@enterprise 8.0

Reporting

December 2017

Groiss Informatics GmbH

Groiss Informatics GmbH

Strutzmannstraße 10/4
9020 Klagenfurt
Austria

Tel: +43 463 504694 - 0
Fax: +43 463 504594 - 10
Email: support@groiss.com

Document Version 8.0.22989

Contents

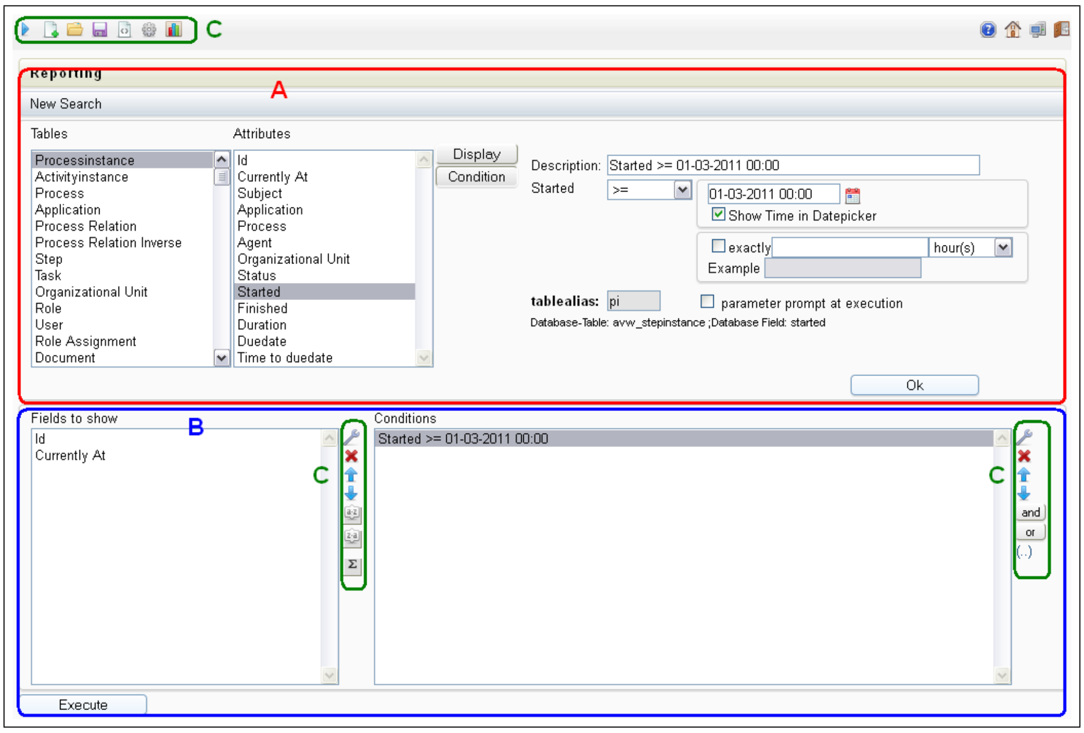
1 Introduction

With the Reporting component (Extended Search) you can create complex statistics and analyses on runtime data using a simple to use search mask. Queries created with this interface can be stored, executed, and edited later. The results can be shown as table, graphically, or exported to MS Excel, XML, PDF or CSV.
The Reporting is integrated into the permission system, which allows you to grant execute permissions for reports to specific user or roles.
You find the reporting component in the administration interface in the folder Search: The link "Extended Search" allows you to create reports, the link "Stored Queries" shows the list of reports.
Note that you need the right "Statistics" for creating reports.

2 Description of the Search Mask

The search mask is the central component and starting point for creating a report. It contains three areas (Fig 2.1):

- Fields and condition to add to report (A)
- The designed report (B)
- Options (C)



Tables

Processinstance
Activityinstance
Process
Application
Process Relation
Process Relation Inverse
Step
Task
Organizational Unit
Role
User
Role Assignment
Document

Attributes

Id
Currently At
Subject
Application
Process
Agent
Organizational Unit
Status
Started
Finished
Duration
Duedate
Time to duedate

Description: Started >= 01-03-2011 00:00
Started >= 01-03-2011 00:00
☑ Show Time in Datepicker
☐ exactly hour(s)
Example
tablealias: pi ☐ parameter prompt at execution
Database-Table: avw_stepinstance ;Database Field: started

Fields to show

Id
Currently At

Conditions

Started >= 01-03-2011 00:00

Figure 2.1: **Search Mask**

2.1 *Types of display fields*

The display fields are the fields, which appear as columns in the query result. Add a field to the result table by selecting the field name of table *Attributes* and clicking the "Ok" button. For express adding double-click on the field name. In the following we describe the display fields:

2.1.1 Display Fields of a Processinstance

- **Id**: Process-Id containing a link to the process history.
- **Currently At**: Tasks and agents of the currently running steps of the process.
- **Subject**: Subject of the process instance.
- **Application**: The application the process belongs to.
- **Process**: Name of the process.
- **Agent**: Agent who started the process.
- **Organizational Unit**: Organizational Unit where the process has been started.
- **Status**: Status of the process, possible values are:
  - **Started**: The process has been started and is running.
  - **Finished**: The process has been finished.
  - **aborted**: The process has been aborted

  See Fig. 2.2 for the transitions between these states.
- **Started**: Start time of the process.
- **Finished**: End time of the process.
- **Duration**: Duration of the process (end - start).
- **Duedate**: Due Date of the process.
- **Time to duedate**: Time interval from now (execution of the query) to the due date of the process.

2.1.2 Display Fields of an Activityinstance

- **Id**: Activityinstance-Id.
- **Child of**: Activityinstance has a parent (the processinstance of this task) and is child of it.
- **Agent**: Agent of the activityinstance.
- **Stepagent**: Agent, where the task is in his worklist.

- **Organizational Unit**: Organizational Unit of this task.
- **Taken**: Time when the task has been taken from role-worklist.
- **Status**: status of task, possible values are:
  - **idle**: Two reasons exist for this state:
    1. The selection of the agent has been interrupted and the task has no agent assigned.
    2. The selection of the path in a choice control structure has been interrupted.
  - **suspended**: Task is in suspension list.
  - **finished**: The task has been finished.
  - **aborted**: Task has been aborted.
  - **active**: Task is in worklist (agent is a user).
  - **waiting**: See state *idle*.
  - **compensated**: The task has been finished and compensated.

  See Fig. 2.2 for the transitions between these states.
- **Started**: Start time of task.
- **Finished**: End time of the task.
- **Duration**: Duration of the task (end - start).
- **Duration without suspension**: Duration of the task without the time in the suspension list.
- **Idletime**: Time span from start to taken.
- **Worktime**: Time span from taken to finished.
- **Worktime without suspension**: Worktime without the time in the suspension list.
- **Suspensiontime**: Time span in the suspension list.
- **Due Date**: Due date of the task.
- **Time to Due Date**: Time interval from now (execution of the query) to the due date of the task.

2.1.3 Display Fields of a Process

- **Id**: Process-Id containing a link to the process history.
- **Name**: Name of the process.
- **Version**: Version of the process.
- **Application**: Application, where the process is present (e.g. default).

