OBJNAME_INTERN**.
- Organizational units do not need to be modeled. They are ignored when generating the fragment definition file using ARIS Report. Organizational units are dynamically assigned to the corresponding functions by the attribute mapping (**orgunit** XML element - see Organizational units [27]).

You can also specify **free attributes** for all objects and connections. These are written to the corresponding objects or connections in the fragment definition file as fixed PPM attribute/value combinations of the **TEXT** type. The list must be specified in ascending order starting with **User attribute text 1** and is evaluated as far as the first non-specified attribute. Format: **<Attribute key>#<Attribute value>**. The separator **#** must not appear in the attribute key or in the attribute value.

Example 19.

For the **New customer order to be created** event, the following free attributes are specified:



	New customer order to be created [Englisch (USA)]
User attribute Text 1	AT_PLANT_NAME#Duesseldorf
User attribute Text 2	AT_PLANT_ID#00123
User attribute Text 3	AT_CUST_ID#456777
User attribute Text 4	AT_CUST_NAME#Mayer
User attribute Text 5	
User attribute Text 6	AT_ORD_NUM#023655
User attribute Text 7	
User attribute Text 8	AT_DIVISION_NAME#HIFI-Video
User attribute Text 9	

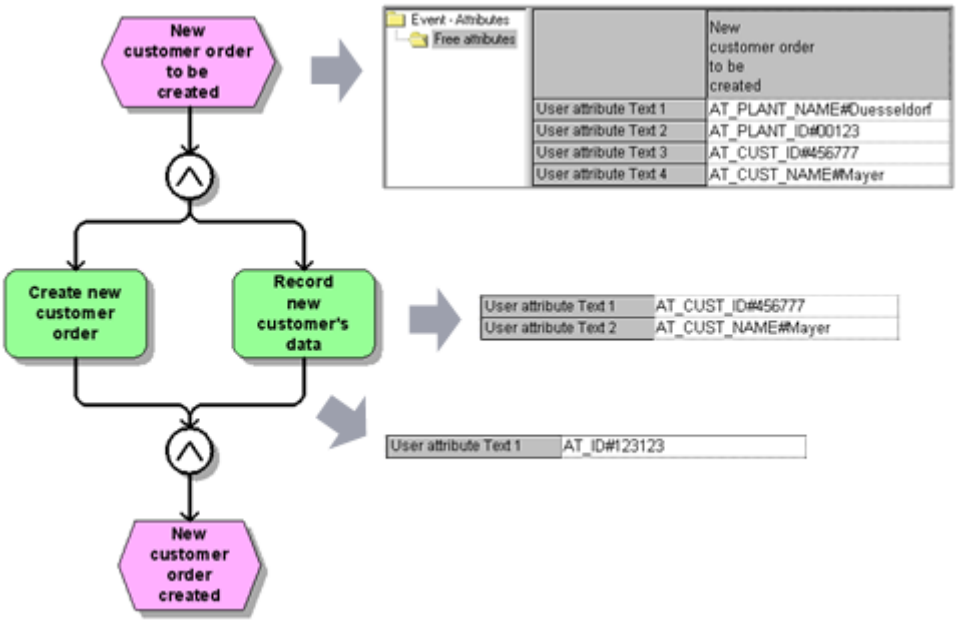
In the file extract below from the fragment definition file created, the transferred free attributes are shown in **bold**.

```
...
<node id="EVT_NEWCUSTORD2BE_CREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">New customer order to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_NEWCUSTORD2BE_CREATED</attribute>
<attribute type="AT_PLANT_NAME">Duesseldorf</attribute>
<attribute type="AT_PLANT_ID">00123</attribute>
<attribute type="AT_CUST_ID">456777</attribute>
<attribute type="AT_CUST_NAME">Mayer</attribute>
</node>
...
```

- The following ARIS connection types are included in report evaluation and transferred to the **CXN_FOLLOWS** PPM connection type:

Source object	Target object	ARIS connection type
Event	Function	**activates**
Event	Rule	**is evaluated by**
Function	Event	**creates**
Function	Rule	**leads to**
Function	Function	**is predecessor of**
Rule	Event	**leads to**
Rule	Function	activates
Rule	Rule	links

The illustration below shows a properly modeled process fragment with additional free attributes specified for objects and a connection:



ARIS Report generates the following fragment definition file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE graphlist SYSTEM "graph.dtd">
<graphlist>
<graph id="FRG_NEWCUST_ORD">
<node id="EVT_NEWCUSTORD2BE_CREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">New customer order to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_NEWCUSTORD2BE_CREATED</attribute>
<attribute type="AT_PLANT_NAME">Duesseldorf</attribute>
<attribute type="AT_PLANT_ID">00123</attribute>
<attribute type="AT_CUST_ID">456777</attribute>
<attribute type="AT_CUST_NAME">Mayer</attribute>
</node>
<node id="FCT_CREATE_NEWCUSTORD" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create new customer order</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_CREATE_NEWCUSTORD</attribute>
</node>
<node id="OT_RULE_AND" type="OT_RULEAND">
<attribute type="AT_OBJNAME_INTERN">OT_RULE_AND</attribute>
<attribute type="AT_SAP_ID">455455</attribute>
</node>
<node id="FCT_REC_NEWCUSTDAT" type="OT_FUNC">
<attribute type="AT_OBJNAME">Enter new customer data</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_REC_NEWCUSTDAT</attribute>
<attribute type="AT_CUST_ID">456777</attribute>
<attribute type="AT_CUST_NAME">Mayer</attribute>
</node>
<node id="OT_RULE_AND" type="OT_RULEAND">
<attribute type="AT_OBJNAME_INTERN">OT_RULE_AND</attribute>
</node>
<node id="EVT_NEWCUSTORD_CREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">New customer order created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_NEWCUSTORD_CREATED</attribute>
</node>
<edge type="CXN_FOLLOWS" source="FCT_REC_NEWCUSTDAT" target="OT_RULE_AND">
<attribute type="AT_ID">123123</attribute>
</edge>
<edge type="CXN_FOLLOWS" source="EVT_NEWCUSTORD2BE_CREATED"
target="OT_RULE_AND"/>
<edge type="CXN_FOLLOWS" source="OT_RULE_AND" target="FCT_REC_NEWCUSTDAT"/>
<edge type="CXN_FOLLOWS" source="OT_RULE_AND"
target="EVT_NEWCUSTORD_CREATED"/>
<edge type="CXN_FOLLOWS" source="FCT_CREATE_NEWCUSTORD"
target="OT_RULE_AND"/>
<edge type="CXN_FOLLOWS" source="OT_RULE_AND"
target="FCT_CREATE_NEWCUSTORD"/>
</graph>
</graphlist>
```

Overview of process fragment modeling guidelines

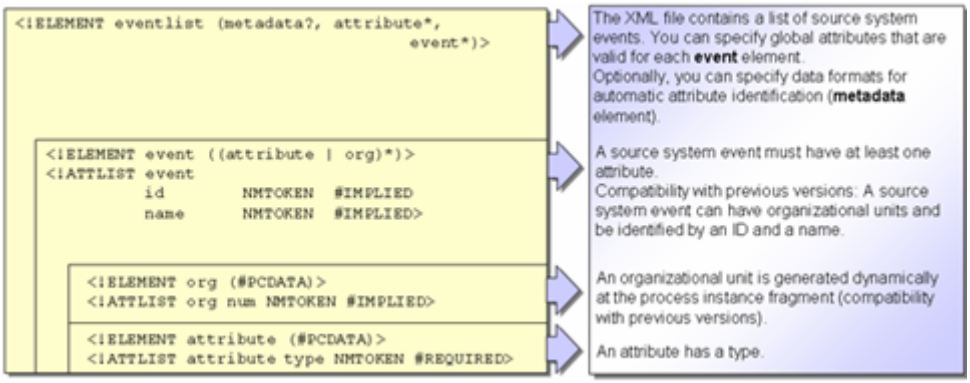
ARIS element	ARIS attribute	PPM fragment definition
EPC (fragment model)	Name	**graph id** of the definition graph created
Function	Name	AT_OBJNAME
	Identifier	AT_OBJNAME_INTERN
Event	Name	AT_OBJNAME
	Identifier	AT_OBJNAME_INTERN
Rule	Identifier	AT_OBJNAME_INTERN
Free attributes	User attribute text <x> x = Integer from 1 - 37	PPM attribute of **TEXT** type including value for objects or connections)

3.2.3.3. Format of system event file

The system event file contains the actual instance information about the actual course of the processes. An extract from the XML file for the above example could look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE eventlist SYSTEM "event.dtd">
<eventlist>
...
<event>
<attribute type="PPM_EVENT_ID">A-Nr 4711</attribute>
<attribute type="PPM_EVENT_NAME">AUFTRAG_ANGELEGT</attribute>
<attribute type="AUFTRAGNR">4711</attribute>
<attribute type="AUFTRAGART">Rush order</attribute>
<attribute type="START_TIME">11.11.2002 11:11:11</attribute>
<attribute type="PPM_ORG_NAME">Mr Miller</attribute>
<attribute type="PPM_ORG_NUM">1</attribute>
</event>
...
<event>
<attribute type="PPM_EVENT_ID">A-Nr 4711</attribute>
<attribute type="PPM_EVENT_NAME">RECHNUNG_ERSTELLT</attribute>
<attribute type="AUFTRAGNR">4711</attribute>
<attribute type="AUFTRAGART">Rush order</attribute>
<attribute type="START_TIME">12.11.2002 14:21:15</attribute>
<attribute type="PPM_ORG_NAME">Ms. Smith</attribute>
<attribute type="PPM_ORG_NUM">1</attribute>
</event>
...
</eventlist>
```

The format of the XML output file is specified by the DTD for PPM system event format:



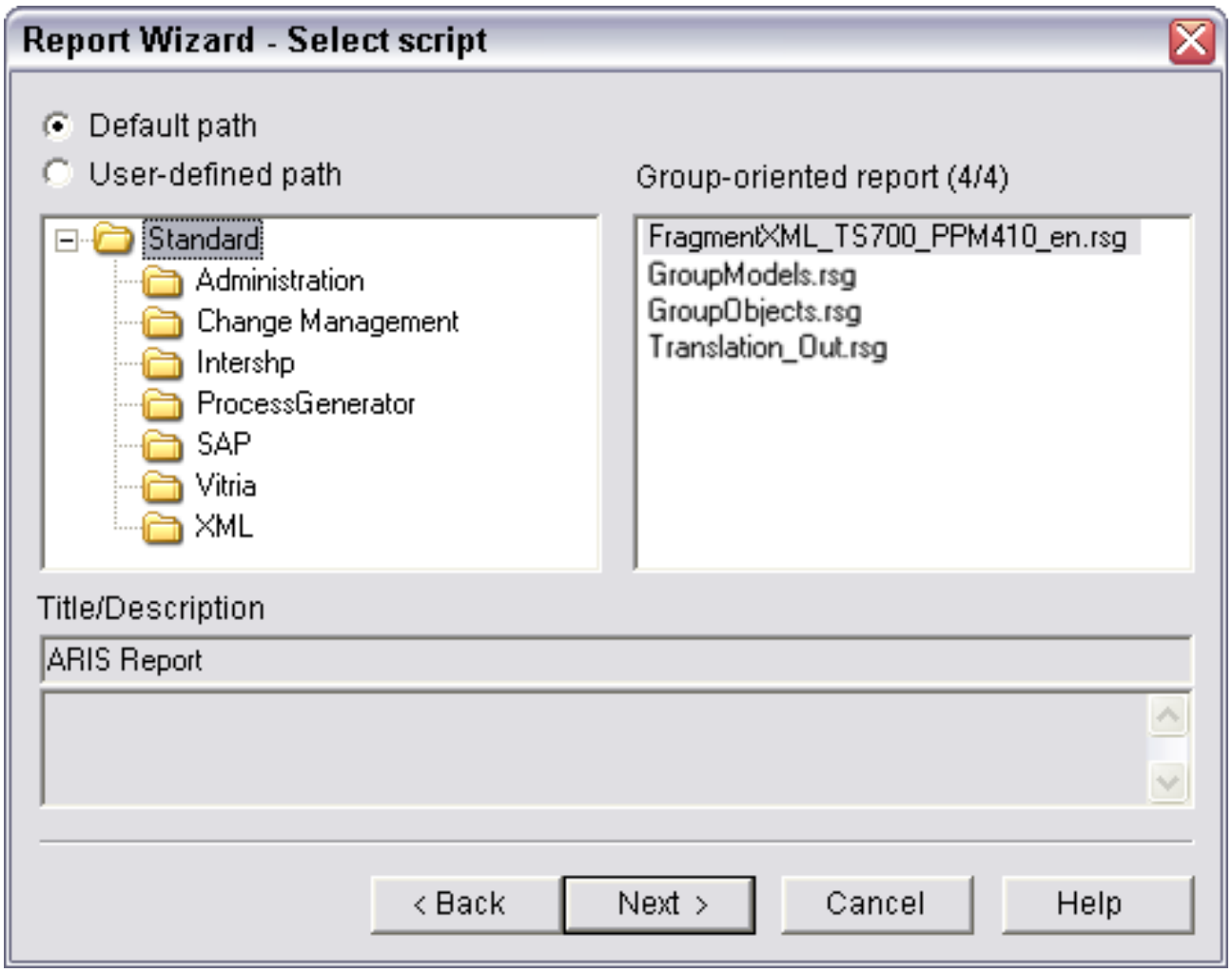
The **id** and **name** XML attributes of the **event** XML element and the **org** XML element enable you to continue using system event files generated for PPM versions 1.x and 2.x with no changes.

From PPM version 3.0 they are replaced by the **PPM_EVENT_ID** and **PPM_EVENT_NAME** system event attributes and by the **orgunit** XML element.

3.2.3.4. Run the ARIS report

The ARIS report for generating the fragment definition XML file is designed as a group report.

Copy the report script **FragmentXML_TS700_PPM410_en.rsg** located in the <PPM installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ARIS to the subdirectory **script\report\en** of your ARIS installation (ARIS default directory for report scripts). To start the report, select the **Evaluate/Report** pop-up menu for the group containing the fragment definitions.



If you are using ARIS Architect import the script **FragmentXML_BA700_PPM410.arx**. For importing, we recommend that you create a separate category called **PPM**. To run the report use the entry **FragmentXML**.

After the report has run successfully, the XML fragment definition file can be found in the selected output directory. The default directory is the **script\report\out** directory in the ARIS installation. The file name is made up of the prefix **Frag_** and a time stamp.

- Ensure that the report runs with no warnings or error messages. The message **Access to an attribute that is not specified** indicates that a required attribute has not been entered, for example, the **Identifier** attribute. In this case, the XML files generated cannot be used.
- If you do not have access to an up-to-date report script for creating fragment definition files, you must convert the **FragmentXML.rsg** script from the directory **\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\ctk\ARIS** of your PPM installation to a currently valid form using **ARIS Script Converter**.

3.2.4. Generating the XML output file

To generate the output file, the possible system events must be identified and named. The example below uses the system events **Order created** and **Invoice created**.

The occurrence of these system events is written to an XML output file with additional information (for example, order number, processing times, base values for measure calculation and the creation of dimensions).

3.2.5. Summary

The simple example below is intended to highlight the import in system event format described.