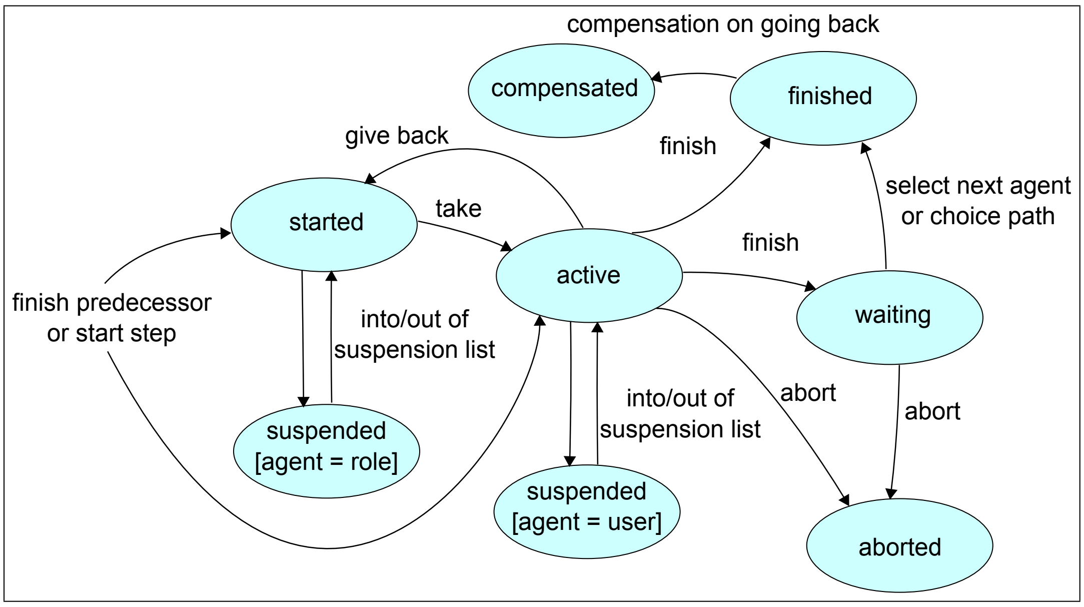

compensation on going back
compensated
finished
give back
finish
started
take
select next agent or choice path
finish
active
finish predecessor or start step
into/out of suspension list
waiting
into/out of suspension list
abort
abort
suspended [agent = role]
suspended [agent = user]
aborted

Figure 2.2: **Activityinstance states**

- **Subject**: Subject of the process.
- **Priority**: Priority of the process.
- **Description**: Process description.
- **Active**: Activity-status of the process (active, inactive).

2.1.4 Display Fields of an Application

- **Id**: Application-Id.
- **Name**: Name of the application.
- **Description**: Description of the application.
- **Organization Hierarchy**: Hierarchy, where the application is present.
- **Application class**: Application class of the server.
- **Client Application Class**: Application class of the client.
- **Application Directory**: Directory, where the application is present.

2.1.5 Display Fields of Process Relation

- **Process 2**: Process-Id of the process which is in relation to the searched process.
- **Process Relation Type**: Type of the process relation.

2.1.6 Display Fields of Process Relation Inverse

- **Process 1**: Process-Id of the process which is in relation to the searched process, but the source not the target of the relation is shown.
- **Process Relation Type**: Type of the process relation.

2.1.7 Display fields of Steps

As step we denote a set of executions of the same task back-to-back, i.e. without another task in between. The following display fields are available:

- **Activity Name**: Name of the activity.
- **Activity**: Activity either in tasks or in processes.
- **Started**: Start time of the step.
- **Finished**: End time of the step.
- **Duration**: Duration, related to the step in the process.
- **Worktime**: Sum of work time of the tasks.
- **Reactiontime**: Time span from start to taken in the first task of the step.
- **Idletime**: Time span from start to taken.
- **Suspensiontime**: Time span in the suspension list.
- **Duration without suspension**: Duration without the time in the suspension list.
- **Worktime without suspension**: Work Time without the time in the suspension list.

Restrictions for step fields:

- They can not be used in conditions.
- you get correct results for step-start, step-finish, etc. only if all tasks belonging to a step are in the result set.

Aggregations of Started

minimum: first step of a task
maximum: last step of a task
Count: counts the steps
Aggregation in expression: The aggregation can be set in the field *Expression*.

Date Format of Started Hour, Day, Week, Month, quarter, Year: Format for Timestamps.

Example:
Proz-ID =1, execution order: order $\rightarrow$ order $\rightarrow$ a_task $\rightarrow$ order

ID	Task	Task:Started	Task:Duration
1	order	21-04-2007 10:00	70 min
1	order	21-04-2007 11:10	50 min
1	a_task	21-04-2007 12:00	60 min
1	order	21-04-2007 13:00	80 min

ID	Task	Step:Start	Step:Duration
1	order	21-04-2007 10:00	120 min
1	a_task	21-04-2007 12:00	60 min
1	order	21-04-2007 13:00	80 min

ID	Task	Count(Step:Start)
1	order	2
1	a_task	1

ID	Task	MIN(Step:Start)	MAX(Step:Start)
1	a_task	21-04-2007 12:00	21-04-2007 12:00
1	order	21-04-2007 10:00	21-04-2007 13:00

2.1.8 Details to time intervals

Time intervals are computed as follows:

- **Processinstance**: *Duration* = *Started* to *Finished*
- **Activityinstance**: *Duration* = *Started* to *Finished*
- **Activityinstance**: *Idletime* = *Started* to *Taken*
- **Activityinstance**: *Worktime* = *Taken* to *Finished*
- **Activityinstance**: *Time to Due Date* = Execution time to *Due Date*

Fig. 2.3 shows the time intervals on a timeline. If the end time of an interval is not yet known, the execution time is taken instead. If you want only intervals where the end time is known add a condition that assures that.
Fig. 2.4 shows the computation of time intervals per step.
The process structure contains three steps, each step referring to a task. The process instance contains four instance steps, the first is an instance of the first process step, the next two are instances of the second step (this can be the result of a "change agent" action), the fourth is an instance of the third step. The right column shows how the work time per step is computed.

2.1.9 Display fields of Tasks

- **Id**: Task-Id.
- **Name**: Name of the task.
- **Version**: Version of the task.

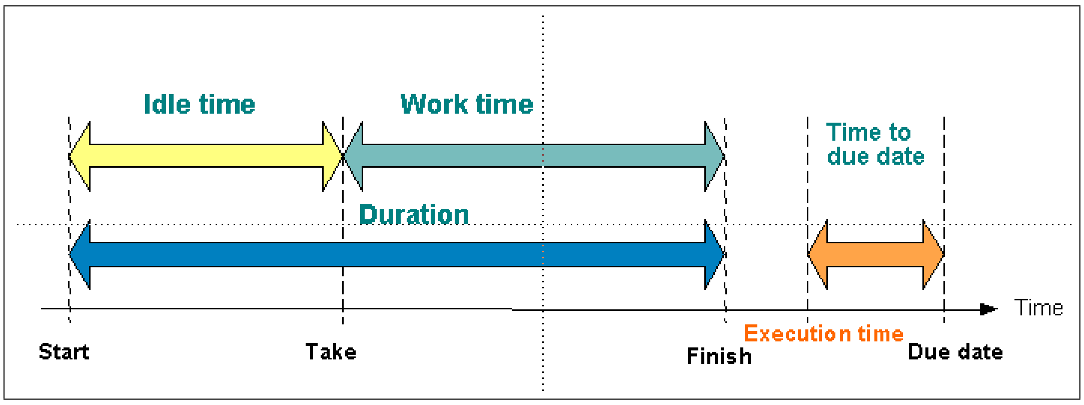
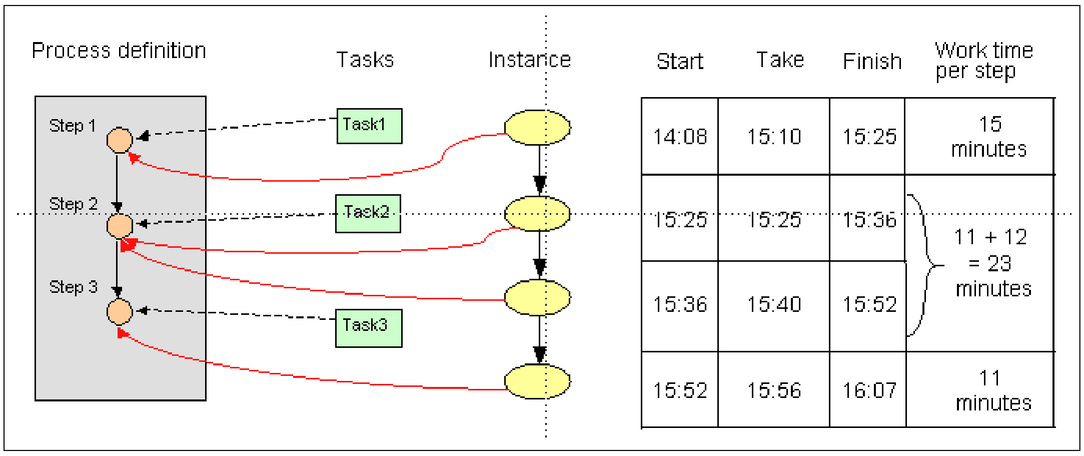


Figure 2.3: **Time intervals for tasks**



Start	Take	Finish	Work time per step
14:08	15:10	15:25	15 minutes
15:25	15:25	15:36	11 + 12 = 23 minutes
15:36	15:40	15:52	
15:52	15:56	16:07	11 minutes

Figure 2.4: **Computation of time intervals per step**

- **Description**: Free Text, which is visible in the worklist via the link to the task details.
- **Method Call**: Java-Method, which is called before the task is put into the worklist of a user.
- **Postcondition**: Java-Method, which is checked at run-time when a user finishes the task (sends it to the next agent).
- **Postcondition-Message**: Free text, which will be shown when the postcondition evaluates to false.
- **Compensation Method**:Java-Method, which will be executed when the activity is passed when going back to an earlier step in the process.
- **Take Hook**: Java-Method, which is called when the task is taken from the role-worklist to the personal worklist.
- **Untake Hook**: Java-Method, which is called when the task is given back to the role worklist.