To import the output files, the fragment definition file **frag.xml** and the mapping file **map.xml** are created:

Fragment definitions in the file **frag.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE graphlist SYSTEM "Graph.dtd">
<graphlist>
<graph id="FRG_ORD_CREATED">
<node id="EVT_ORD_TOBECREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">Customer order to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_ORD_TOBECREATED</attribute>
</node>
<node id="FCT_CREATE_ORDER" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create customer order</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_CREATE_ORDER</attribute>
</node>
<node id="EVT_ORD_CREATED" type="OT_EVT">
<attribute type="AT_OBJNAME">Customer order created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_ORD_CREATED</attribute>
</node>
<edge type="CXN_FOLLOWS" source="EVT_ORD_TOBECREATED"
target="FCT_CREATE_ORDER"/>
<edge type="CXN_FOLLOWS" source="FCT_CREATE_ORDER" target="EVT_ORD_CREATED"/>
</graph>
<graph id="FRG_INVOICED">
<node id="EVT_TOBE_INVOICED" type="OT_EVT">
<attribute type="AT_OBJNAME">Invoice to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_TOBE_INVOICED</attribute>
</node>
<node id="FCT_INVOICE" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create invoice</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_INVOICE</attribute>
</node>
<node id="EVT_INVOICED" type="OT_EVT">
<attribute type="AT_OBJNAME">Invoice created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_INVOICED</attribute>
</node>
<edge type="CXN_FOLLOWS" source="EVT_TOBE_INVOICED" target="FCT_INVOICE"/>
```

```
<edge type="CXN_FOLLOWS" source="FCT_INVOICE" target="EVT_INVOICED"/>
</graph>
</graphlist>
```

Mapping definitions in the file **map.xml**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE eventmapping SYSTEM "eventmapping.dtd">
<eventmapping>
<processfragmentmapping>
<processfragment graphid="FRG_ORD_CREATED">
<condition eventattributetype="PPM_EVENT_NAME">
<value>FRG_ORD_CREATED</value>
</condition>
</processfragment>
<processfragment graphid="FRG_INVOICED">
<condition eventattributetype="PPM_EVENT_NAME">
<value>FRG_INVOICED</value>
</condition>
</processfragment>
</processfragmentmapping>
<attributemapping>
<objectattributes objectname="EVT_ORD_TOBECREATED" graphid="FRG_ORD_CREATED">
<attribute ppmattributetype="AT_MERGE_KEY_1">
<eventattributetype>EVT_PRED</eventattributetype>
</attribute>
</objectattributes>
<objectattributes objectname="FCT_CREATE_ORDER" graphid="FRG_ORD_CREATED">
<orgunit eventattributetype="PPM_ORG_NAME"/>
<attribute ppmattributetype="AT_MERGE_KEY_2">
<eventattributetype>START_TIME</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_KI_FBZ">
<eventattributetype>PROCESSINGTIME</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_SAPCLIENT">
<eventattributetype>KUNDENNR</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_SAP_BSTYP">
<eventattributetype>AUFTRAGAT</eventattributetype>
</attribute>
</objectattributes>
<objectattributes objectname="EVT_ORD_CREATED" graphid="FRG_ORD_CREATED">
<attribute ppmattributetype="AT_ID">
<eventattributetype>AUFTRAGNR</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_SAPCLIENT">
<eventattributetype>KUNDENNR</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_MERGE_KEY_1">
<eventattributetype>EVT_NR</eventattributetype>
</attribute>
</objectattributes>
```

```
<objectattributes objectname="EVT_TOBE_INVOICED" graphid="FRG_INVOICED">
<attribute ppmattributetype="AT_MERGE_KEY_1">
<eventattributetype>EVT_PRED</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_ID">
<eventattributetype>AUFTRAGNR</eventattributetype>
</attribute>
</objectattributes>
<objectattributes objectname="FCT_INVOICE" graphid="FRG_INVOICED">
<orgunit eventattributetype="PPM_ORG_NAME"/>
<attribute ppmattributetype="AT_MERGE_KEY_2">
<eventattributetype>START_TIME</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_KI_FBZ">
<eventattributetype>PROCESSINGTIME</eventattributetype>
</attribute>
</objectattributes>
<objectattributes objectname="EVT_INVOICED" graphid="FRG_INVOICED">
<attribute ppmattributetype="AT_ID">
<eventattributetype>AUFTRAGNR</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_SAPCLIENT">
<eventattributetype>KUNDENNR</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_MERGE_KEY_1">
<eventattributetype>EVT_NR</eventattributetype>
</attribute>
</objectattributes>
</attributemapping>
</eventmapping>
```

Extracting data from the application system has created the XML output file **events.xml**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE eventlist SYSTEM "event.dtd">
<eventlist>
<event>
<attribute type="EVT_PRED">0</attribute>
<attribute type="PPM_EVENT_ID">4711</attribute>
<attribute type="PPM_EVENT_NAME">FRG_ORD_CREATED</attribute>
<attribute type="PPM_ORG_NAME">Mr Miller</attribute>
<attribute type="AUFTRAGNR">4711</attribute>
<attribute type="AUFTRAGART">Rush order</attribute>
<attribute type="START_TIME">11.11.2002 11:11:11</attribute>
<attribute type="KUNDENNR">5711</attribute>
<attribute type="PROCESSINGTIME">7 MINUTE</attribute>
<attribute type="EVT_NR">1</attribute>
</event>
<event>
<attribute type="EVT_PRED">1</attribute>
<attribute type="PPM_EVENT_ID">4711</attribute>
<attribute type="PPM_EVENT_NAME">FRG_INVOICED</attribute>
<attribute type="PPM_ORG_NAME">Ms. Smith</attribute>
```

```
<attribute type="AUFTRAGNR">4711</attribute>
<attribute type="KUNDENNR">5711</attribute>
<attribute type="START_TIME">12.11.2002 14:21:15</attribute>
<attribute type="PROCESSINGTIME">14 MINUTE</attribute>
<attribute type="EVT_NR">2</attribute>
</event>
</eventlist>
```

Executing the command line

```
runxmlimport -user system -password manager -f frag.xml -m
map.xml -i event.xml
```

generates process instance fragments in the PPM database. The listing below in PPM graph format illustrates the process instance fragments generated in the PPM database. In the listing, the process instance fragment information, which has been added to the fragment definition, is written in bold:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE graphlist SYSTEM "graph.dtd">
<graphlist>
<graph id="FRG_ORD_CREATED" xml:lang="en">
<node id="FRG_ORD_CREATED-EVT_ORD_TOBECREATED-1-8835472025300230616"
type="OT_EVT">
<attribute type="AT_OBJNAME">Customer order to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_ORD_TOBECREATED</attribute>
<attribute type="AT_EPK_FRAGMENT_ID">FRG_ORD_CREATED</attribute>
<attribute type="AT_INTERNAL_OBJECT_KEY">EVT_ORD_TOBECREATED0</attribute>
<attribute type="AT_MERGE_KEY_1">0</attribute>
<attribute type="AT_ORIG_EPK_ID">1</attribute>
</node>
<node id="FRG_ORD_CREATED-FCT_CREATE_ORDER-2-8344758599463564840"
type="OT_FUNC">
<attribute type="AT_OBJNAME">Create customer order</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_CREATE_ORDER</attribute>
<attribute type="AT_EPK_FRAGMENT_ID">FRG_ORD_CREATED</attribute>
<attribute type="AT_INTERNAL_OBJECT_KEY">
FCT_CREATE_ORDER11.11.2002 11:11:11</attribute>
<attribute type="AT_KI_FBZ">7 MINUTE</attribute>
<attribute type="AT_MERGE_KEY_2">11.11.2002 11:11:11</attribute>
<attribute type="AT_ORIG_EPK_ID">1</attribute>
<attribute type="AT_SAPCLIENT">5711</attribute>
</node>
<node id="FRG_ORD_CREATED-EVT_ORD_CREATED-3-7299716715746582786"
type="OT_EVT">
<attribute type="AT_OBJNAME">Customer order created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_ORD_CREATED</attribute>
<attribute type="AT_EPK_FRAGMENT_ID">FRG_ORD_CREATED</attribute>
<attribute type="AT_ID">4711</attribute>
<attribute type="AT_INTERNAL_OBJECT_KEY">EVT_ORD_CREATED1</attribute>
<attribute type="AT_MERGE_KEY_1">1</attribute>
<attribute type="AT_ORIG_EPK_ID">1</attribute>
```

```
<attribute type="AT_SAPCLIENT">5711</attribute>
</node>
<node id="FRG_ORD_CREATED-1--2--7661491465378853387" type="OT_ORG">
<attribute type="AT_INTERNAL_OBJECT_KEY">
FRG_ORD_CREATED-1--2-7661491465378853387</attribute>
<attribute type="AT_OBJNAME">Mr Miller</attribute>
</node>
<edge type="CXN_FOLLOWS"
source="FRG_ORD_CREATED-EVT_ORD_TOBECREATED-1-8835472025300230616"
target="FRG_ORD_CREATED-FCT_CREATE_ORDER-2-8344758599463564840" />
<edge type="CXN_FOLLOWS"
source="FRG_ORD_CREATED-FCT_CREATE_ORDER-2-8344758599463564840"
target="FRG_ORD_CREATED-EVT_ORD_CREATED-3-7299716715746582786" />
<edge type="CXN_UNDIRECTED"
source="FRG_ORD_CREATED-1--2-7661491465378853387"
target="FRG_ORD_CREATED-FCT_CREATE_ORDER-2-8344758599463564840">
<attribute type="AT_COUNT_PROCESSINGS">1</attribute>
</edge>
</graph>
<graph id="FRG_INVOICED" xml:lang="en">
<node id="FRG_INVOICED-EVT_TOBE_INVOICED-1-246376083169224336" type="OT_EVT">
<attribute type="AT_OBJNAME">Invoice to be created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_TOBE_INVOICED</attribute>
<attribute type="AT_EPK_FRAGMENT_ID">FRG_INVOICED</attribute>
<attribute type="AT_ID">4711</attribute>
<attribute type="AT_INTERNAL_OBJECT_KEY">EVT_TOBE_INVOICED1</attribute>
<attribute type="AT_MERGE_KEY_1">1</attribute>
<attribute type="AT_ORIG_EPK_ID">2</attribute>
</node>
<node id="FRG_INVOICED-FCT_INVOICE-2--1285282699037363371" type="OT_FUNC">
<attribute type="AT_OBJNAME">Create invoice</attribute>
<attribute type="AT_OBJNAME_INTERN">FCT_INVOICE</attribute>
<attribute type="AT_EPK_FRAGMENT_ID">FRG_INVOICED</attribute>
<attribute type="AT_INTERNAL_OBJECT_KEY">FCT_INVOICE12.11.2002
14:21:15</attribute>
<attribute type="AT_KI_FBZ">14 MINUTE</attribute>
<attribute type="AT_MERGE_KEY_2">12.11.2002 14:21:15</attribute>
<attribute type="AT_ORIG_EPK_ID">2</attribute>
</node>
<node id="FRG_INVOICED-EVT_INVOICED-3-1541227247726154405" type="OT_EVT">
<attribute type="AT_OBJNAME">Invoice created</attribute>
<attribute type="AT_OBJNAME_INTERN">EVT_INVOICED</attribute>
<attribute type="AT_EPK_FRAGMENT_ID">FRG_INVOICED</attribute>
<attribute type="AT_ID">4711</attribute>
<attribute type="AT_INTERNAL_OBJECT_KEY">EVT_INVOICED2</attribute>
<attribute type="AT_MERGE_KEY_1">2</attribute>
<attribute type="AT_ORIG_EPK_ID">2</attribute>
<attribute type="AT_SAPCLIENT">5711</attribute>
</node>
<node id="FRG_INVOICED-1--2--8231301810541650135" type="OT_ORG">
<attribute type="AT_INTERNAL_OBJECT_KEY">FRG_INVOICED-1--2-
8231301810541650135</attribute>
<attribute type="AT_OBJNAME">Ms. Smith</attribute>
</node>
<edge type="CXN_FOLLOWS"
```

```
source="FRG_INVOICED-EVT_TOBE_INVOICED-1-246376083169224336"
target="FRG_INVOICED-FCT_INVOICE-2--1285282699037363371" />
<edge type="CXN_FOLLOWS"
source="FRG_INVOICED-FCT_INVOICE-2-1285282699037363371"
target="FRG_INVOICED-EVT_INVOICED-3-1541227247726154405" />
<edge type="CXN_UNDIRECTED"
source="FRG_INVOICED-1--2--8231301810541650135"
target="FRG_INVOICED-FCT_INVOICE-2--1285282699037363371">
<attribute type="AT_COUNT_PROCESSINGS">1</attribute>
</edge>
</graph>
</graphlist>
```

3.3. Data formats

This chapter describes the formats of the source system data, which are expected in the form of attribute values in the XML output file.

Attribute mapping assigns a PPM attribute type to the source system attributes. Source system attributes are pure carriers of information in the form of a string. The data type of the PPM attribute specifies how the text of the source system attribute will be interpreted.

Numerical data

Numerical data is essentially interpreted as information in the decimal system. Floating point numbers use full stops as separators between places before and after the decimal point, regardless of the setting of the operating system.

Attribute values are imported with a unit, for example, **57 USD**. If no unit is specified, the base unit of the data type on which the PPM attribute type is based will be assumed.

Percentages are either specified as **58 PERCENT** or as a factor in the form of **0.58 VALUE_ONLY**.

Time-based data

PPM differentiates between three categories of time-based reference data:

Date:

The day of the calendar is specified, for example, **26.06.2003**.

Time:

The time of day is specified in 24-hour format, for example, **14:29**.

Time stamp:

The specification of a time stamp is made up of date and time information, for example, **26.06.2003 14:29**.

For parsing of the data string, the format string specified in the file **AdapterConfig_settings.properties** in the client configuration folder for the relevant category is used.

Warning
The format string information from the file **AdapterConfig_settings.properties** is used when importing in graph format. To import time data that differs from the default PPM format, the **format** XML attribute is provided in attribute mapping from PPM 3.0.

As an extension to the Java standard (see Definition of attribute mapping [20], **Transformations** section), PPM provides the format string **Q.yyyy**, which enables time values to be imported as a quarter, for example, **2.2003** for the second quarter of 2003.

3.3.1. Special characters in XML documents

Special characters such as **&**, **>**, **<** and " are control characters (meta characters) of the XML document. If these characters directly occur in the source data, misinterpretations may occur while importing the XML file. To be able to use these characters in the reference data, you need to replace them by the following entities (strings):

Characters	Entity	Characters	Entity
&	&	"	"
<	<	>	>

The table below shows the representation of special characters in XML files:

Characters	Entity	Characters	Entity	Characters	Entity
¡	¡	»	»	£	£
¥	¥	|	¦	§	§
©	©	ª	ª	«	«
®	®	°	°	±	±
1	¹	2	²	3	³
¼	¼	½	½	¾	¾
Ä	Ä	Ö	Ö	Ü	Ü
ä	ä	ö	ö	ü	ü
ß	ß	x	×	÷	&divide
È	È	É	É	Ê	Ê
Ë	Ë	è	È	é	É
ê	Ê	ë	Ë	Ø	&Oslach;

You can use any character by indicating its font code. To do this, specify the ASCII code of the relevant character in decimal notation as follows: **&#<Decimal code of character>;**.

For example, to use the @ character, enter **@** in the XML document.

3.4. Generating the process instance fragments

Graph format

Importing into the PPM system involves importing the fragment instance data contained in the XML output file and generating process instance fragments.

Event format

Importing into the PPM system involves mapping the instance data for the system events contained in the XML output file to the process fragment definitions in the specified fragment file and generating process instance fragments. The attributes specified in the mapping file are then copied to the process instance fragments generated.

Regardless of the format used to import the fragment instance, before a process instance fragment is finally saved in the PPM database, process keys (see Merge process fragments [82]) are calculated to enable the process instance fragments to be merged into a process instance.

> **Warning**
> At least one process key must be calculated for each imported process instance fragment. Process instance fragments for which no process key can be calculated are not saved in the PPM database. These fragments are not a component of an overall process.

During the import, relevant status messages are output on the command prompt and optionally in the specified log file.

3.4.1. Extending the attribute configuration

You can automatically identify attributes contained in the XML import files that are not known in the PPM system. Optionally, you can assign a data type to the new attributes added and automatically transfer them to the attribute configuration of your PPM client.

Automatic attribute identification supports both data import formats.

You can use the following options to enable automatic extension of the attribute configuration:

Command line arguments

When calling up **runxmlimport**, specify the additional arguments **-autoextendattributes** and **-extractattributes** in the command line.