- **Active**: Activity-status of the task (active, inactive).

2.1.10 Display fields of Organizational Units

- **Id**: OU-Id.
- **Name**: Name of the OU.
- **Description**: Description of the OU.
- **E-Mail**: E-Mail address of the OU.
- **Address**: Address of the OU.
- **Tel.-Nr.**: Phone number of the OU.
- **Active**: Activity-status of the OU (active, inactive).
- **Type**: Type of the OU (External OU, Dependent).
- **Organization Class**: The organization class the OU belongs to.
- **Follow-OU**: Follow-OU, which replaces the current OU.

2.1.11 Display fields of Roles

- **Id**: Role-Id.
- **Name**: Name of the role.
- **Description**: Description of the role.
- **Type**: Type of the role (local, global, hierarchic).
- **Active**: Activity-status of the role (active, inactive).
- **Reference-Role**: Reference roles are used for defining different roles with different rights but one "reference" role used in process definitions.
- **Application**: Application, where the role is known.

2.1.12 Display fields of Users

- **Id**: User-Id.
- **Surname**: Surname of the user.
- **First name**: First name of the user.
- **Description**: Description of the user.
- **E-Mail**: E-Mail address of the user.
- **Tel.-Nr.** Phone number of the user.

- **Language**: Language for the user interface.
- **Active**: Activity-status of the user (active, inactive).
- **Server**: @enterprise-Server, where the worklist is accessible.
- **Date of the last password change**: Date, when the password was changed.
- **Has to change password at next login**: User has to change password at next login.
- **Password never expires**: Password of the user never expires.
- **Cannot change password**: User has no right to change the password.

2.1.13 Display fields of Role Assignments

- **Organizational Unit**: OU, where the role is assigned.
- **User**: User, who has the role.
- **Role**: Id of the assigned role.

2.1.14 Display fields of Notes

- **Subject**: Subject of the note (free text).
- **Content**: Content of the note (free text).

2.1.15 Display fields of Documents

- **Name**: Name of the document.
- **Text**: Content-text of the document.
- **Metatext**: Metatext of the document.
- **Created**: Creation date of the document.
- **Changed**: Change date of the document.
- **Organizational Unit**: Organizational Unit, where the document is assigned.
- **Creator**: Creator of the document.

2.1.16 Display fields of Documentfolder

- **Folder**: Link to the folder.

2.1.17 Display fields of document versions

- **Version**: Link to version of document.
- **Created by**: User who created this version.
- **Created at**: Timestamp of creation of this version.
- **Description**: Description (free text) written by the creator.

2.1.18 Form fields as display fields

For adding form fields click on a form in the list *Tables*, select a formfield of the table *Attributes* and activate the button "Ok". If you select a version independent form (no number after the form name), you must create a view over the form versions, see the Administration Guide, section Forms, for details. If the form is not a process form but a subform you must select the main form (process form).

2.1.19 User defined display fields

When selecting the display field "User defined" you will see a mask with the following fields:

- **Columntitle**: text for the column header.
- **Schema**: schema of a database.
- **Tablealias**: a table alias for this query.
- **Data Type**
- **Database-Table**: name of database table.
- **Database Field**: name of field of table.
- **Condition**: SQL-expression, used as selection to the table.
- **Aggregation**: maximum, minimum, count, average, sum, Aggregation in expression
- **Date Format**: hour, Day, Week, Month, quarter, Year: grouped by time spans

Example 1: Surname of agent of the process where userid starts with user

Database-Table: avw_user
Tablealias: u
Database Field: surname
Columntitle: Surname
Data Type: String
Condition: u.id like 'user

Example 2: Create a list of tasks with id and oid:
Attribute 1:
Database-Table: avw_task
Tablealias: t
Database Field: oid
Columntitle: oid
Data Type: String
Condition:

Attribute 2:
Database-Table:

Tablealias: t
Database Field: id
Columntitle: id
Data Type: String
Condition:

2.1.20 Aggregations

The following aggregations are possible:

- count (e.g. count(id))
- maximum (e.g. max(duration))
- minimum
- average
- sum
- Aggregation in expression: Select this option if select statement includes a sql aggregation.

Count can be used for all fields, the other aggregations can only be used for date or numeric fields (exception: *Aggregation in expression*).

2.1.21 Grouping by setting the date format

If you select a display field of type date (for example Process:Start), you can select the following date formats: hour, day, week, month, quarter, and year. With this feature you can group by time intervals. For example you can retrieve the number of processes started per quarter by selecting the following display fields (see also the example section and Fig. 3.1 and 3.2):

- Count(Process:Name)
- Process:Started [quarter]

2.1.22 Options for fields

Clicking on an entity and an attribute of this entity while button *display* is activated, the attribute option page is displayed. Double-clicking on an already added column shows you the page too, but in edit mode. There are 2 modes for editing the options. In normal mode you can specify the following options:

- Columntitle: The column header, can include I18N keys (e.g: @@@key@@)
- Aggregation: Select an aggregation as described in 2.1.20

To switch into the extended mode, click on the link *extended*. There you can specify the following:

- Columntitle includes HTML: Enable this option if the Displayname is a HTML Code fragment to parse. Note that only one root Element is possible:e.g. `<img src='test.gif'/>` is not possible but:

  `<div><img src='test.gif'/>Test</div>` is correct.

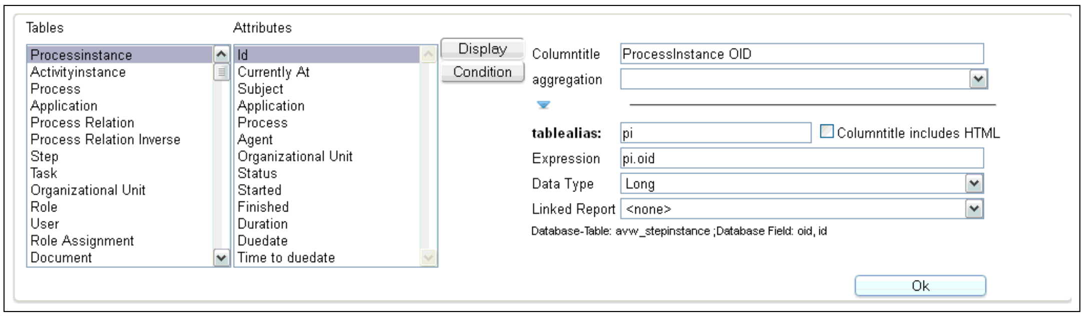
- Tablealias: Is used as table alias in SQL Query. The standard alias, defined in schema, is prefilled, but can be changed. Each entity/alias pair has to be joined to the report (see 2.1.23. Once the attribute is added to the report, the alias can not be modified in edit mode to ensure that selected joins are correct.

- Expression: Can be used to overwrite default select of this attribute, which is defined in schema. Ensure that Expression includes a complete Sql select statement (e.g: *pi.oid*). If aggregation is set to *Aggregation in expression*, a sql aggregation is suspected in expression (e.g: *count(pi.status)*)

- Data Type: By specifying this option, default type of the result data is overwritten. Ensure that the selected data fits to the selected data type. If set to *TimeInterval* two comma-separated date fields are expected in expression.

- Linked Report: If a column is linked to another report, the data value of of the clicked column will be used as parameter at execution. Therefore the linked report has to expect the parameter condition on index n+1 and n is the index of the last parameter at execution condition of the initial report.

Confirm the modifications by clicking on *OK* button.
Fig. 2.5 shows how this extended mode is used to select only the ProcessInstance OID, which is not an attribute in ProcessInstance entity. So the expression is set to *pi.oid* and data type is *Long*.

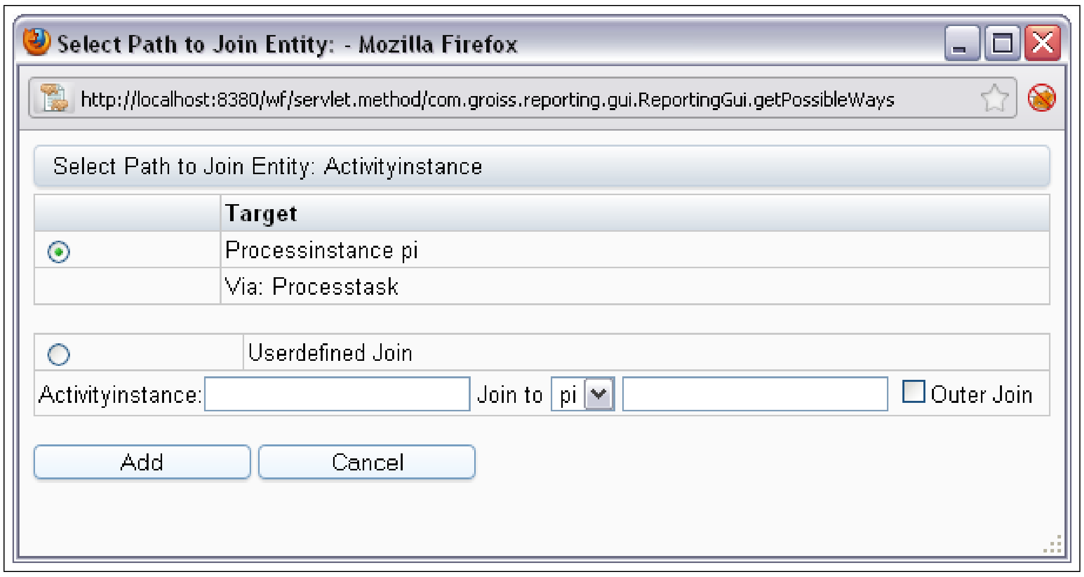


Columntitle: ProcessInstance OID
aggregation:
tablealias: pi
☐ Columntitle includes HTML
Expression: pi.oid
Data Type: Long
Linked Report: <none>
Database-Table: avw_stepinstance ;Database Field: oid, id

Figure 2.5: **Selecting OID of ProcessInstance via Extended Options**

2.1.23 Selecting Joinpaths

Reporting enables the user to define the way how the entities of the reports are joined. This increases the amount of possible reports while complexity of reporting designer increases too. Any new entity (except the first) has to be joined to the report. An entity is defined by its table alias. When adding a new column or condition of an entity, which wasn't added to the report already, engine will ask the user for the join to use. These possible joinways are predefined in schema. Fig. 2.6 shows the joinpath selection for entity activityinstance to a report, which has already columns of processinstance. To select joinpath, select the radio button and click on *Add*. The popup will close and the attribute/condition will be added to the report. Fig. 2.7 shows how join selection works if there are more than one entity already


Select Path to Join Entity: - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.getPossibleWays

Select Path to Join Entity: Activityinstance

	Target
●	Processinstance pi
	Via: Processtask

○ Userdefined Join

Activityinstance: [ ] Join to [pi] [ ] ☐ Outer Join

Figure 2.6: **Selecting Joinpath for Processinstance and Activityinstance**

added to the report and more than one predefined joinpaths exists to these entities. Each join target can be chosen with different paths, so in front of the target (e.g: *ProcessInstance*) a button (+) is located. Click on this button to see the possible paths, choose the path and click on *Add*.
If the predefined joins do not include the needed join, user-defined joins can be declared. Fig. 2.8 shows how 2 entities (processinstance pi, processinstance pi2) can be joined via id to get all subprocesses. If checkbox *outer* is activated, an outer join is made to ensure that even processes which don't have subprocess are displayed in result.

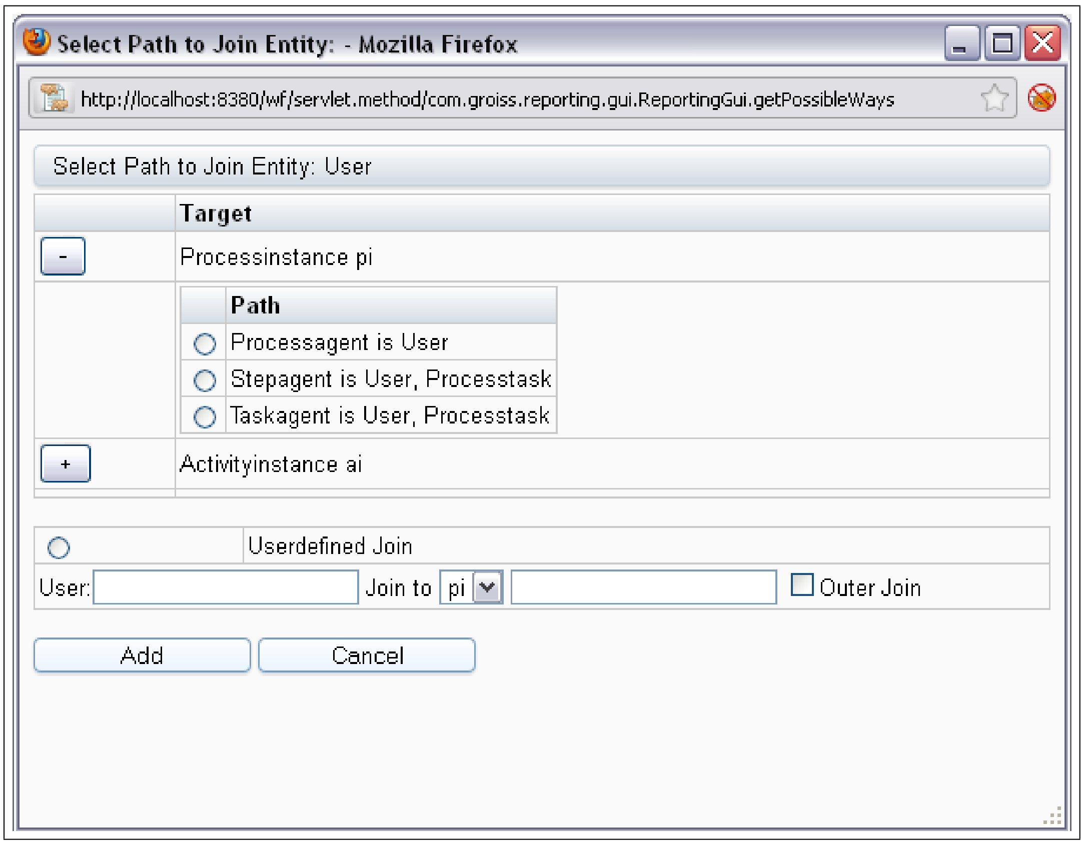
2.2 View Options

2.2.1 Order and sorting

The order of the display fields in the result table *Fields to show* can be changed by using the buttons "Up" and "Down".
You can sort the result by each of the display fields. Select a field and click either "ascending" or "descending" dependent on the sort order you want. If you mark more than one field for sorting, the first sort criteria is the first attribute marked for sorting in the list, and so on.