Specifying these arguments has the following effect when importing:

Argument	Effect
-extractattributes <File name>	Unknown attributes are saved in the specified file. No XML fragment files are imported.
-autoextendattributes	Unknown attributes are transferred to the attribute configuration. If the **automapping** option is enabled, the attribute mapping configuration is extended accordingly. The XML fragment data is then imported.
Both arguments are specified	The actions described under -autoextendattributes and -extractattributes are performed consecutively.

Data source file

Assign the value **true** to the **autoextendattributes** attribute of the **attributesettings** XML element in the **datasource** file used.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datasource SYSTEM "datasource.dtd">
```

```
<datasource name="Events" type="EVENT">
...
<attributesettings autoextendattributes="true">
...
</attributesettings>
</datasource>
```

Certain attributes can be excluded from automatic attribute identification using pattern identification. To do this, specify the name pattern of the attributes you want to exclude from automatic extension in the **excludepattern** XML element. You can use the placeholders **?** (any single character) and ***** (any set of characters).

In the example below, all system event attributes whose name begins with **TEST** or that have the name **USER** are excluded from automatic attribute identification.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datasource SYSTEM "datasource.dtd">

<datasource name="Events" type="EVENT">
...
<attributesettings autoextendattributes="true">
<excludepatterns>
<excludepattern>TEST*</excludepattern>
<excludepattern>USER</excludepattern>
</excludepatterns>
...
</attributesettings>
</datasource>
```

3.4.1.1. Specify the data type of unknown attributes

You can specify the data type to be assigned to an unknown attribute in the following ways:

- Meta data specified in the system event file (only for event import format)
- Pattern identification in the attribute name, for example, **TEXT_***
- Attribute value parsing

The sequence shown corresponds to the prioritization of the automatic data type identification.

> **Warning**
> The data type of existing PPM attributes cannot be modified.

Meta data

In the system event files to be imported, you can specify a PPM data type and, optionally, a format (for example, to identify time stamps) and attribute description for each attribute of a system event. Attributes that are not yet known in the PPM system are automatically generated with the specified data type.

Meta data is specified in the **metadata** XML element:

```
<eventlist>
<metadata>
<attr_desc type="...">
<ppmdatatype>...</ppmdatatype>
<format>...</format>
<description>...</description>
</attr_desc>
...
</metadata>

<event>
...
</event>
...
</eventlist>
```

The XML tags for the **metadata** XML element have the following meaning:

XML tag	Description
attr_desc	Individual meta data definition of a system event attribute. Multiple definitions can be specified.
type	Name of the system event attribute
ppmdatatype	PPM data type to be assigned to the PPM attribute generated.
format (optional)	Format to be used for parsing the attribute value. If you have also enabled automatic mapping, the format is also used to transform the value into PPM time format.
description (optional)	Description of the PPM attribute generated. The specified description is transferred in the attribute configuration in the default language of the PPM client.

```
<eventlist>
<metadata>
<attr_desc type="SWWWIHEAD-WI_ID">
<ppmdatatype>TEXT</ppmdatatype>
<description>Work item ID</description>
</attr_desc>
<attr_desc type="SWWWIRET-WI_AED">
<ppmdatatype>DAY</ppmdatatype>
<format>yyyyMMdd</format>
<description>End date of work item</description>
</attr_desc>
</metadata>

<event>
<attribute type="SWWWIHEAD-WI_ID">000000525723</attribute>
<attribute type="SWWWIRET-WI_AED">20011211</attribute>
</event>
</eventlist>
```

The extractors available for PPM (PPM Process Extractor SAP-2-PPM and PPM Process Extractor JDBC-2-PPM) automatically generate the **metadata** XML element, allowing you to directly import the system event files generated without doing anything. Attributes that are not yet known in the PPM system are automatically generated with the correct data type.

Pattern identification

The **datatypekeydetectionsettings** XML element in the data source file can be used to specify a list of patterns (**datatypedetectionpattern** XML elements) to be used for data type assignment. The names of all new attributes identified are compared with the patterns. The first pattern identified in the list specifies the data type of the attribute.

Example 20.

The following extract from the data source file configures the following naming pattern to identify the data type of new attributes identified:

- A system event whose name begins with **L_** or ends in **L** is assigned the data type **LONG**.
- A system event whose name begins and ends in **D** is assigned the data type **DOUBLE**.

```
...
<attributesettings autoextendattributes="true">
<datatypedetectionsettings>
<datatypekeydetectionsettings >
<datatypedetectionpattern datatype=”LONG”>*L
</datatypedetectionpattern>
<datatypedetectionpattern datatype=”LONG”>L_*
</datatypedetectionpattern>
<datatypedetectionpattern datatype=”DOUBLE”>D*D
</datatypedetectionpattern>
</datatypekeydetectionsettings>
</datatypedetectionsettings>
</attributesettings>
...
```

Attribute value parsing

If pattern identification is unable to identify a data type, an attempt is made to identify the data type based on the value of the attribute. The following data types are identified: **BOOLEAN**, **LONG**, **DOUBLE**, **TIME**, **TIMEOFDAY**, **DAY**, and **TEXT**. The enumeration corresponds to the priority for data type identification.

Data type BOOLEAN

The attribute values **true** and **false** are assigned to the **BOOLEAN** data type regardless of capitalization. Other values, for example, **0** and **1** are not identified as **BOOLEAN**.

Data types LONG and DOUBLE

If the **BOOLEAN** data type is not identified, the system attempts to identify numerical values. When identifying numerical data types, a distinction is made between integers and floating point numbers. In the **doubleonly** attribute for the **datatypevaluedetectionsettings** XML element, you can use the value **TRUE** (default value) to specify that integer attribute values should be assigned to the data type **DOUBLE**.

Identification of the **DOUBLE** data type does not require you to specify a thousands separator. The point (period) is the only decimal separator identified.

Data types TIME, TIMEOFDAY, and DAY

If no numerical data type is identified, the system attempts to identify the attribute value as a time stamp (**TIME**), time of day (**TIMEOFDAY**), or date (**DAY**). The enumeration corresponds to the priority for data type identification. For each of the data types time stamp, time of day and date, you can optionally use the **timeformat**, **timeofdayformat** and **dayformat** XML elements to specify format strings describing the formats of the specified attribute values. If you do not specify format strings, the following default formats apply:

Data type	Default format
TIME	dd.MM.yyyy HH:mm:ss
TIMEOFDAY	MM/dd/yyyy
DAY	HH:mm:ss

Data type TEXT

If no data type has been identified so far by parsing the attribute value, the attribute is assigned the data type **TEXT**.

Behavior with ambiguous data types

You can use the **numberofvaluestocheck** attribute for the **datatypevaluedetectionsettings** XML element to specify how often the data type is to be retrieved by parsing the attribute value. If the XML attribute is not specified, the default value of **100** is used. If different data types are identified for an attribute, the data type previously identified is converted into a general data type. The following table applies:

Data type	General data type
BOOLEAN	TEXT
LONG	DOUBLE, TEXT
DOUBLE	TEXT
TIME	TEXT
TIMEOFDAY	TEXT
DAY	TEXT
TEXT	TEXT

Example 21.

The file extract below shows a complete configuration for automatic data type identification by parsing the attribute value.

```
...
<attributesettings autoextendattributes="true">
<excludepatterns>
<excludepattern>TEST*</excludepattern>
<excludepattern>USER</excludepattern>
</excludepatterns>
<datatypedetectionsettings>
<datatypekeydetectionsettings>
<datatypedetectionpattern datatype = "LONG">LG_*
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "LONG">*LONG*
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "LONG">*_lng
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "DOUBLE">DOUBLE_*
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "DOUBLE">*_dbl
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "DAY">DAY_*
</datatypedetectionpattern>
```

```
<datatypedetectionpattern datatype = "DAY">*_day
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "TIMEOFDAY">TIMEOFDAY_*
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "TIMEOFDAY">*_tod
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "TIME">TIME_*
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "TIME">*_tm
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "TEXT">TEXT_*
</datatypedetectionpattern>
<datatypedetectionpattern datatype = "TEXT">*_txt
</datatypedetectionpattern>
</datatypekeydetectionsettings>
<datatypevaluedetectionsettings
doubleonly = "FALSE"
numberofvaluestocheck = "100">
<timeformat>dd.MM.yyyy HH:mm:ss</timeformat>
<timeofdayformat>HH:mm:ss</timeofdayformat>
<dayformat>dd.MM.yyyy</dayformat>
</datatypevaluedetectionsettings>
</datatypedetectionsettings>
<attributeprefix>AT_</attributeprefix>
</attributesettings>
...
```

3.4.2. Extending the mapping configuration

If you are using event format to import, you can enable automatic mapping, that is, all attributes of a system event are transferred to the specified object types in the assigned fragment definition.

Extension of the mapping configuration is configured using attributes of the **automapping** XML element.

XML attribute	Description
nodetype	determines the object type for which this **automapping** is valid. Valid values: **OT_FUNC** for functions, **OT_EVT** for events and **PROCESS** for processes.
graphid	determines the fragment definition graph for whose objects this **automapping** is valid. The specified value corresponds to the **id** attribute of the **graph** XML element in the fragment definition.
addmergeattributes (optional)	For the object type **PROCESS**, determines whether the attributes are added to the merge configuration, which means that they will be retained as process attributes when merging instance fragments (value **TRUE**), or not (value **FALSE**). For the object types **OT_FUNC** and **OT_EVT**, this entry is ignored. For the object type **PROCESS**, you must specify **addmergeattributes**.

If you have to specify automatic mapping extension for multiple object types or fragment definition graphs, you can specify a separate **automapping** XML element for each object type or fragment definition graph required. Automatic mapping takes account of the prefix specified in the **attributeprefix** XML element, for example, **AT_**. If you have specified an explicit mapping for particular attributes, this overwrites the attributes previously transferred by automatic mapping.

Example 22.

In the file extract below, automatic mapping extension is configured for functions of the fragment definition graph **FRG_CATCH_ALL**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE eventmapping SYSTEM "eventmapping.dtd">
<eventmapping>
<processfragmentmapping>
<!--FRG_CATCH_ALL-->
<processfragment graphid="FRG_CATCH_ALL">
...
</processfragment>
...
</processfragmentmapping>
<attributemapping>
...
<automapping
nodetype="OT_FUNC"
graphid="FRG_CATCH_ALL"
/>
<attributemapping>
</eventmapping>
```

3.4.3. Multi-valued system event attributes

If a system event attribute occurs several times in a system event, there is no guarantee that the last value read will be transferred. To transfer all values for attributes that occur several times in a system event, you can specify the **multieventattributetype** XML element instead of the **attribute** XML element in the mapping configuration. All values of system event attributes identified in this way are concatenated with a separator (semicolon by default). Use the optional **delimiter** XML attribute to specify a separator other than the default.

Using multiple system event attribute values makes it easier to merge parallel process paths if you write all merge keys for the preceding fragment to the system event file as **multieventattributetype** attributes.

Example 23.

System event file extract

```
...
<event>
<attribute type="EVENTTYP">Change customer order</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>
</event>
...
```

Example 24.

Mapping file extract

```
...
<attributemapping>
```

```
...
<!-- Mapping Event Start BEGIN -->
<objectattributes objectname="EVT_START" graphid="FRG_CATCH_ALL">
<attribute ppmattributetype="AT_OBJNAME">
<eventattributetype>EVENTTYP</eventattributetype>
<value> to be done</value>
</attribute>
<attribute ppmattributetype="AT_ID">
<eventattributetype>AT_PRCNO</eventattributetype>
</attribute>
<!-- Mulivalue Mapping -->
<attribute ppmattributetype="AT_KEY">
<multieventattributetype delimiter=";">PREV_KEY</multieventattributetype>
</attribute>
...
</objectattributes>

...
```

The **AT_KEY** attribute generated for the **EVT_START** event is assigned the value **1;2**. If the merge is configured accordingly, two merge keys will be generated for this event.

3.4.4. Direct import of process attributes

The attributes required to calculate measures and dimensions at process instance level are normally copied to the process instance from objects in the imported fragment instances. This operation can be configured using attribute copy rules or a corresponding calculation rule in the Measure calculator.

Alternatively, you can directly import process instance attributes (graph format) or generate them by mapping system event attributes to the fragment definition (event format).

To overwrite process attributes of existing process instances, you can directly import fragment instances without objects and connections containing exclusively process attributes (graph format) or generate them by mapping system event attributes to a fragment definition with no objects (event format).

Graph format

Example 25.

Graph without objects and connections

```
...
<graph id="FRG_EMPTY">
<attribute type="AT_ID">1</attribute>
<attribute type="AT_SAP_BELEGNR">Document 2</attribute>
<attribute type="AT_SAPCLIENT">R3</attribute>
</graph>
...
```

Event format

When using event format, the system event attributes are transferred to the instantiated graphs in a fragment definition with no objects.

Example 26.

Extract from fragment definition with no objects:

```
...
<graph id="FRG_EMPTY">
<attribute type="AT_ID">1</attribute>
</graph>
...
```

Example 27.

Extract from mapping rule:

```
...
<processattributes graphid="FRG_EMPTY">
<attribute ppmattributetype="AT_SAP_BELEGNR">
<value>Document </value>
<eventattributetype>SAP_BELEGNR</eventattributetype>
</attribute>
<attribute ppmattributetype="AT_SAPCLIENT">
<eventattributetype>SAPCLIENT</eventattributetype>
</attribute>
</processattributes>
...
```

Warning
In order to be able to assign fragments to be imported with no objects to process instances, you must ensure that valid process keys can be calculated based on process attributes.

To ensure that directly imported process attributes are retained in the instance, you must specify every directly imported attribute in the **mergeattributes** XML element for the merge configuration.

```
...
<processmerge>
<mergeattributes>
<attribute key = "AT_SAP_BELEGNR"/>
<attribute key = "AT_SAPCLIENT"/>
</mergeattributes>
</processmerge>
...
```

3.4.5. Special case of scaled system

In a scaled system, data is imported to the sub-server. All sub-servers must have an identical configuration. The **autoextendattributes** and **addmergeattributes** options would change the configuration of an individual sub-server and the uniform configuration of all sub-servers would be lost. Therefore, using these two options is not possible in a scaled system.

If you still want to use the automatic extension of the attribute configuration, you need to export the possible extensions for each sub-server and manually import these on the master server, which can then distribute the extensions to all sub-servers in the system.

Before you import data with new attributes to the sub-server, perform the following steps:

1. Identify new system event attributes and extract them using the -**extractattributes <File name>** parameter in the **xmlimport** command line program.
2. Import the new attributes on the master server.
3. Identify new attributes that you want to be retained when merging process fragments and extract these using the **-extractmergeattributes <File name>** parameter in the **xmlimport** command line program.
4. Manually extend the merge configuration on the master server, as described in the **Add the merge attributes** section below.
5. Perform an XML data import to the sub-server with the **automapping** option.

Add the merge attributes

If you have specified automatic mapping extension for processes (**<automapping nodetype="PROCESS" ...**), you can extract the new attributes added that are to be retained when merging process fragments by specifying the **-extractmergeattributes <File name>** parameter in the command line. No XML data is imported. The file generated contains only a **mergeattributes** XML element with a list of all new attributes, and has the following structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mergeattributes>
<attribute key="..." />
...
</mergeattributes>
```

Use the **-export -merger <File name>** parameter to export the merge configuration on the master server. Edit the merge configuration by adding the attributes (**attribute** XML elements for **mergeattributes** XML element) to all merge attributes from the exported master server merge configuration previously exported for each sub-server using **-exctractmergeattributes <File name>**.

Finally, use the **-import -merger <File name>** parameter to import the merge configuration from the master server.

Example 28.