2.2.2 Groupingcolumn and -key

Grouping of columns enables reporting to do a second level aggregation, which groups all displayfields. Any aggregation-function of the specific column data type is useable as



Select Path to Join Entity: - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.getPossibleWays

Select Path to Join Entity: User

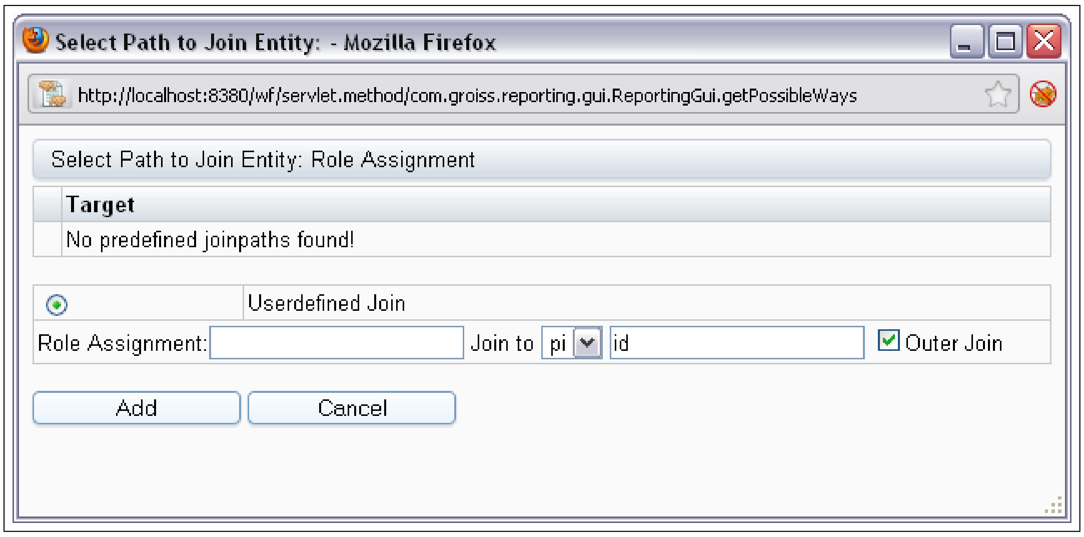
	Target
-	Processinstance pi
	○ Processagent is User ○ Stepagent is User, Processtask ○ Taskagent is User, Processtask
+	Activityinstance ai

○ Userdefined Join

User: Join to pi ☐ Outer Join

Add Cancel

Figure 2.7: **Joining User either to ProcessInstance or ActivityInstance**



Select Path to Join Entity: - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.getPossibleWays

Select Path to Join Entity: Role Assignment

Target
No predefined joinpaths found!

◉ Userdefined Join

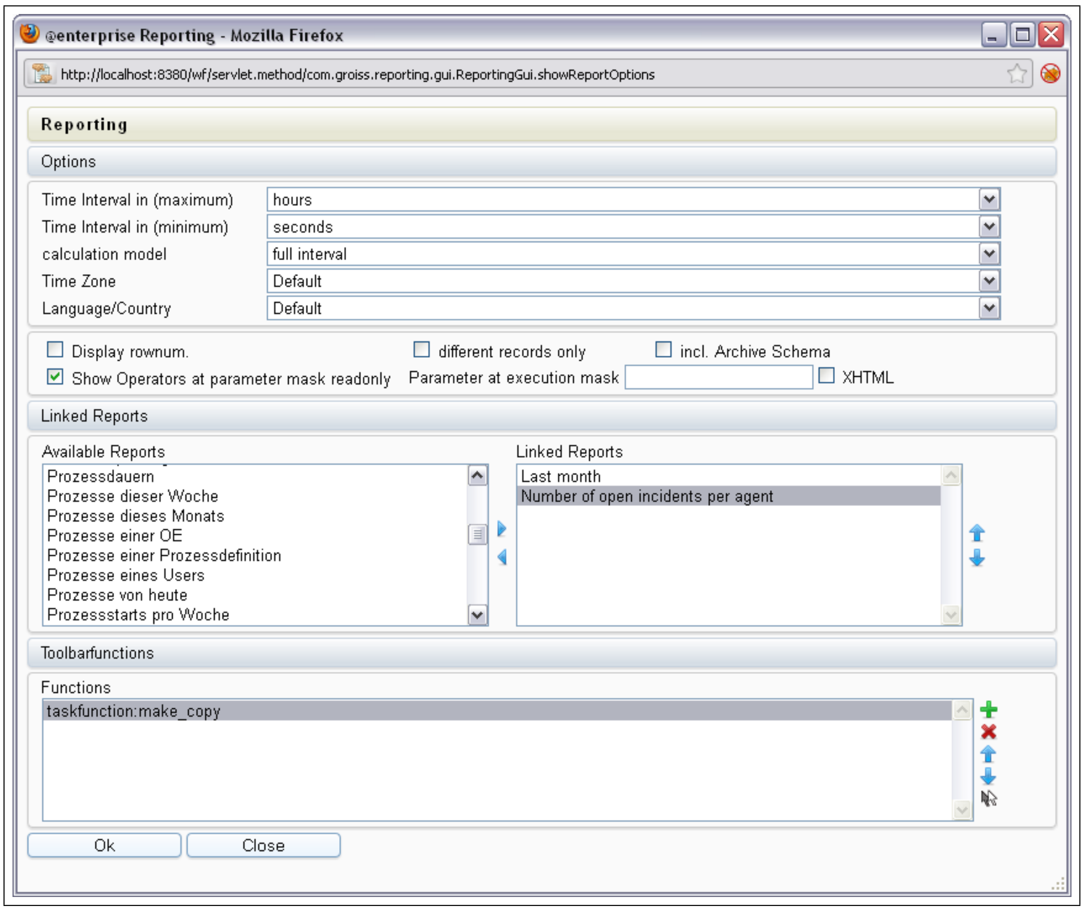
Role Assignment: Join to pi id ☑ Outer Join

Add Cancel

Figure 2.8: **Userdefined Joins**

grouping-function. After selecting the first grouping-function, grouping keys maybe specified. So a grouping line is displayed in the result if when one of the keys is changing. Here the sorting and order of the key columns is determining.

2.3 Reportoptions


@enterprise Reporting - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.showReportOptions

Reporting

Options

Time Interval in (maximum)	hours
Time Interval in (minimum)	seconds
calculation model	full interval
Time Zone	Default
Language/Country	Default

☐ Display rownum. ☐ different records only ☐ incl. Archive Schema

☑ Show Operators at parameter mask readonly Parameter at execution mask ☐ XHTML

Linked Reports

Available Reports:
Prozessdauern
Prozesse dieser Woche
Prozesse dieses Monats
Prozesse einer OE
Prozesse einer Prozessdefinition
Prozesse eines Users
Prozesse von heute
Prozessstarts pro Woche

Linked Reports:
Last month
Number of open incidents per agent

Toolbarfunctions

Functions:
taskfunction:make_copy

Ok Close

Figure 2.9: **Reportoptions**

2.3.1 Time interval

Can be shown in seconds, minutes, hours, days or weeks.

2.3.2 Computing of time intervals

For computing time intervals you can use two computing models: "full interval" and "no weekends", in the latter the weekends are removed from the time interval. In the reporting schema you can add additional computing models. See the API documentation for how to implement a time model.

2.3.3 Timezone and Locale

To execute a report in a specific timezone and/or Locale, you can specify it here. If *default* is chosen, the timezone and locale of the executing user is taken.

2.3.4 Checkbox *Display row number*

If this checkbox is activated, row numbers will be displayed in result.

2.3.5 Checkbox *different records only*

If this checkbox is activated, same records will not be shown a second time in the result.

Example: Search all processnames, which are available.
We assume that the processname *parfor* exists a second time. If this checkbox is activated and you start the search, you will see the name *parfor* one time only.

2.3.6 Parametermask

This parameter is used to overwrite the parameter at execution mask of the report.

2.3.7 Checkbox *Show Operators at parameter mask read-only*

Default values for operator and value of parameter at execution conditions can be chosen since @enterprise 8.0. The operator selection is shown read-only if this checkbox is checked.

2.3.8 Checkbox *incl. Archive Schema*

If this checkbox is activated, the archive schema is included in the search. If archive schema exists, there is a view which contains both, old and new stepinstances. By activating this checkbox, this view will be used instead of the avw_stepinstance table.

2.3.9 Linked Reports

Any stored query can be linked to a report. HTML-Links to execute the reports are displayed at the bottom of the resultpage.

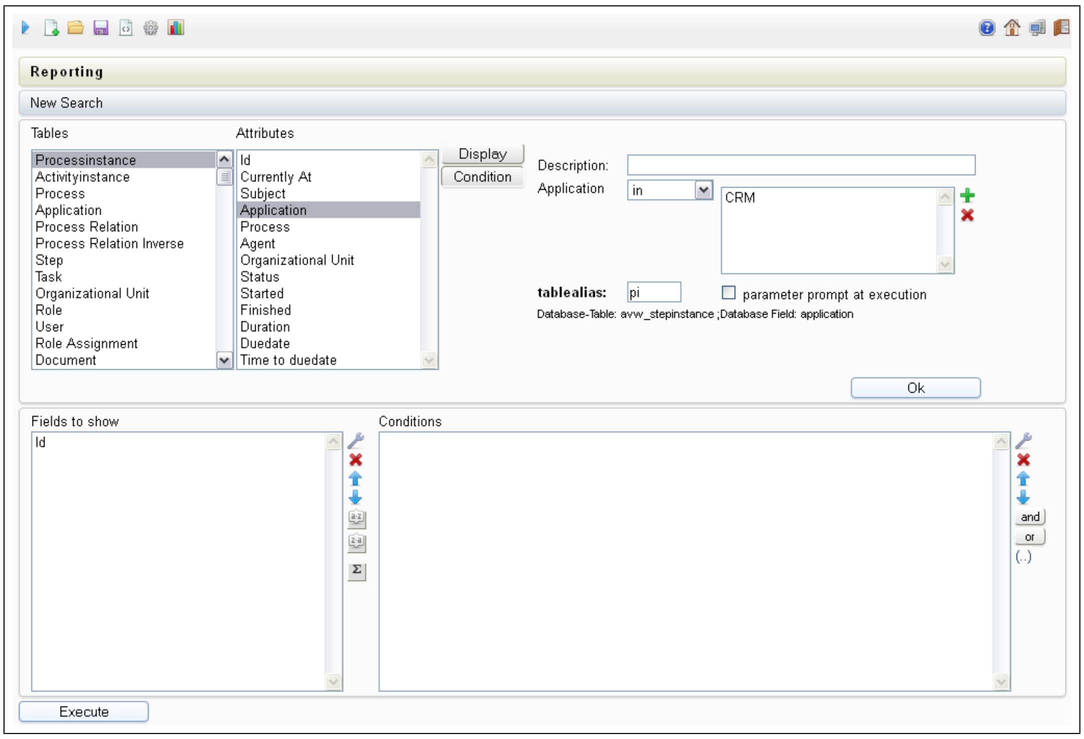
2.3.10 Toolbarfunctions

Like at the Gui configurations, toolbar functions can be configured. These functions can be declared as task functions in administration or as xml node in a config-file. One toolbar-function may be marked as doubleclick action, which only fires on html export and selection on lines. This action is executed when the user doubleclicks on the result line.

2.4 Query conditions

With conditions you can reduce the amount of lines in the result set. Click the "Condition" button beside the list *Attributes* to get the condition attributes. The attributes are categorized in:

Objects (Applications, Org.Units, Processes, Tasks, Agents) You can specify sets of objects where the searched value is included or not included, see Fig. 2.10. When selecting agents you can also select the user, who executes the query. Clicking on the *Add* Icon will open a select mask. Select the object and click on *OK* to take the value. To add more than one value, double-click on the values.



Description:

Application: in — CRM

tablealias: pi

parameter prompt at execution

Database-Table: avw_stepinstance ;Database Field: application

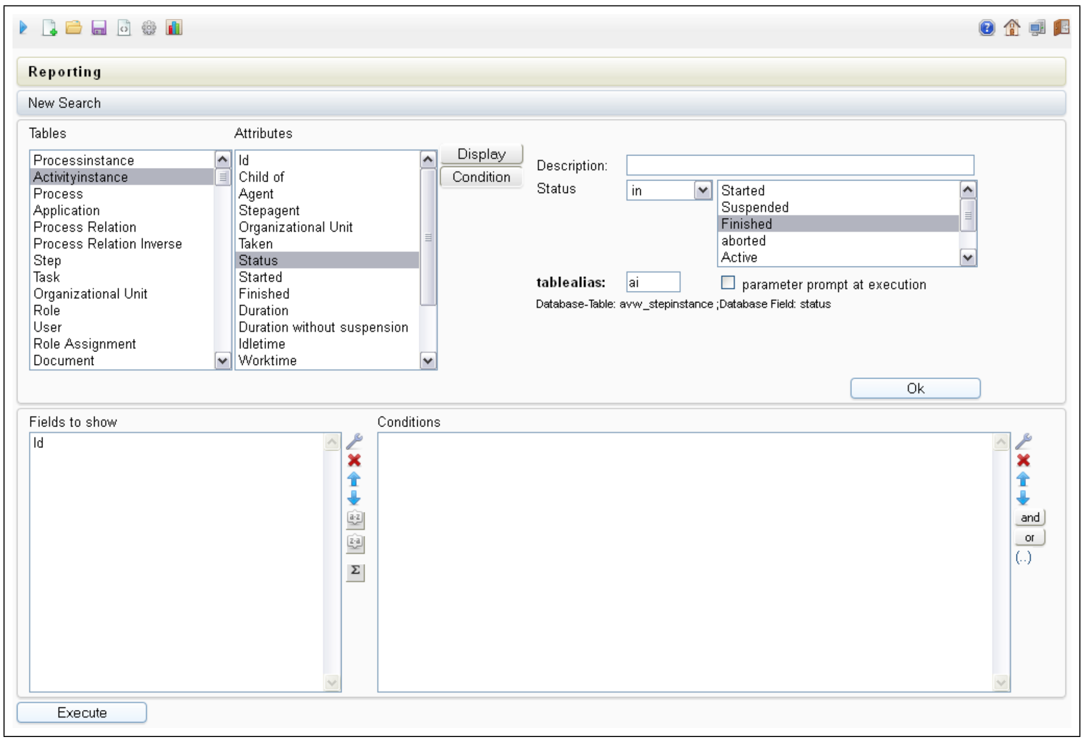
Figure 2.10: **Condition for objects**

Text fields (Id, Name, Subject) Specifies a string search. If operator is *like* or *not like* any string is found, which has the given text as substring (Infix-search). To do a prefix search, add a % at the end of the search pattern, for postfix search add a % at the beginning of the search pattern. Activate the checkbox *Ignore Case* to enable case insensitive search.

States (Processinstance state, Activityinstance state) Select a item from the lists *Status* (see Fig. 2.11).

Dates (Started, Finished, Duedate) You have several possibilities:

- Specify a date, where the attribute value is greater (after) or equal to this date,



Reporting

New Search

Tables

- Processinstance
- Activityinstance
- Process
- Application
- Process Relation
- Process Relation Inverse
- Step
- Task
- Organizational Unit
- Role
- User
- Role Assignment
- Document

Attributes

- Id
- Child of
- Agent
- Stepagent
- Organizational Unit
- Taken
- Status
- Started
- Finished
- Duration
- Duration without suspension
- Idletime
- Worktime

Display

Condition

Description:

Status: in

- Started
- Suspended
- Finished
- aborted
- Active

tablealias: ai

☐ parameter prompt at execution

Database-Table: avw_stepinstance ;Database Field: status

Ok

Fields to show

Id

Conditions

and

or

(..)