Exported master server merge configuration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mergerconfig SYSTEM "mergerconfig.dtd">
<mergerconfig>
<mergehandling>
<processmerge>
<mergeattributes>
<attribute key = "AT_SAPSYSTEM"/>
<attribute key = "AT_SAP_BELEGNR"/>
</mergeattributes>
</processmerge>
<eventmerge priority="1">
<mode>
<keymerge/>
</mode>
</eventmerge>
</mergehandling>
```

```
</mergerconfig>
```

Example 29.

Exported merge attributes for sub-server 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mergeattributes>
<attribute key = "AT_SAPCLIENT"/>
</mergeattributes>
```

Exported merge attributes for sub-server 2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mergeattributes>
<attribute key="AT_SAP_BSTYP" />
<attribute key="AT_SAP_BSTYP" />
</mergeattributes>
```

Example 30.

Consolidated master server merge configuration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mergerconfig SYSTEM "mergerconfig.dtd">
<mergerconfig>
<mergehandling>
<processmerge>
<mergeattributes>
<attribute key = "AT_SAPSYSTEM"/>
<attribute key = "AT_SAP_BELEGNR"/>
<!-- Merge attributes sub-server 1 -->
<attribute key = "AT_SAPCLIENT"/>
<!-- Merge attributes sub-server 2 -->
<attribute key="AT_SAP_BSTYP" />
<attribute key="AT_SAP_BSTYP" />
</mergeattributes>
</processmerge>
<eventmerge priority="1">
<mode>
<keymerge/>
</mode>
</eventmerge>
</mergehandling>
</mergerconfig>
```

3.4.6. Archiving of XML import files

If you want to prevent XML import files already imported from being imported again next time, you can specify that the imported files will be renamed or moved to a different directory.

The option of archiving import files is configured using the **archive** XML element in the data source file used:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datasource SYSTEM "datasource.dtd">

<datasource name="Events" type="EVENT">
...
<archive>
<directory>...</directory>
<prefix>...</prefix>
</archive>
</datasource>
```

XML element	Description
directory	Specifies the directory to which the imported XML import files will be moved. Any directories in the specified path that do not yet exist will be created automatically.
prefix	Specifies the prefix given to the name of the imported XML import files.

If you specify both XML attributes, evaluation of the **directory** attribute is given priority and the **prefix** attribute is ignored.

3.5. runxmlimport command line program

XML import files are imported using the **runxmlimport** command line program. During an import, process instance fragments are imported into the PPM system's internal buffer and initially not processed further.

> **Warning**
> Make sure that the PPM server has been started before the XML import. Only one command line program can be active.

If an error occurs during the import operation, an error message appears on the console and the return value of the program is not equal to **0**.

Calling up the program without parameters or with **-h** or **-?** outputs the online help on the console. The help describes all available options:

Import process instances in XML format

runxmlimport -user <user name> -password <password> [-client <name>]

[-datasource <file>]

[-datasourcelist <file>]

[-i <file1>[,<file2>...]]

[-f <fragment file> -m <mapping file>]

[-extractattributes <attribute file>] [-autoextendattributes]

[-extractmergeattributes <mergeattribute file>]

[protocoloptions] [-language <ISO code>] [-version]

-user <user name>	Name of the user
-password <password>	User password
-client <name>	Name of the client. If no client is specified, the import process for the default client is run.
-version	Version number
-datasource <file>	Data source to be used for the XML import.
-datasourcelist <file>	List of data sources to be used for the XML import.
-i <file1> [,<file2>...]	Import files. ZIP files are supported.

If the import is run in event format, -f and -m must be specified if no data source or data source list have been specified.

[-f <fragment file>]	Fragment file
[-m <mapping file>]	Mapping file
[-extractattributes <attribute file>]	Generates an XML file with all attribute types that occur in the input file(s), but that have not been defined yet.
[-autoextendattributes]	Before XML import, all attribute types are imported that occur in the input file(s), but that have not been defined yet.
[-extractmergeattributes <mergeattribute file>]	Generates an extract of the merge configuration file with all new attributes that are to be retained as process attributes when merging fragment instances.

The **protocoloptions** option can consist of the following instructions:

-protocolfile <file name>	Logging to file <file name>
-information {yes|no|default}	Logging of information
-warning {yes|no|default}	Logging of warnings
-error {yes|no|default}	Logging of errors
-language <ISO code>	Log output language

3.5.1. runxmlimport arguments

-version

The version of the PPM software and the database schema are output on the console. Other arguments are ignored.

-user <user name> -password <password>

Specify the user name and the password of the PPM user who is executing the import. The user must have the **Data import** function privilege.

-client <client name>

Specify the PPM client in whose database schema the imported process fragments are to be saved. If you do not enter anything here, the default client (**Default**) is used.

-i <Name of XML files>

Specify any number of files to be imported. The names are allowed to contain placeholders. **?** is a placeholder for a single character, ***** for any number of characters. The file name must not begin with a minus sign.

Example: The argument **-i <Path>\?ata*.xml** reads all files in the **<Path>** directory with the extension **<.xml** that begin with any character followed by the string **<ata** and then any other string.

If the specified files are a ZIP archive, all XML files in the ZIP archive are read regardless of the folder structure of the archive.

-m <name of mapping file> -f <name of fragment file>

Specify the type of import. If you specify the **-f** and **-m** arguments, the PPM system event format is used automatically. If you do not specify these arguments, graph format is used.

-datasource <file name>

The XML elements **data**, **fragments**, and **mapping** indicated in the file specify the XML files to be used and replace the **-i**, **-f**, and **-m** command line parameters. The format of the XML file is described in the document type definition **datasource.dtd**. PPM Customizing Toolkit uses this file for configuration and to extract data sources.

The file is located under
<installation directory>\ppm\server\bin\agentLocalRepo\.unpacked\<installation_time>_ppm-client-run-prod-<version>-runnable.zip\ppm\dtd

The following examples illustrate the contents of a data source file for data import in graph and system event format:

Data source file **datasource.xml** for graph format

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datasource SYSTEM "datasource.dtd">
<datasource name="Sales" type="GRAPH">
<description name="default_description"
language="de">Demo-Datenbankinhalt</description>
<description name="default_description"
language="en">Demo database content</description>
<data>C:\ARIS\ppm\server\bin\work\data_ppm\custom\umg_en\data\umgsales_umg_en.zi
p</data>
<eventattributetypes />
</datasource>
```

Data source file **datasource.xml** for event format

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datasource SYSTEM "datasource.dtd">
<datasource name="CustomerService" type="EVENT">
<description name="default_description"
language="de">Demo-Datenbankinhalt</description>
<description name="default_description"
language="en">Demo database content</description>
```

```
<data>C:\ARIS\ppm\server\bin\work\data_ppm\custom\umg_en\data\CS\csdemodata.zip<
/data>
<fragments>C:\ARIS\ppm\server\bin\work\data_ppm\custom\umg_en\xml\CS\CustomerSer
vice_Fragments.xml</fragments>
<mapping>C:\ARIS\ppm\server\bin\work\data_ppm\custom\umg_en\xml\CS\CustomerServi
ce_Mapping.xml</mapping>
<eventattributetypes />
</datasource>
```

If you specify the parameter **-datasource** in the command line, the parameters **-i**, **-f**, and **-m** are not considered.

For more detailed information on configuring data sources, refer to the technical reference **PPM Process Extractors**.

-datasourcelist <file name>

With the **-datasourcelist** argument, you can import multiple data sources simultaneously, except for data sources of the **GRAPH** type. The import corresponds to multiple importing using the **-datasource** argument.

See chapter Import multiple data sources [58].

-autoextendattributes

If you specify this switch in the command line, new attributes will be identified and the attribute configuration in the PPM system extended accordingly. If you have enabled the **automapping** option in the data source configuration used, new attributes identified will be transferred to the specified objects in the fragment definition.

-extractattributes <File name>

Attributes contained in the XML import files that are not known in the PPM system are identified and written to the specified file. The file generated is DTD-compatible and can be imported using the **runppmconfig** command line program to extend the PPM attribute configuration. If you use the **-extractattributes** argument without the **-autoexentattributes** switch, no XML import data will be imported.

-extractmergeattributes <File name>

This argument writes attributes transferred to process instances using the **automapping** option to the specified file. No XML import files are imported.

Log options

These arguments can be used to limit the log output. Error messages resulting in program abortion will always be output in the console.

3.6. Import multiple data sources

Multiple data sources can be imported simultaneously by means of the **-datasourcelist <file>** argument (see chapter runxmlimport command line program [55]). To import multiple data sources a configuration file is available in which you can specify a list of data sources. During an XML import, the data of the data sources specified in the configuration file are imported consecutively, just as if the XML import was called consecutively multiple times using the -**datasource <file>** argument. The sequence of the data source import is specified in the configuration file.

The configuration file must match the **datasourcelist.dtd** DTD, which looks as follows.

```
<!ELEMENT datasourcelist (datasource*)>
<!ELEMENT datasource (#PCDATA)>
<!ATTLIST datasource
name ID #REQUIRED
type (EVENT | MYSAP | JDBC | CSV ) #REQUIRED
>
```

You need to specify an ID for each data source, that is, the name of the data source also used in PPM CTK, the data source type, and the path to the data source file.

An XML file can look as follows.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE datasourcelist SYSTEM "datasourcelist.dtd">
<datasourcelist>
<datasource name="CLEARING" type="EVENT">M:/ARIS/ppm/server/bin/work/data_ppm/
custom/umg_en/xml/CLEARING/CLEARING.xml</datasource>
<datasource name="BILLING" type="MYSAP"> M:/ARIS/ppm/server/bin/work/data_ppm/
custom/umg_en/xml/BILLING/BILLING.xml</datasource>
<datasource name="SHIPMENT" type="JDBC"> M:/ARIS/ppm/server/bin/work/data_ppm/
custom/umg_en/xml/SHIPMENT/SHIPMENT.xml</datasource>
<datasource name="MATERIAL_DOCUMENT" type="CSV">M:/ARIS/ppm/server/bin/work/
data_ppm/custom/umg_en/xml/MATERIAL_DOCUMENT/MATERIAL_DOCUMENT.xml</datasource>
M:/ARIS/ppm/server/bin/work/data_ppm/custom/umg_en/xml/PURCHASE_PROCESS/
PURCHASE_PROCESS.xml</datasource>
</datasourcelist>
```

Error behavior

If the XML import is called via a valid configuration file that does not contain any data sources the import ends without outputting an error message.

If the XML import is called via a configuration file containing multiple data sources, and if an error occurs during the import of a data source that leads to cancelation of this import, the import continues with the next data source file. This means that the cancelation of the import of one data source does not result in the cancelation of the overall import.

If an error occurs during the import of at least one data source from a configuration file, which has so far lead to an exit error status (that is, "-1") during the import of individual data sources, the import using that configuration file will also return this exit error state.

3.7. Re-importing the same data

In the PPM system, repeated importing of the same source data always leads to the same unique result.

3.7.1. Graph format

When re-importing the instance data using graph format, the process instances are uniquely identified by the **AT_EPK_KEY** attribute. Existing process instances are overwritten with imported process instances with the same attribute value.

3.7.2. System event format

When re-importing the instance data using system event format, identical objects are automatically overwritten. Identical objects are identified by an identical internal object key, which is calculated during the import and is stored in the **AT_INTERNAL_OBJECT_KEY** object attribute. Rules for the calculation of the object keys are specified in the **internalobjectkeyrules** XML element in the file **keyrules.xml**. In case of identical objects, the last object imported is transferred into the process instance.

Example

The file extract below defines rules for the calculation of the object key for functions and events. Events are identified as identical if the values of the **AT_OBJNAME_INTERN** and **AT_MERGE_KEY_1** attributes match. Functions are identified as identical if the values of the **AT_OBJNAME_INTERN** and **AT_END_TIME** attributes match.

```
...
<internalobjectkeyrule>
<refobjects>
<refobject objecttype="OT_EVT"/>
</refobjects>
<keyparts>
<keypart attributetype="AT_OBJNAME_INTERN"/>
<keypart attributetype="AT_MERGE_KEY_1"/>
</keyparts>
</internalobjectkeyrule>
<internalobjectkeyrule>
<refobjects>
<refobject objecttype="OT_FUNC"/>
</refobjects>
<keyparts>
<keypart attributetype="AT_OBJNAME_INTERN"/>
<keypart attributetype="AT_END_TIME"/>
</keyparts>
</internalobjectkeyrule>
...
```

Warning
Define internal object key rules, which ensure that identical object keys are calculated for identical objects in the process instance, so that these objects are overwritten if re-imported.

4. Import of process instance-independent data

This chapter describes the import interface for importing process instance-independent measure and dimension values.

Process instance-independent data takes aspects into account that are not process-oriented, for example, commercial fixed costs, customer satisfaction, or financial measures, which characterize the financial view of a company.

Process instance-independent measures

The values of process instance-independent measures (see chapter Process instance-independent measures [61]) are measure values that are not calculated based on process instance data. They are imported directly with the referenced dimension values and without reference to process instance data.

Process instance-independent measures can be used as a basis for user-defined measures and thus be combined with process instance-dependent measures.

Process instance-independent dimension data

You can import into a PPM system values of keys and descriptions for one-level, two-level, and n-level text dimensions in advance that is, before the actual data import (see chapter Dimension values [72]).

4.1. Process instance-independent measures

The following chapters describe the configuration of import data formats (XML, CSV, XLS) for process instance-independent measures as well as the import and export of process instance-independent measure values.

A precondition for this is the existence of the corresponding process instance-independent measures in the PPM system. If you want to define and register process instance-independent measures, please refer to the technical reference **PPM Customizing** for instructions.

4.1.1. Data import formats

You can import import data of process instance-independent measures in the following formats:

- XML (default)
- CSV
- XLS

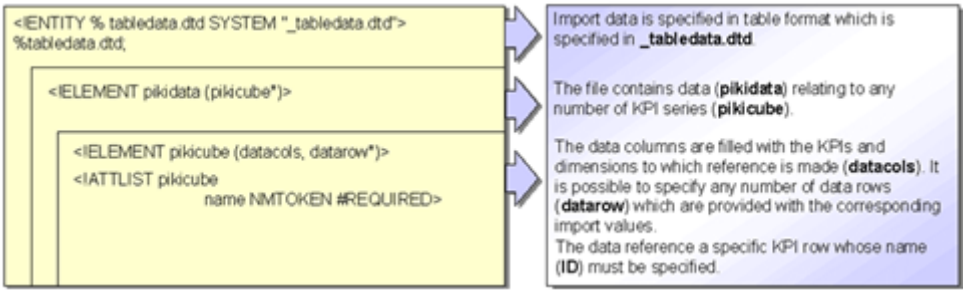
Regardless of the import data format, you must always specify all key dimensions (**iskeydimension="TRUE"** attribute of the **refdim** element in the definition of the data series, see technical reference **PPM Customizing**)

and at least one of the process instance-independent measures of the data series. You can optionally specify further process instance-independent measures or non-key dimensions of the relevant PIKI cube in the import data structure.