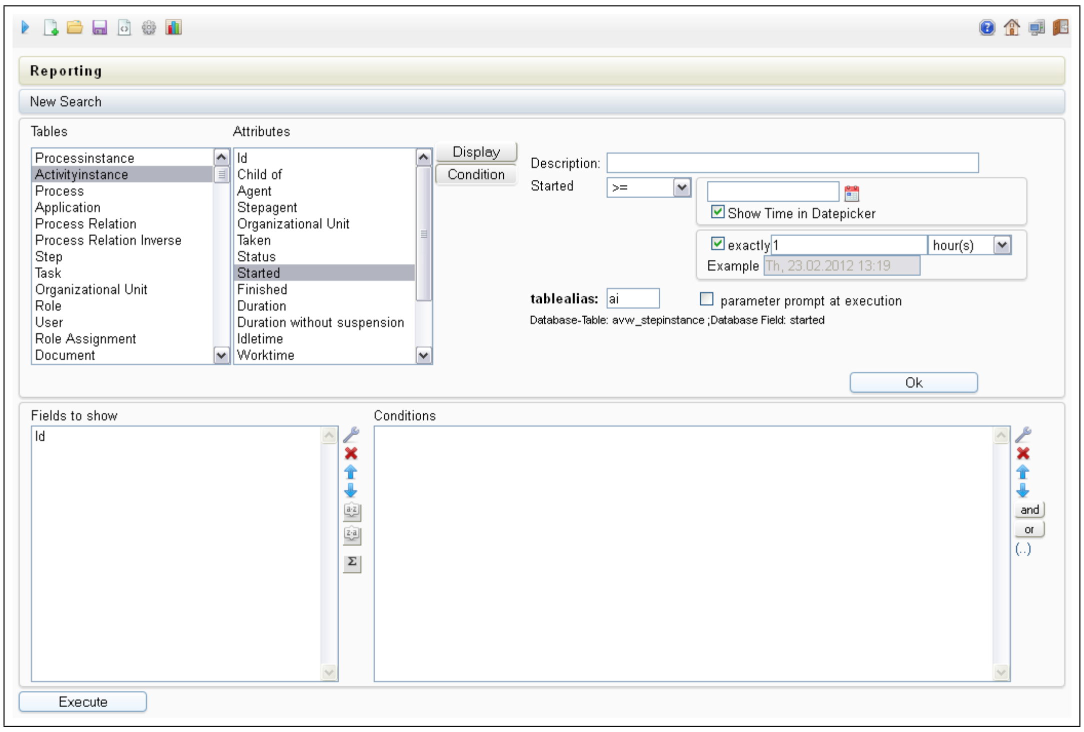
Execute

Figure 2.11: **Conditions for states**

- The attribute value is less (before) this date.
- You specify a time span in hours, days, weeks, month, or years. The checkbox "exactly" defines the end point of the time span: When checked, the time span ends at execution time, else it ends at the begin of the execution hour, day, month, etc. The attribute value must lay in the time span ending at this defined end point.
- The attribute value is less than the start point of the time span described above.
- The attribute value is null,
- The attribute value is not null.

Fig. 2.13 show some examples of date conditions.

Form fields Select a form of the list *Tables* and then a formfield of the list *Attributes* to set conditions in the condition mask.
If you activate the button *Show Form*, you will get a preview of the form, where you can enter values in the fields. After applying, the entered values will be set as condition in the list *Conditions*.
Otherwise you can specify a formfield condition like any other condition. see Fig. 2.15.



Reporting

New Search

Tables

- Processinstance
- Activityinstance
- Process
- Application
- Process Relation
- Process Relation Inverse
- Step
- Task
- Organizational Unit
- Role
- User
- Role Assignment
- Document

Attributes

- Id
- Child of
- Agent
- Stepagent
- Organizational Unit
- Taken
- Status
- Started
- Finished
- Duration
- Duration without suspension
- Idletime
- Worktime

Display | Condition

Description:

Started >=

☑ Show Time in Datepicker

☑ exactly 1 hour(s)

Example Th, 23.02.2012 13:19

tablealias: ai

☐ parameter prompt at execution

Database-Table: avw_stepinstance ;Database Field: started

Ok

Fields to show

Id

Conditions

and | or | (..)

Execute

Figure 2.12: **Conditions for date fields**

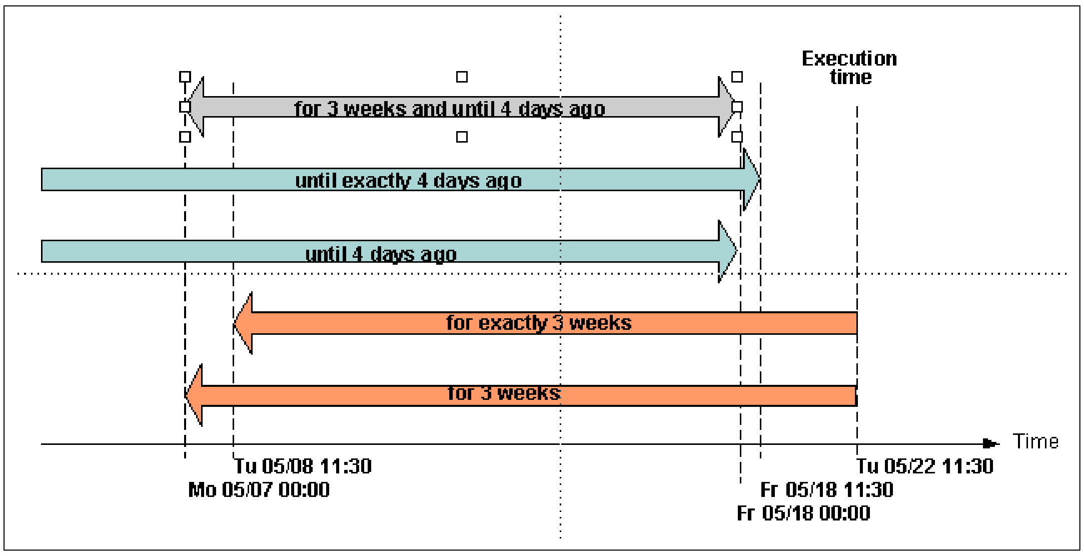
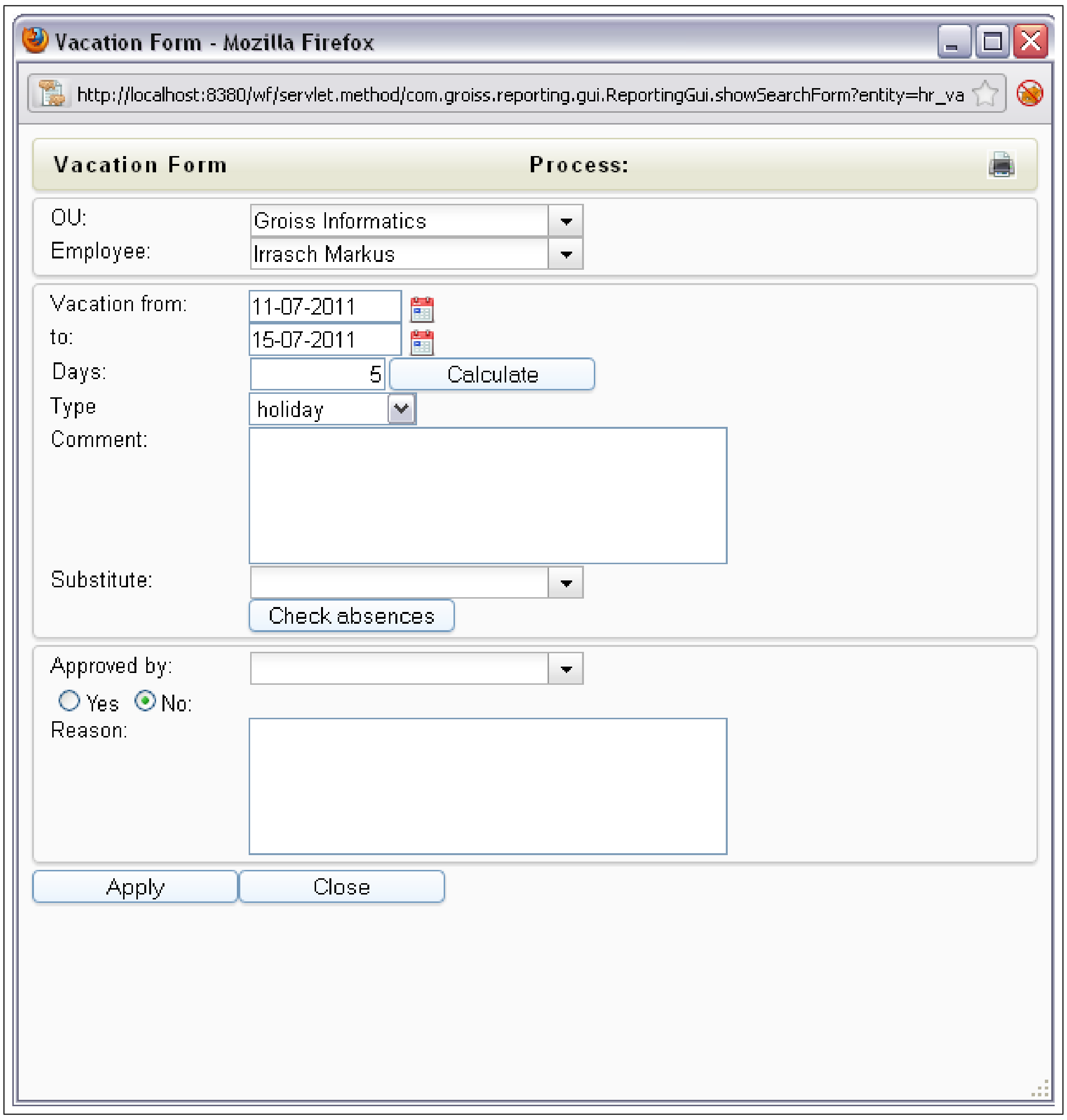


Figure 2.13: **Date condition examples**

2.5 *Connection of conditions*

All conditions you add can be connected with "AND" or "OR". Furthermore, you can set parentheses by selecting one or more conditions and click on the corresponding button *(..)*.