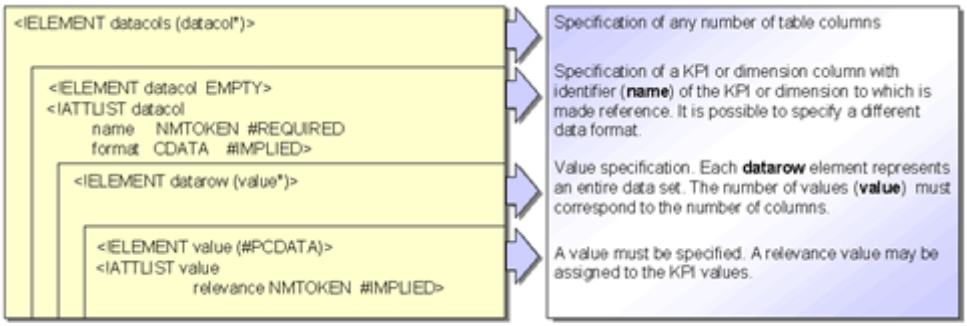
4.1.1.1. XML format

The XML data import format for process instance-independent measures is preset by the following document type definitions:

DTD **pikidata.dtd** (referencing of data series as import data)



DTD **_tabledata.dtd** (table format for importing process instance-independent data):



Thus, import data of process instance-independent measures in XML format have the following general structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pikidata SYSTEM "pikidata.dtd">
<pikidata>
  <pikicube name="...">
    <datacols>
      <datacol name="..."/>
      ...
    </datacols>
    <datarow>
      <value relevance="...">...</value>
       ...
    </datarow>
    ...
  </pikicube>
  ...
</pikidata>
```

ELEMENT and ATTLIST pikidata	Description
pikidata	List of import data on any number of process instance-independent data series (PIKI cube)
pikicube	Data series into which the specified data is to be imported
name	Data series identifier. Must match the name of the PIKI cube (**pikicube name** XML attribute) specified in the measure configuration.

ELEMENT and ATTLIST datacols	Description
datacols	Import data structure (table columns of the data series)
datacol	Specification of a table column. A column is specified for each process instance-independent measure and each relevant referenced dimension.
name	Table column identifier. Must match the names of the process instance-independent measures (**pikidef name**) specified in the measure configuration and the names of the referenced dimensions (**refdim name**).
format	Optional import format for column values. Supported formats are floating point numbers, time and time of day values.
datarow	Specific import data for the data series in the form of a data row. The sequence of import values (**value** elements) must match the column sequence (**datacols**) so that each criterion of the data series (process instance-independent measure or referenced dimension) can be assigned a unique value.
value	Data row column value to be imported (value of a referenced dimension or a process instance-independent measure)
relevance (optional)	Relevance value referring to the corresponding value of a process instance-independent measure

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pikidata SYSTEM "pikidata.dtd">
<pikidata>
<pikicube name="PIKICUBE_COSTS">
<datacols>
<datacol name="OVERHEAD_COSTS"/>
<datacol name="PROCESSTYPE"/>
<datacol name="TIME" format="MMM yyyy"/>
<datacol name="MATERIAL"/>
</datacols>
<datarow>
<value relevance="100">1000 EUR</value>
<value>Order processing\Car industry</value>
<value>June 2000</value>
<value>NaviSet B6{Navigation system}</value>
</datarow>
<datarow>
...
</datarow>
...
</pikicube>
...
</pikidata>
```

Specification of the import data structure (XML elements datacols and datacol)

The XML-Element **datacols** specifies the identifiers of the process instance-independent measures and referenced dimensions (**datacol name="..."**) that the data import refers to for the relevant data row. The

import data structure is specified in a simple table format in which each criterion of the data series (process instance-independent measure or referenced dimension) represents a table column.

Data import

The actual import values are extracted from the **datarow** XML elements. Each of these elements represents a data row that is imported. Each specified **value** element corresponds to the matching **datacol** element of the specified import data structure.

The sequence of the values (**value** XML element) in a data row (**datarow** XML element) must match the sequence of columns (in the **datacols** XML element) in the import data structure.

Example 31.

In the XML attribute **format="MMM yyyy"**, the data format for the value to be imported for the referenced dimension **TIME** is specified. Format information is optional.

Specifying a descriptive text in curly brackets **{ }** means that the values for the one-level dimension **MATERIAL** are expected in the form **<Identifier>{<Descriptive text>}**. Specifying a descriptive text is optional, as only the identifier is crucial for the dimension reference.

The example data set has a relevance value for the process instance-independent measure **OVERHEAD_COSTS**. If you do not specify a relevance value, **relevance="1"** is the default value.

If no unit is specified for a data value, by default the base unit for the attribute data type is used on which the process instance-independent measure or referenced dimension is based.

Example 32.

There are two different formats for importing values of two-level dimensions:

- Value definition in a row:

Example 33.

The values are separated by a backslash (****).

```
<datacols>
  ...
  <datacol name="PRINCIPAL"/>
  ...
</datacols>
...
<datarow>
  ...
  <value>DE{Germany}\0000000001{Becker}</value>
  ...
</datarow>
...
```

- Value definition in two rows:

Example 34.

The values are specified in two XML elements that directly follow one another.

```
...
  <datacols>
    ...
    <datacol name="PRINCIPAL"/>
    <datacol name="PRINCIPAL"/>
    ...
  </datacols>
  <datarow>
    ...
    <value>DE{Germany}</value>
    <value>0000000001{Becker}</value>
    ...
  </datarow>
  ...
```

4.1.1.2. CSV format

For importing in CSV format, the following special features apply:

- A CSV file can only contain data for one data series.
- The values specified in a **CSV file** must not contain the specified data separator. The data separator is specified using the **-csvchar "<Character>"** option in the **runpikidata** command line program.

Example 35.

This example shows the importing of **CSV** format with row numbers.

```
1 PIKICUBE_COSTS
2 OVERHEAD_COSTS(.,);OVERHEAD_COSTSNUM;TIME(MM.yyyy);
```

```
1 PIKICUBE_COSTS
2 OVERHEAD_COSTS(.,);OVERHEAD_COSTSNUM;TIME(MM.yyyy);PROCESSTYPE;PROCESSTYPE
3 1.000,00 EUR;100;05.2000;Order processing;Car industry
4 1.020,00 EUR;;06.2000;Order processing;Car industry
...
```

Explanation

- Row 1: Name of the data row (for example, **PIKICUBE_COSTS**)
- Row 2: Specification of the import data structure

  The individual table columns of the data series are separated by semicolons. Format information can be specified after the column name in round brackets.

  The optional specification of the relevance must directly follow the relevant measure. The relevance column name is made up of the name of the measure supplemented by the string **NUM** (for example, **OVERHEAD_COSTSNUM**).
- From row 3: Import data

Each row represents a data series data row to be imported. The individual values are separated by the specified separator.

For the import values of the process instance-independent measure **OVERHEAD_COSTS**, a relevance value (**OVERHEAD_COSTSNUM**) of 100 (**1,000.00 EUR;100**) is specified in the first data row (row 3), the second data row (row 4) has no relevance value (**1,020.00 EUR;;**). In this case, the relevance value is automatically set to **1**.

The values for two-level dimensions (in the example: **PROCESSTYPE**) are imported using two separate, successive columns with the same name. The first column contains the rough value, the second the detailed value.

The optional format **MM.yyyy** for the referenced dimension **Time** consists of an integer month number followed by a four-digit year number. The individual field values are separated by a full stop (see Data formats [42]).

4.1.1.3. XLS format

You can import data for process instance-independent data rows from Excel files into the PPM system. For importing in XLS format, the following special features apply:

- An Excel file can only contain data for one data series on each worksheet. All worksheets from the Excel file are imported.
- When importing a worksheet, the entire area of the sheet is always imported. Columns and rows with no content are skipped.
- The names of the worksheets to be imported can be specified with the optional command line parameter **-sheet**. If the name contains spaces, you need to enclose the name in quotation marks. To import several worksheets from a file, enter the names separated by spaces.
- The cell format for dimension columns must have the data type **Text**. The format of cells in measure columns must be of the type **Text** or **Number**. Formatting rules (for example, separators for decimals and thousands) can only be specified for measure values in text format.

Implicit positioning

In principle, the data format corresponds to that in CSV format. The information in the worksheet is expected in the following order:

- Cell A1: Name of the data series
- Cell A2 and following cells (row 2 only): Definition of data structure
- Cell A3 and following cells (all rows): Data values

	A	B	C
1	PC_CUSTSAT		
2	CUST_SATISFACT(,)	TIME(MM.yyyy)	PRINCIPAL
3	7.60	01.2004	DE{DEUTSCHLAND}
4	6.60	02.2004	US{USA}
5			
6			

2004 / 2005 / 2006

Explicit positioning

The name of the data series is specified in cell A1. In cell B1, the area is specified that contains the import data structure information, and cell C1 specifies the area that contains the actual import data. The positioning has the general format

<Position of start cell>[:<Position of end cell>]

If you only specify the position of the start cell, for example, C8, the data from all subsequent columns will also be imported. In this case, the specification corresponds to an infinite area of the table.

If you specify the same row number when specifying the position of the range of import data to be extracted, for example, C9:E9, the specified columns are extracted to the end of the worksheet and an unlimited number of data rows is thus imported. To import a precisely limited range of values, you must specify the first and last data row, for example, C9:E10.

Example 36.

	A	B	C	D	E
1	PC_CUSTSAT	C8:E8	C9:E10		
2					
3					
4					
5					
6					
7					
8			CUST_SATISFACT(,.)	TIME(MM.yyyy)	PRINCIPAL
9			7.60	01.2005	DE{DEUTSCHLAND}
10			6.60	02.2005	US{USA}
11					

The import data structure of the process instance-independent data series **PC_CUSTSAT** is specified in cells C8 to E8, and the actual values to be imported are specified in cells C9 to E10.

You can use the remaining Excel table area for comments, etc. This area is ignored during the import.

> **Warning**
> To use explicit positioning, you must observe the basic sequence: You must specify the row for the import data structure before the rows containing the data values.

Relative starting point

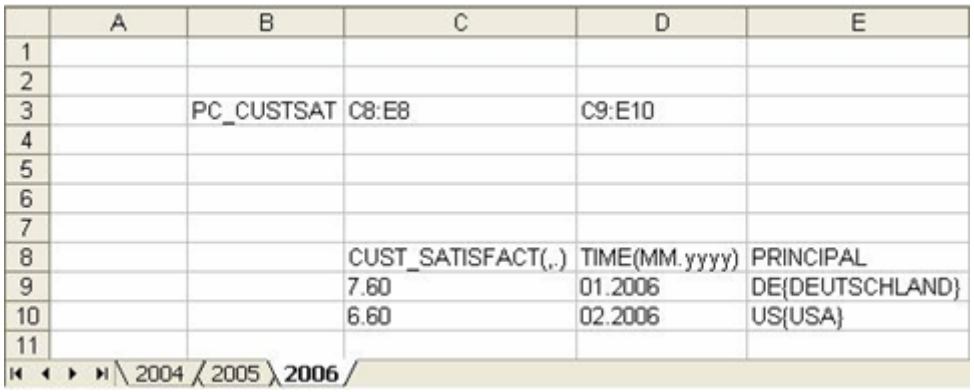
The name of the data series is specified in a position different from the implicit position (cell A1). The position of the cell in the worksheet is specified by the **sheet** parameter and the name of the worksheet. The syntax is

-sheet <Worksheet name>:<Cell>.

Example 37.

	A	B	C	D
1				
2				
3		PC_CUSTSAT		
4		CUST_SATISFACT(,.)	TIME(MM.yyyy)	PRINCIPAL
5		7.60	01.2005	DE{DEUTSCHLAND}
6		6.60	02.2005	US{USA}
7				
8				

2004 **2005** 2006



	A	B	C	D	E
1					
2					
3		PC_CUSTSAT	C8:E8	C9:E10	
4					
5					
6					
7					
8			CUST_SATISFACT(,)	TIME(MM.yyyy)	PRINCIPAL
9			7.60	01.2006	DE{DEUTSCHLAND}
10			6.60	02.2006	US{USA}
11					

2004 / 2005 \ **2006** /

For both examples, the data is imported using the following command:

```
runpikidata -user <username> -password <password> -mode import -format XLS
-file excelpikidata.xls -sheet 2006:B3
```

4.1.2. Data reimport

Data import into process instance-independent data series **adds** new data rows and **overwrites** changed existing data rows.

If the data import is repeated, existing data rows are overwritten with changed, updated values of process instance-independent measures and referenced dimensions of the data series, and new data rows are added to the relevant data series. The PPM system recognizes existing data rows through a data series' key dimension values.

Key dimensions are determined by the **iskeydimension="TRUE"** attribute of the **refdim** element in the definition of the data series. For further information, please refer to the document **PPM Customizing**.

Example 38.

The following data row has already been imported into an existing data series:

Example 39.	Example 40.	Example 41.	Example 42.	Example 43.	Example 44.
D_COUNTRY*	D_PLANT*	D_DEPARTMENT*	D_RECORDED BY	SALES	COSTS
Example 45.	**Example 46.**	**Example 47.**	**Example 48.**	**Example 49.**	
Germany	Hamburg	42	Smith	400000	

Example 50.

Referenced dimensions marked with * are the key dimensions of the data series. Therefore, the value combination **Germany; Hamburg; 42** is the identifier of the displayed data row. The two process instance-independent measures **SALES** and **COSTS** are configured for the data series.

If data import is repeated, for example, with the values **Germany; Hamburg; 42; Huber;;280000**, the PPM system recognizes that the data row already exists and overwrites the value of the dimension **D_RECORDED BY** (**Schmidt**) with the changed value (**Huber**) and adds a value for the process instance-independent measure **COSTS**.

Example 51.	Example 52.	Example 53.	Example 54.	Example 55.	Example 56.
D_COUNTRY*	D_PLANT*	D_DEPARTMENT*	D_RECORDED BY	SALES	COSTS
Example 57.	**Example 58.**	**Example 59.**	**Example 60.**	**Example 61.**	**Example 62.**
Germany	Hamburg	42	Huber	400000	280000

Example 63.

The value of the process instance-independent measure **SALES** remains unchanged and is retained because no new value was imported during the repeat data import.

4.1.3. Export of values of process instance-independent data series

The values of process instance-independent measures can be exported in XML format using the **runpikidata** command line program. To do so, use the command line argument **-mode export** (see chapter runpikidata command line program [70]).

The data on all data series of the specified client are written to the specified file using the argument **-file**. To export particular data series, specify the names of the data series you want to export, separated by commas, using the **-pikicube** argument.

The PPM user performing the operation requires the **Data import** function privilege.

```
runpikidata -user system -password manager -client umg_en -mode export -file
pikidata -pikicube
PC_SALES_REVENUES
```

4.1.4. Deletion of values of process instance-independent data series

You can delete values of process instance-independent data series in the PPM system using the command line program **runpikidata**. To do so, use the command line arguments **-mode delete** and **-pikicube <cube name>[,<cube name>,...]** to specify the data series from which you want to delete values.

By default, all values of the specified PIKI cubes are deleted.

You can limit the data rows of a data series to be deleted by specifying a paramset that contains a filter on a referenced dimension of the data series, for example. To do this, enter the name of the paramset file using the **-ps** parameter.

If the delete paramset contains dimension filters that are not included as referenced dimensions in the definition of the specified data series an error message is output when you run the program. The PPM user performing the operation requires the **Data import** function privilege.

Example 64.

You want to delete import values of the data series with the internal identifier **PC_SALES** (PIKI cube Sales) in the time period of the 1st quarter in 2009.

In the analysis, set a delete paramset with the corresponding time filter and export it to a local XML file. Then call up the **runpikidata** command line program as follows:

```
runpikidata -client <ppmclient> -user <username> -password <password> -pikicube
PC_SALES -mode delete -ps <deleteparamset>.xml
```

4.1.5. runpikidata command line program

The data import or export of process instance-independent data series is executed using the command line program **runpikidata**:

```
runpikidata -user <user name> -password <password>

  [-client <name>]
-mode (import|export|delete)
[-compatible4]
-file <file1>[,<file2>...]
[-format {XML|CSV|XLS} [[-csvchar "<character>"] -encoding "<encoding name>"]]
-pikicube <cubename>[,<cubename>...]
-ps <filename>
[-sheet <name>[:<cell>] [<name>[:<cell>]]]
[-version]
[-language <ISO code>][ protocoloptions ]
[-recoveryfile {yes|no}]

-user <user name> Name of user
-password <password> User password
-client <name> Name of client If

  no client is specified
the server of the

  default client will be used.
-language <ISO code> Import language
-mode (import|export|delete) Import, export,
or deletion of data
-compatible4 Import of dimension values as

  in PPM 4.x without checking the

  refinement level
-file <file1>[,<file2>...] Import files (-mode import)

  or export files

  (-mode export).

  ZIP files are supported.
-format {XML|CSV|XLS} Data format for the import

  (default: XML)
-csvchar "<character>" Data separator for the

  "CSV" import format
```

```
  (Default: comma)
-encoding "<encodingname>" Encoding of the CSV file
-pikicube <cubename> Data series from which data

  [,<cubename>...] is to be deleted or

  exported (-mode delete|export)
-ps <filename> Paramset specifying the

  data to be deleted (-mode delete)
-sheet <name>[:<cell>]> Specifies Excel sheets for the

  "XLS" import format. The

  cell specifies the position from

  which the data is to be

  imported.
-recoveryfile {yes|no} Creates after successful

  import the new

  process analysis-relevant

  analysis server recovery files

  Default: yes)

-version Version number of the application

  and the database database schema

protocoloptions can consist of the following instructions:
-protocolfile <file name> Log to file

  <file name<
-information {yes|no} Logging of

  information
-warning {yes|no} Logging of warnings
-error {yes|no} Logging of errors
```

-version

The version of the PPM software and the database schema are output on the console. Other arguments are ignored.

-user <user name> -password <password>

With this parameter, you specify the user name and the password of the PPM user who is executing the import. The user must have the **Data import** function privilege.

-client <client name>

With this parameter, you specify the PPM client in whose database schema the imported process fragments are to be saved. If you do not use this option the default client is used.

-mode import|export|delete)

For example, with this parameter you specify that you want to import (**-mode import -file <file1>[,<file2>...]**) a source file (or several source files) including data of the selected import format (**-format {XML|CSV|XLS}**) or export (**-mode export -pikicube <cubename> -file <filename>**) data of a data series existing in the PPM system into an XML file, or that you want to delete entirely (**-mode delete -pikicube <cubename>[,<cubename>...]**) or in part (**-mode delete -pikicube <cubename> -ps <filename>**) data from the specified data series.

-compatible4

With the optional parameter, you specify that when importing values of referenced, multi-level text dimensions (**refdim**) the refinement level (**refinement**) of the import data will not take place, just as in PPM versions 4.x. This is necessary if dimension values of process instance-independent measure import data for PPM from version 5 exist with a refinement level other than the one specified in the definition of the data series (see technical reference **PPM Customizing**).

-file <file1>[,<file>,…]

With this parameter you specify the import file(s) and the path to the import file(s). The source file can be a ZIP file, which contains one or more import files with the same data format.

-format {XML|CSV|XLS}

With this parameter you specify the data import format used.

-csvchar <character>

With this parameter you specify the separator for data field values in CSV import format (default value is the comma).

-encoding "<encodingname>"

With this parameter you specify the name of the CSV encoding you want to use. The **-encoding** parameter is mandatory for the CSV format.

-sheet <name>[:<cell>]>

With this parameter you specify the name of the Excel worksheet from which you want to import the data. You can specify the names of multiple worksheets, separated by spaces. The **<cell>** argument can be used for each worksheet to specify the relative start position (cell containing the name of the data series) from which the data is to be imported.

-recoveryfile {yes|no}

With this parameter you specify if analysis server recovery files relevant for process analysis are to be created after import or deletion of values of process instance-independent data series. The default value is **yes**.

4.2. Dimension values

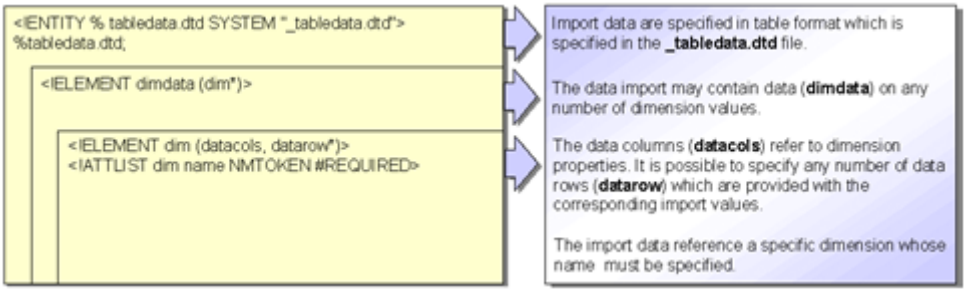
For one-, two-, and n-level text dimensions, you can import dimension values (especially comprehensive level descriptions) into the PPM system before the actual PPM import. Dimension values consist of a mandatory ID and an optional description.

Advantages:

- You can create planned value definitions in PPM before process instances are imported.
- You can significantly reduce the data volume of the process instances to be imported during the actual PPM import by not specifying the level descriptions already imported.

4.2.1. XML format

The format of the XML file is specified by the following DTD:



Warning
The name of the data columns for the key and description of a dimension level must start with the prefix **LEVEL** and end with the suffix ID (key) or DESC (description). Additionally, please use consecutive numbers starting with **1** to specify the level. Otherwise, the import will be canceled and an error message is displayed.

The following table illustrates the assignment of data columns to the dimension values (**<n>** represents the consecutive numbering of the level):

dimdata configuration	Dimension configuration	Example
LEVEL<n>_ID	ID of the n-th dimension level	**LEVEL5_ID**
LEVEL<n>_DESC	Description of the n-th dimension level	**LEVEL5_DESC**

Example 65.

For the two-level dimension **Sold-to party** (**PRINCIPAL**), three values are imported:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dimdata SYSTEM "dimdata.dtd">
<dimdata>
<dim name="PRINCIPAL">
<datacols>
<datacol name="LEVEL1_ID"/>
<datacol name="LEVEL1_DESC"/>
<datacol name="LEVEL2_ID"/>
<datacol name="LEVEL2_DESC"/>
</datacols>
```

```
<datarow>
<value>DE</value>
<value>Germany</value>
<value>0000000003</value>
<value>Becker</value>
</datarow>
<datarow>
<value>FR</value>
<value>France</value>
<value>0000000092</value>
<value>Leclerc</value>
</datarow>
<datarow>
<value>EN</value>
<value>United Kingdom</value>
<value>0000000027</value>
<value>Crichton</value>
</datarow>
</dim>
</dimdata>
```

Warning
When importing values for multi-level text dimensions, you always need to indicate the key of the first dimension level (**LEVEL1_ID**).

4.2.2. CSV format

For importing in CSV format, the following special features apply:

- A CSV file can only contain data for one dimension.
- The dimension values specified in a **CSV file** must not contain the specified data separator. The data separator is specified using the **-csvchar "<Character>"** option in the **rundimdata** command line program.

Example 66.

This example shows the importing of **CSV** format with row numbers.

```
1 PRINCIPAL
2 LEVEL1_ID;LEVEL1_DESC;LEVEL2_ID;LEVEL2_DESC
3 DE;Germany;0000000003;Becker
4 FR;France;0000000092;Leclerc
5 UK;United Kingdom;0000000027;Crichton
...1 PRINCIPAL
```

Explanation

- Row 1: Dimension identifiers

- Row 2: Definition of data structure

  The individual columns are separated by a semicolon. The column names are keywords, which must not be changed.
- From row 3: Data values

Each row contains one record. The individual values are separated by the specified separator. The values specified must each match the data types specified in the attribute type configuration. From left to right, the column entries represent: Country code, Name of country, Principal ID, Principal name

4.2.3. Default and replacement values

If you specified replacement or default values for the key or the description of a dimension level in the measure configuration and want to use these values for importing with **rundimdata**, you need to leave the corresponding **value** elements in the XML import file or column values in the CSV file empty.

For more information about using default and replacement values in text dimensions, refer to the **PPM Customizing** Technical Reference.

4.2.4. Data reimport

In principle, the import of process instance-independent dimension values (**rundimdata -mode import**) is additive, that is, dimension values that do not exist are added in the PPM system (only for the two-level text dimension **PROCESSTYPE** an additive import is impossible).

If you additionally specify the command line option **-overwrite**, existing descriptions of dimension levels are overwritten with the imported, changed descriptions.

If you want to delete the descriptions of particular dimension levels, import empty descriptions for these dimension levels in the overwrite mode using **-mode import -overwrite**.

4.2.5. Delete dimension values

To delete dimension values, use the **-replace** option of the **rundimdata** command line program. This marks all dimension values that are not included in the import file for deletion. To delete the data permanently, the analysis server must be completely reinitialized. Therefore, the recovery files of the analysis server relevant for process analysis are recreated by default after import. After the import, the command line program outputs a corresponding message.

If you want to delete all dimension data of a dimension, import a semantically empty file.

The option **-replace** to replace or delete dimension values implies the option **-overwrite**. Therefore, you must not specify the option **-overwrite** in the command line when using the option **-replace**.

4.2.6. rundimdata command line program

Process instance-independent dimension values in XML and CSV format are imported using the **rundimdata** command line program:

```
rundimdata -user <username> -password <password> [-client <name>]
-mode import
-file <file1>[,<file2>...]
[-format {XML|CSV} [[-csvchar "<character>"] -encoding "<encoding name>"]]
[-overwrite |-replace]
[-recoveryfile {yes|no}]
[-version]
[-language <ISO code>][ protocoloptions ]

-user <user name> Name of user
-password <password> User password
-client <name> Name of client If no client is specified,
the server of the default client will be used.
-language <ISO code> Import language
-mode import Import of data
-recoveryfile {yes|no} After import into the analysis server
the recovery files relevant for
process analysis are recreated.
-file <file1>[,<file2>...] Import files. ZIP files are supported.
-format {XML|CSV} Data format for the import (default: XML)
-csvchar "<character>" Data separator for "CSV" import format
(default: comma)
-overwrite Overwriting dimension descriptions
-replace Delete existing dimension values that
have been changed by a data import for dimensions
or were supplemented.
-encoding "<encodingname>" Encoding of the CSV file
-version Version number of the application and the
database schema

protocoloptions can consist of the following instructions:
-protocolfile <file name> Logging in the file <file name>
-information {yes|no|default} Logging of information
-warning {yes|no|default} Logging of warnings
-error {yes|no|default} Logging of errors
```

-version

The version of the PPM software and the database schema are output on the console. Other arguments are ignored.

-user <user name> -password <password>

With this parameter, you specify the user name and the password of the PPM user who is executing the import. The user must have the **Data import** function privilege.

-client <client name>

With this parameter, you specify the PPM client for which you want to save the imported dimension values. If you do not use this option the default client is used.

-mode import

With this parameter you specify that the source files are to be imported with process instance-independent data.

-file <file1>[,<file2>...]

With this parameter you specify the import file(s). The source files can be ZIP files containing one or more XML file(s) of the same data format.

-encoding "<encodingname>"

With this parameter you specify the name of the CSV encoding you want to use. The **-encoding** parameter is mandatory for the CSV format.

-recoveryfile {yes|no}

With this parameter you specify if analysis server recovery files relevant for process analysis are to be created after import of process instance-independent dimension values. The default value is **yes**.

You can boost the performance of multiple sequential dimension data imports by suppressing the creation of the analysis server recovery files for all previous imports. To do so, specify the option **-recoveryfile no**. Only the last import creates the analysis server recovery files relevant for process analysis if the option is missing or if **-recoveryfile yes** was specified. Please make sure that the analysis server recovery files relevant for process analysis are created in any case once the import is complete. You can force the creation of all recovery files using the command line call **runppmadmin** with the option **-recoveryfile force**.

-format {XML|CSV}

With this parameter you specify the data format used.

-csvchar "<character>"

With this parameter you specify the separator for data field values in CSV import format (default value is the comma).

-overwrite

With this parameter you specify that existing **descriptions** of text dimension levels will be overwritten with changed (also empty) values. Text dimension **keys** (IDs) cannot be overwritten.

-replace

With this parameter, you can overwrite already imported dimension data. Dimensions not included in the import file are deleted. This parameter implies the parameter **-overwrite**. After the import, the analysis server recovery files relevant for process analysis are recreated. (default behavior)

4.3. Data analytics

In addition to process, function, and interaction analyses, you can now evaluate process-independent data using Data analytics. Data analytics enables the analysis of comprehensive data in table format, consisting of multiple linked tables. In Data analytics, analysis criteria, that is, dimensions and measures, are based on the table structure of the data basis, with each table column representing an analysis criterion.

The technical documentation **PPM Data Analytics** provides information on the import of Data analytics data.

5. How to handle large EPCs

5.1. Import large EPCs

If the maximum number of functions allowed per EPC is exceeded during the import of large EPCs an error message is displayed. However, the import is not aborted.

The threshold for the maximum number of functions allowed in an EPC is controlled by the configuration parameter **KI_EPC_FUNCTION_COUNT_THRESHOLD** defined in the file **EpkImport_settings**. The default value for this parameter is **500**. It applies if the parameter is missing or has a value of **<= 0**. If the parameter falls back to the default value a corresponding warning is written to the log.

As soon as a large EPC exists in the database XML import, PPM import, and process import are blocked in **DEFAULT** mode. A corresponding error message is output and import is canceled. Process import in **RECOVERIMPORT** mode is not affected.

To resolve this issue you can set the configuration parameter to a sufficient value. However, this can lead to the system requiring increased main memory. Alternatively, you can delete the EPC that causes the problem (see following section).

5.2. Delete large EPCs

You can delete a large EPC using the **runppmdelete** command line program. This applies only if you have defined a process measure for the number of functions in a process instance. If you did so, you define a paramset as follows for the command line program.

- The ParamSet contains the number of functions **only** as a filter
- The ParamSet contains the number of processes as a measure.
- The ParamSet contains a time filter.
- The ParamSet contains a process type filter or a process type group filter.

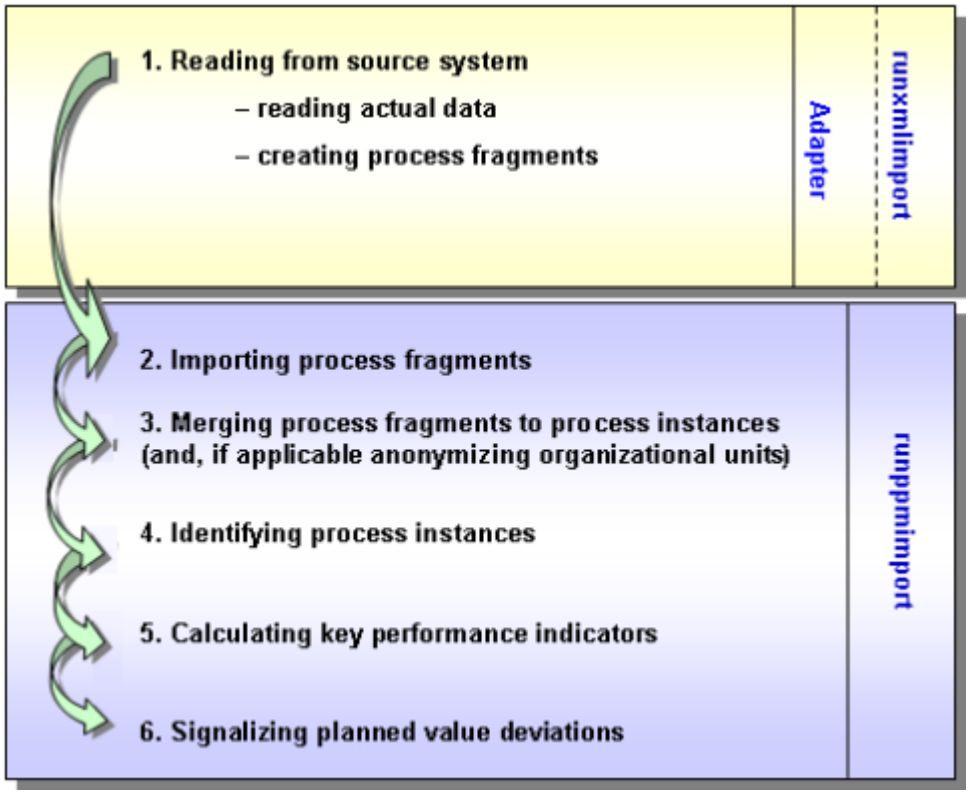
The measure can also be defined at a later time (that is, when imports are already blocked). In this case, the EPCs need to be recalculated using the **runppmimport –keyindicator new** command line program.

6. Appendix

6.1. Design of a Process Warehouse

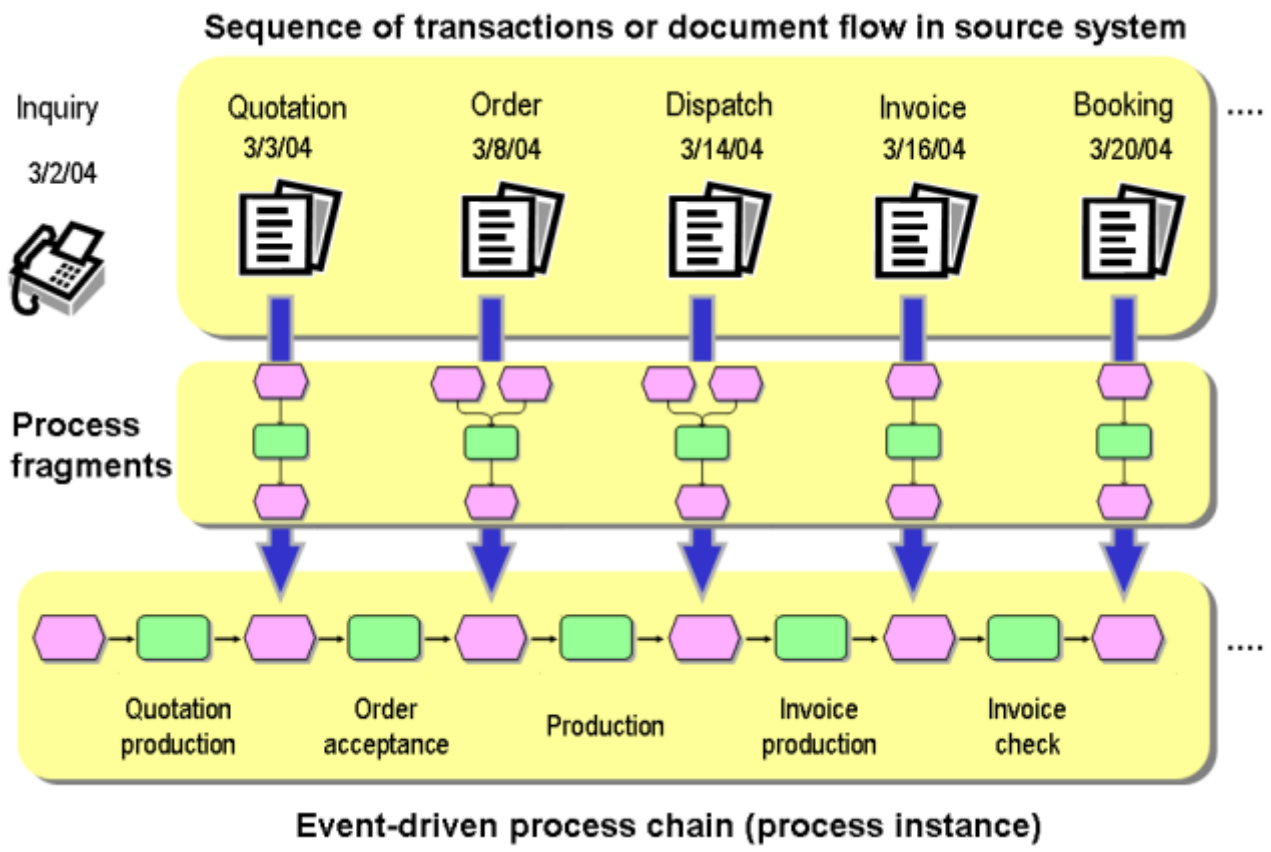
This chapter provides an overview of the design of a Process Warehouse in PPM and then gives a brief explanation of each individual step. It is assumed that the necessary installation and configuration of the PPM system has been completed.

The design of a Process Warehouse comprises the following steps:



1. First, the actual data is extracted from the source system and provided in the form of process fragments for importing.
2. The process fragments are imported.
3. The next step involves searching all imported data for the fragments belonging to each business event and merging them into a process instance. Object attributes are copied to the process instance. When merging the process fragments, information about the actual user can be made anonymous.
4. The process instances generated are then classified: process instances of the same kind are assigned to process types, which are in turn summarized into process type groups.
5. For each process instance, the defined measures are calculated and stored in info cubes.
6. Exceeding of planned values is checked and, if necessary, signaled.

Designing a Process Warehouse therefore involves generating process fragments from sequences of transactions and document flows from the source system and merging them into process instances. These process instances are used as reference objects for analyses and assessments of process performance. They can be displayed as process models in PPM.



The event-driven process chain (EPC) consisting of a chain of objects is used to represent a process instance. Both objects and the process instance itself can have attributes, in which the instance data is saved. This data is used to classify the process instances into types and to calculate the measures. The calculated measures are in turn saved in attributes of the objects and process instances. The attributes are therefore the actual carriers of information in the PPM system. The available attributes are defined in the PPM configuration. The configuration is composed of default attributes, such as **Process identifier** and **End time** and freely definable system-specific attributes.

Additional information about EPCs:

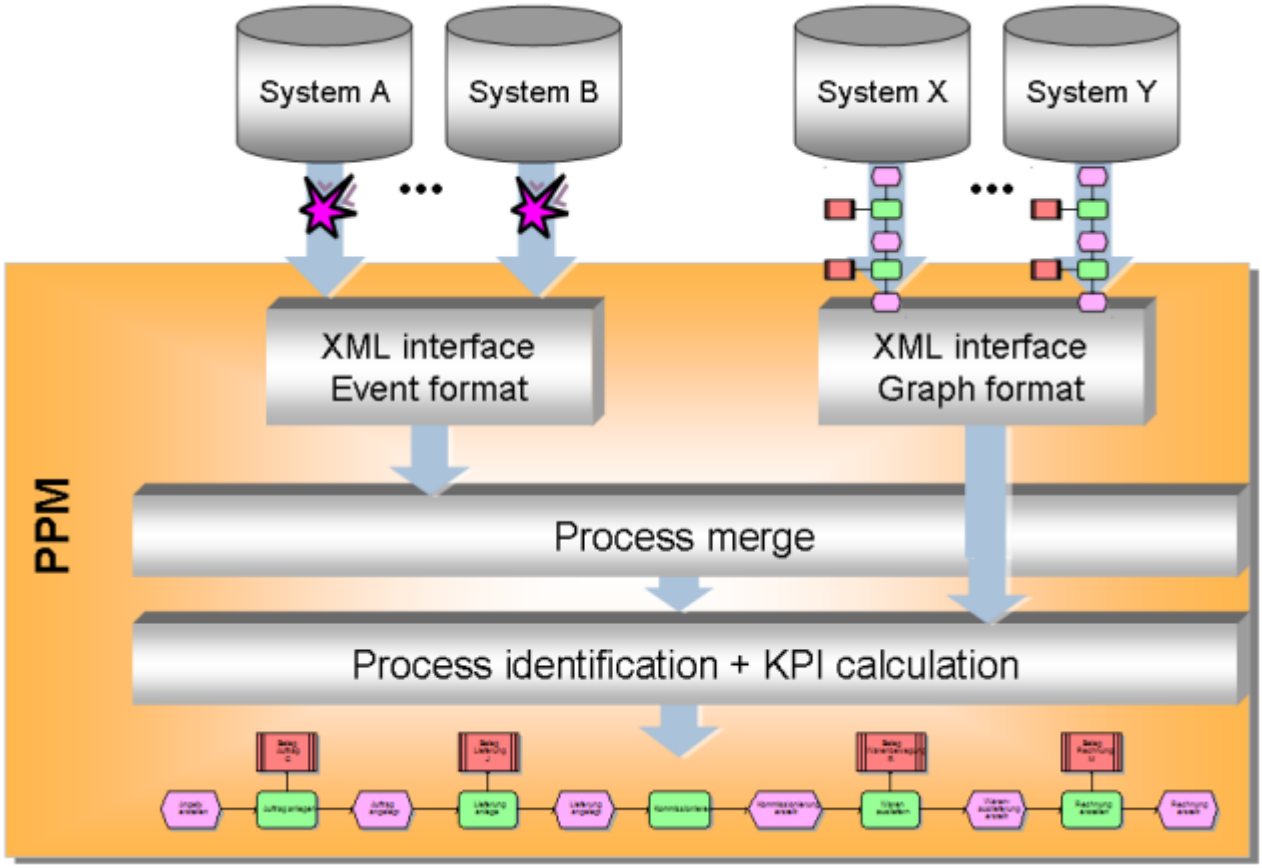
An event-driven process chain (EPC) is a model type developed by Prof. Scheer to graphically describe the chronological sequence of a performance delivery process. It is based on the following assumptions:

- Each activity within a process is triggered by a commercially relevant change of state of an information object. Each activity can result in a commercially relevant change of state of an information object.
- The state of a business-relevant information object is defined graphically by an object of the **Event** type.
- Objects of the **Function** type are used for the graphic representation of activities. Linking events and functions in series and connecting these objects with directed connections represents the control flow of the process graphically.
- As an event can trigger several functions and, conversely, a function can have several events as its result, AND, OR or exclusive OR connections (rules) are inserted at these branches. They illustrate the logical relationship that exists between the sequenced objects.
- Organizational units describe the groups of users executing a function.

6.1.1. Generate process fragments

Process data can be retrieved from the application systems in various ways.

Example 67.



With SAP R/3, special adapters access operational R/3 document data online and transform the SAP document flows into process descriptions. These adapters are only mentioned here for the sake of completeness.

With all other application systems, process data is imported into the PPM system offline via a generalized XML import interface using a file.

The XML import interface can process two different types of XML files: XML files in PPM graph format and in PPM system event format.

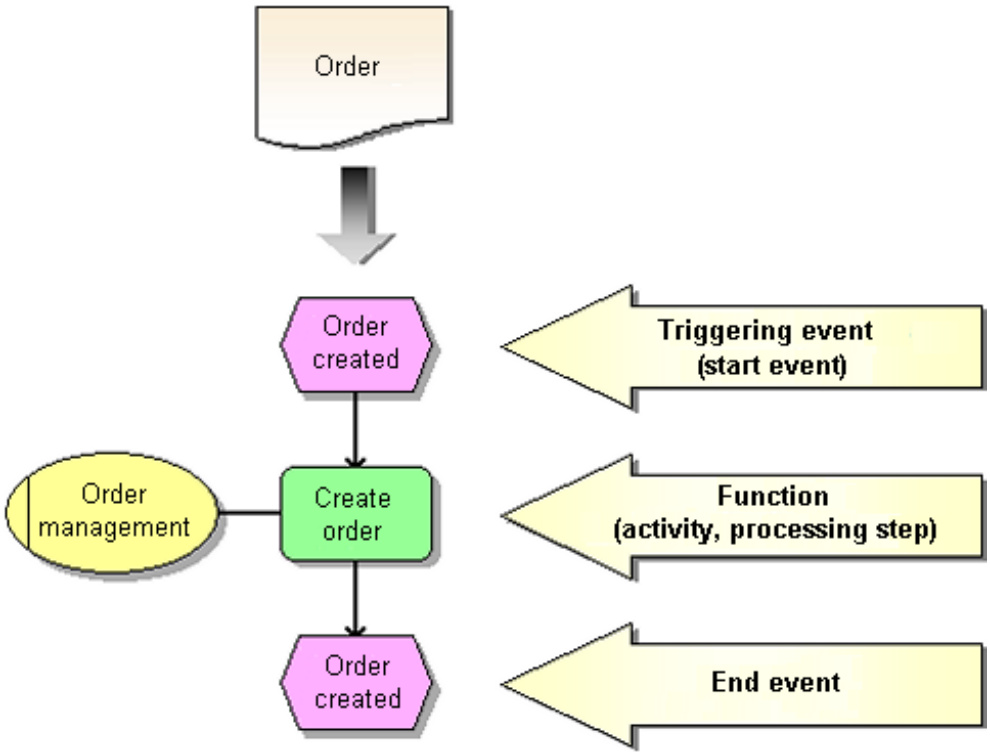
Graph format

Graph format is used to transfer already structured process data from process-oriented application systems (for example, workflow systems). The application-specific adapter generates XML files, in which process instances including their procedural logic are described in PPM graph format.

System event format

System event format is used for all activity-oriented application systems, in which the information making up the process (procedural logic) cannot be extracted. The system events are interpreted as process fragments.

Example 68.



A process fragment describes one part of an overall process. It contains at least one function with its triggering and resulting events. A process fragment can be interpreted as an individual operation, an activity or a transaction within an overall process. In addition to the chronological flow, a process fragment can also contain information about the processor of a function in the form of organizational units.

When extracting from source systems, every system event is assigned a process fragment using the mapping definition. The instance data for the system event is written to the objects in the process fragment as attributes.

Using the attribute values of the fragment events, the individual process fragments are then merged into process instances.

Process fragments are only imported into the PPM database if they can be assigned to particular process instances.

6.1.2. Merge process fragments

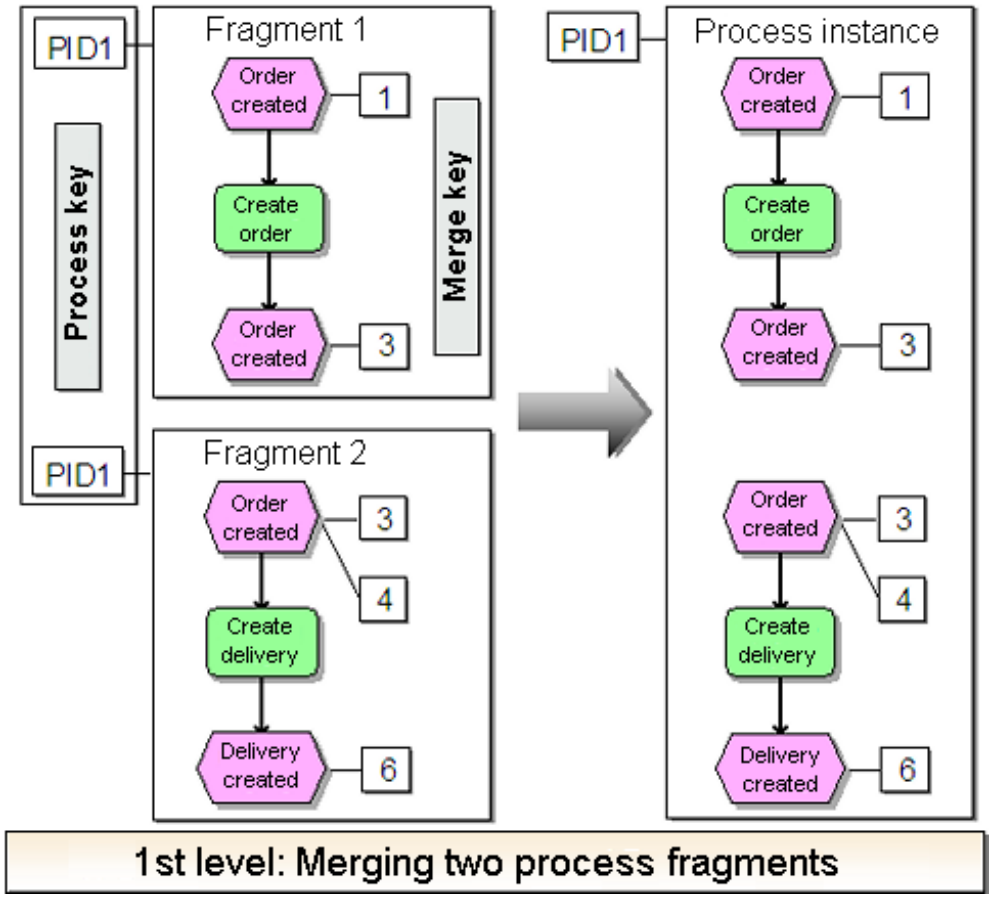
This chapter describes the merging of the imported process fragments into process instances. Merged process fragments correspond to business processes, which have actually been run through. They are known as process instances and, in the same way as the fragments, are represented in the EPC notation familiar from ARIS.

When merging, attributes are copied to the process instances and any processor data specific to a particular person is made anonymous.

The merging process takes place in two stages:

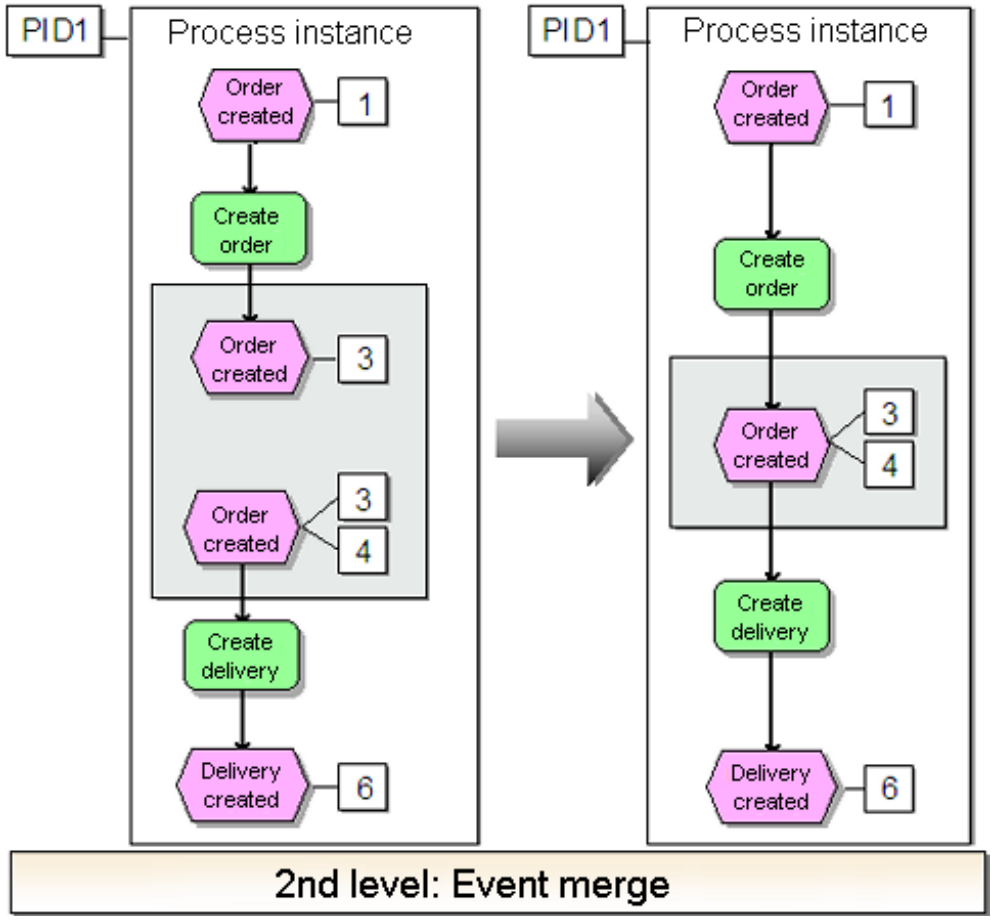
In the first step, the process fragments belonging to the same process instance, are identified and copied into a process instance using process keys.

Example 69.



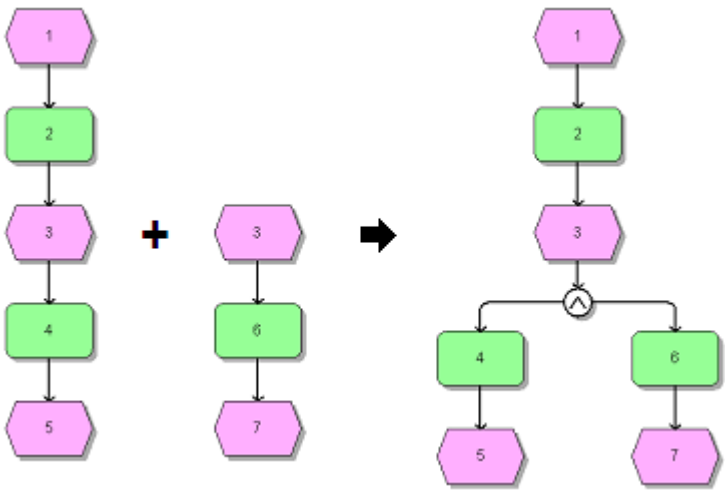
In the second step, the unconnected process fragments are linked together by merging the merge events. Merge events are events for which merge keys have been calculated.

Example 70.



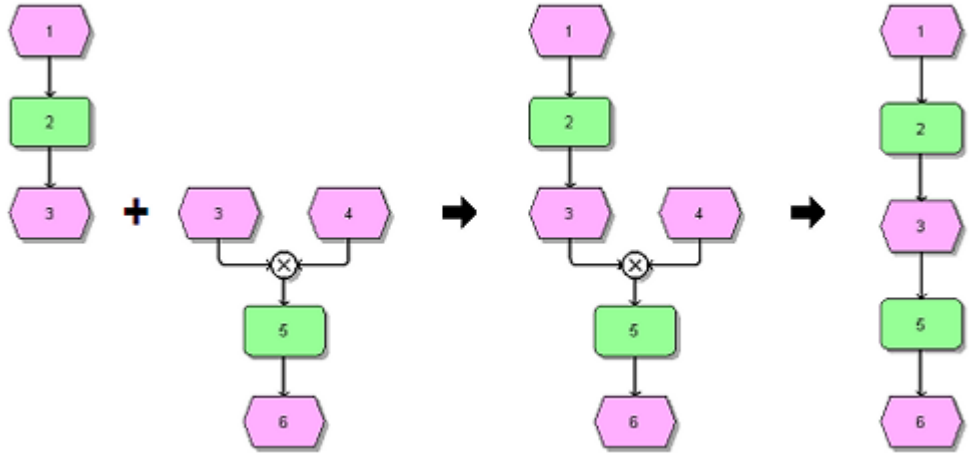
Any branches arising from the merge process are extended using rules and any unnecessary rules are deleted.

Example 71.



In the example below, a joining XOR rule is deleted during the merge, as process instances represent actual business transactions and they cannot contain any XOR rules. The XOR rule and the event that has no preceding function are deleted (event 4).

Example 72.



As it is not possible to distinguish between the individual source systems when merging the imported data, PPM enables processes to be viewed across systems and across the company.

The way in which process and merge keys are calculated is set individually using rules in the XML configuration for each client.

6.1.2.1. Copying the process instance attributes

When merging process fragments into process instances, attributes of objects are copied to the process instances. Process instance attributes form the basis for the calculation of individual measures depending on the dimensions.

When importing complete process instances in graph format, there is no merge process. These process instances already have process instance attributes.

6.1.2.2. Making organizational units anonymous

At an instance level, the actual processors for a function are known. As the users often cannot be shown on data protection grounds and are unimportant for the calculation of measures, it is possible to make the processors anonymous. To do this, all employees must be assigned to an organizational unit in PPM. When the data is imported, the staff are then replaced by the associated organizational unit in the course of merging the process. The information about the actual user is irretrievably lost.

6.1.3. Typify processes

After merging the imported process fragments into process instances, the process instances need to be classified to enable meaningful measure analyses. To do this, they are arranged in a freely definable two-level hierarchy: Process instances are assigned to process types, which are in turn summarized into process type groups. A process type group can therefore contain several process types, which in turn can contain several process instances. However, a process instance can only be assigned to a single process type and thus also to only one process type group.

The rules for identification are freely definable for each specific source system. The typification information is saved in process instance attributes.

The assignment of process types and process type group is represented in the process tree in PPM. The configuration of the process tree also specifies which measures and dimensions are available for the individual process types.

Process instances are actual business events that have occurred and are made up of imported process instance fragments. Similar process instances are summarized in process types, which in turn are assigned to a process type group. The term "process" as known from ARIS modeling is not specified as part of PPM and should therefore not be used.

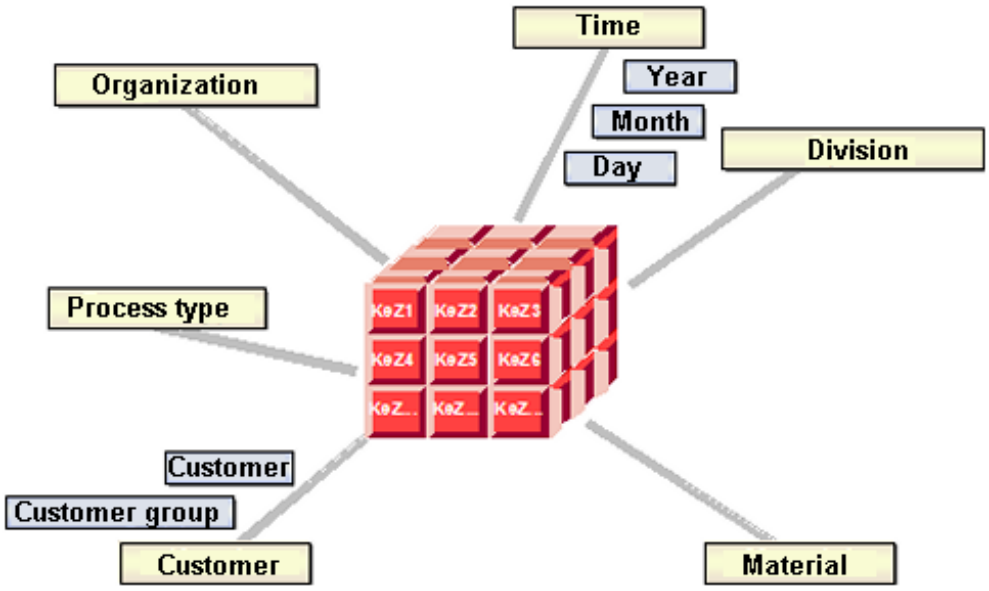
6.1.4. Calculate measures

A process instance is analyzed based on calculated measures. Measures are the properties of a process or a function calculated from measured variables. A distinction is made between function and process measures. Alongside predefined standard measures, including the **Number of processes** and **Process cycle time** indicators, any number of measures can be defined in the configuration using the associated calculation rule.

Dimensions are criteria according to which the measures of process instances and functions can be differentiated, for example the process type or the location.

For the process types contained in the process tree, the specified measures are calculated depending on the dimensions. The result is permanently stored in the PPM database. The values are stored in so-called **data cubes** to ensure that time-efficient queries can be made.

Example 73.



6.1.5. Checking planned values

For process monitoring purposes, the PPM system allows you to define planned and alarm values. Planned values relate to a set of process instances, alarm values relate to an individual process instance. Critical upward or downward alarm value infringements are possible for individual process instances, although the planned values for the associated set of process instances are still being met.

Process type-specific checking for upward or downward infringement of target values for individual measures concludes the PPM system's data import. The calculated measures are compared with the

planned values defined in the PPM user interface and the specified actions, such as sending an e-mail to the process manager (planned value message), are executed.

6.2. Import scenarios

The **IMPORT_SCENARIO** parameter is available from PPM version 10.2 onwards. The parameter specifies an import scenario that is used by the import programs **runxmlimport**, **runppmimport**, **runpikidata**, and **runppmcompress**.

The parameter is available in the **_vmparam.bat** memory parameter configuration file and in the **EpkImport_settings.properties** import parameter configuration file. Both files must always contain the same **IMPORT_PARAMETER** value.

- **_vmparam.bat** is located in **<PPM installation>\server\bin\work\data_ppm\config\<client name>\bin**
- **EpkImport_settings.properties** is located in **<PPM installation>\server\bin\work\data_ppm\config\<client name>**

In PPM CTK client setup, you can select a predefined import scenario that is set in both files. The scenarios **Small**, **Medium**, and **Large** are available. If you do not select a scenario, the **Small** scenario is set as default. If the entry is missing in the settings, or if a different value is specified than the allowed values, the **Small** scenario is set as default.

Each scenario contains a specific set of parameter values. The parameter values are optimized for the system environment.

- **Small** - Optimized configuration for server systems with at least 1 GB available main memory for import programs.
- **Medium** - Optimized configuration for server systems with at least 4 GB available main memory for import programs.
- **Large** - Optimized configuration for server systems with at least 8 GB available main memory for import programs.

If the scenarios do not meet your requirements, you can change the values within an existing import scenario. For example, if more than 8 GB main memory for import programs are required.

Example 74.

IMPORT_SCENARIO=LARGE is set. To increase the EPC read rate within this scenario to 1.500.000, set LARGE_READ_RATE_EPC=1500000 in EPKImport_settings.properties. Set also LARGE_LOCAL_PPM5_VM_MAX_MEM_RT_PPMIMPORT in _vmparam.bat to a higher value (for example, LARGE_LOCAL_PPM5_VM_MAX_MEM_RT_PPMIMPORT=10756) because now the PPM import requires more main memory.

6.2.1. Parameter values

The **IMPORT_SCENARIO** parameter is available in the **_vmparam.bat** file and the **EpkImport_settings.properties** file. The value for IMPORT_SCENARIO must be the same in both files (ensured by PPM CTK).

The following parameter values in the **_vmparam.bat** file are assigned depending on the import scenario selected. The parameters are written with the scenario plus an underscore.

- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MAX_MEM_RT_XMLIMPORT
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MAX_MEM_RT_PPMIMPORT
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MAX_MEM_RT_PIKIDATA
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MAX_MEM_RT_PPMCOMPRESS
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MIN_MEM_RT_XMLIMPORT
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MIN_MEM_RT_PPMIMPORT
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MIN_MEM_RT_PIKIDATA
- <IMPORT_SCENARIO>_LOCAL_PPM5_VM_MIN_MEM_RT_PPMCOMPRESS

Example 75.

- SMALL_LOCAL_PPM5_VM_MAX_MEM_RT_XMLIMPORT=1024 is used for IMPORT_SCENARIO=SMALL
- MEDIUM_LOCAL_PPM5_VM_MAX_MEM_RT_XMLIMPORT=4096 is used for IMPORT_SCENARIO=MEDIUM
- LARGE_LOCAL_PPM5_VM_MAX_MEM_RT_XMLIMPORT=8192 is used for IMPORT_SCENARIO=LARGE

Except for the exceptions listed below, all parameter values in the **EpkImport_settings.properties** file are assigned depending on the import scenario selected. The parameters are written with the scenario plus an underscore (for example, <IMPORT_SCENARIO>_READ_RATE_EPC).

Example 76.

- SMALL_READ_RATE_EPC=100000 is used for IMPORT_SCENARIO=SMALL
- MEDIUM_READ_RATE_EPC=500000 is used for IMPORT_SCENARIO=MEDIUM.
- LARGE_READ_RATE_EPC=1000000 is used for IMPORT_SCENARIO=LARGE

7. Support and legal information

This section provides you with some general information regarding product support and legal aspects.

7.1. 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 SAG ARIS GmbH, 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 ARIS sales organization. Please note that this type of manufacturer-specific or customer-specific customization is not covered by the standard ARIS software maintenance agreement and can be performed only on special request and agreement.

7.2. Support

Product support

We provide support for ARIS products to all customers with a valid support contract.

Information on product version availability and support, as well as sustained support dates, is available at ARIS Community.

Contact our Global Support services at ARIS Community to raise and update support incidents, or post ideas and feature requests.

Community

Register with ARIS Community to download products, updates and fixes, find expert information, and interact with other users.

Documentation

For information on how to use and activate products and for other technical support questions, visit the ARIS Documentation website.

If you want to give feedback on any documentation topic, write to documentation@aris.com.

7.3. Legal notices

This document applies to ARIS Process Performance Manager Version 10 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 © 2000-2026 SAG Aris GmbH, Saarbrücken, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name ARIS and all SAG Aris GmbH product names are either trademarks or registered trademarks of SAG Aris GmbH 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.

The Software may contain or be distributed with certain commercial third party components (“Third Party Components”), which may be subject to special license terms and conditions (“Third Party Terms”) located at https://aris.com/terms.

Third Party Terms include important licensing and warranty information and disclaimers of third party licensors. For the avoidance of doubt, if Customer uses the Software in accordance with the Documentation, the Third Party Terms do not restrict the rights granted under this Agreement. In the event that a third party manufacturer no longer provides active support for any Third Party Component, Supplier will use commercially reasonable efforts to respond to any issues with the Software related to the Third Party Components.

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with SAG Aris GmbH and/or its subsidiaries and/or its affiliates and/or their licensors.