Vacation Form - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.showSearchForm?entity=hr_va

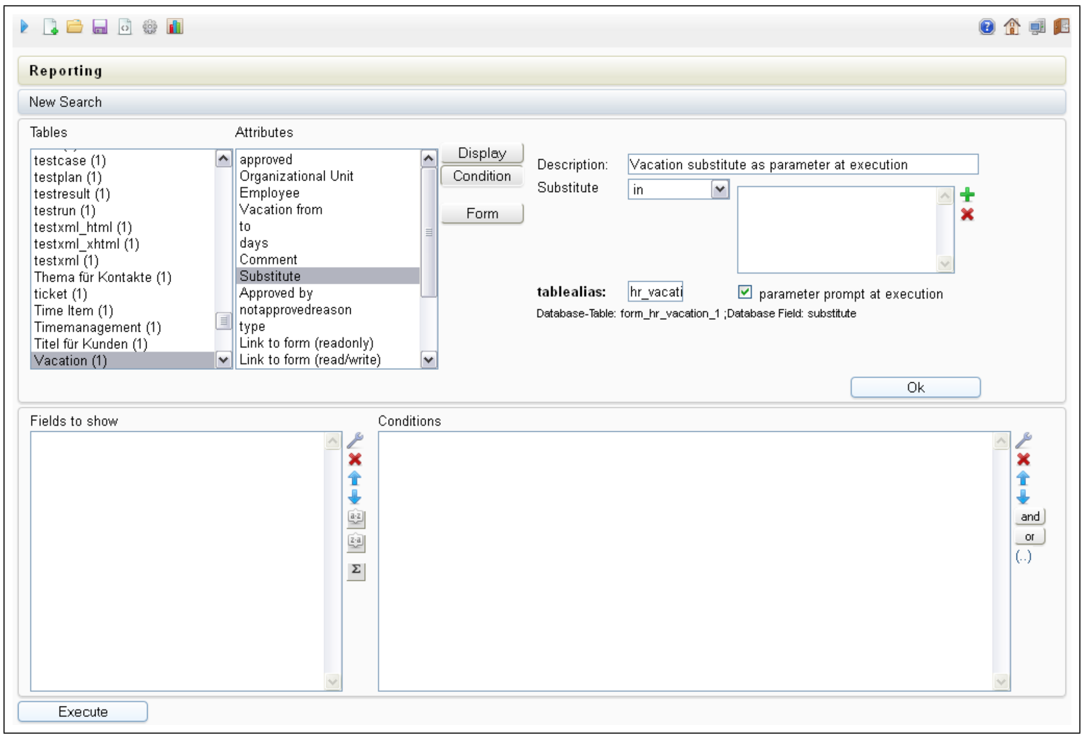
Vacation Form Process:

OU:	Groiss Informatics
Employee:	Irrasch Markus
Vacation from:	11-07-2011
to:	15-07-2011
Days:	5
Type	holiday
Comment:	
Substitute:	
Approved by:	
Yes [✓ No]:	
Reason:	

Figure 2.14: **Set condition by entering values in the formfields**

2.6 *Queries with parameters*

You can define queries where some parameters are set at execution time (explicit parameter). For example, a query returning all instances of a given process. Which process you want, is specified at execution time. When you execute the query, the system shows you a mask for filling in the concrete values for the parameters (see Fig. 2.16).
To enable explicit parameter condition, activate the *parameter prompt at execution* checkbox. Its even possible to assign default values to operator and value fields, which will be prefilled in the param mask.
It is also possible to define queries with implicit parameters. The difference to explicit parameters is that the implicit parameters are determined by the context and not over a mask. For example: Agent at execution returns only results where agent is the user, who



Description: Vacation substitute as parameter at execution
Substitute: in
tablealias: hr_vacati
[✓] parameter prompt at execution
Database-Table: form_hr_vacation_1 ;Database Field: substitute

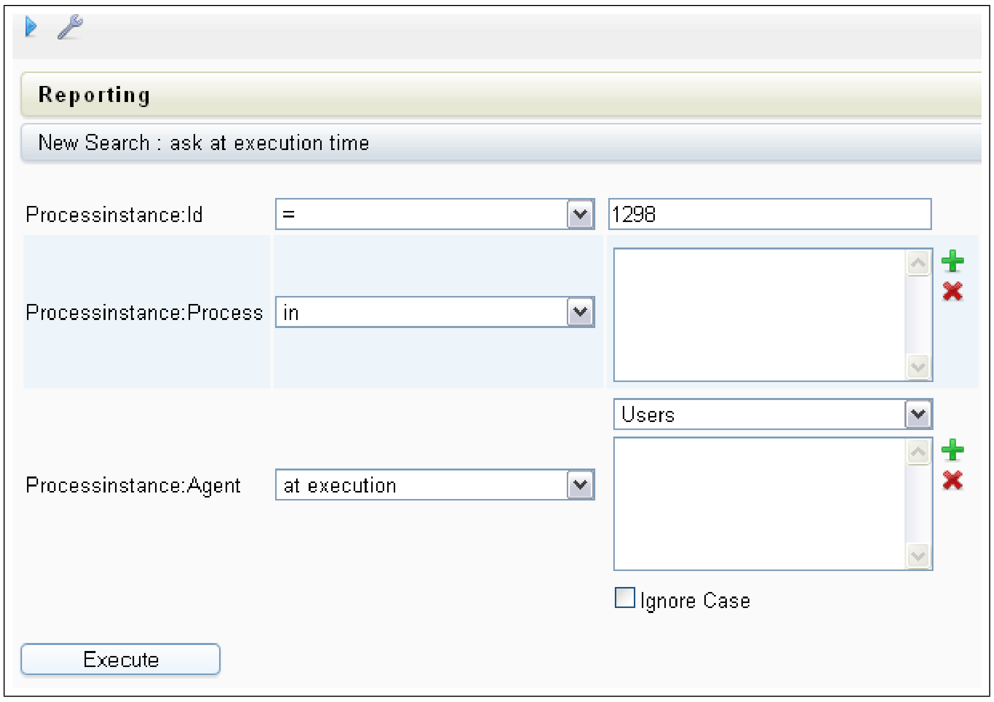
Figure 2.15: **Set formfield condition by entering values in condition form**

executes the report.
The following implicit parameter conditions are possible:

- relative time conditions (e.g. since 3 days)
- Agent at execution conditions

2.7 *User-defined conditions*

As last case in the section about conditions we present the possibility to define arbitrary conditions. You should know some SQL to define such conditions (see example 3.4). Depending on the selected display or condition fields you can refer the table aliases used in the report.


Reporting

New Search : ask at execution time

Processinstance:Id	=	1298
Processinstance:Process	in	
Processinstance:Agent	at execution	Users

☐ Ignore Case

Figure 2.16: **Setting the parameters**

3 Examples

This sections shows a few example reports. You need the right "statistics" to create and execute the following examples.

3.1 Example 1 (Aggregationen; Date fields)

Compute the minimum, average, and maximum work times per step for all finished tasks (in days), which has been started in the last month. Sort the results by process-id. Additionally there should be sum row which shows the number of processes and the average of the three time interval columns.
Select the following display fields:

- ProcessInstance:Id .... to have the process in the result table (ascending sorted).
- ActivityInstance:Worktime with aggregation *minimum*.
- ActivityInstance:Worktime with aggregation *average*.
- ActivityInstance:Worktime with aggregation *maximum*.

Then add the following conditions:

- ActivityInstance:Started >= exactly 1 month
- ProzessInstance:Status in (Finished)

Then set the option "Time Interval in" to "days". Fig. 3.1 shows the search mask, Fig.3.2 the result.

3.2 Example 2 (Grouping over time intervals; implicit and explicit parameters)

How often per week a task of a specified process is taken by the user executing the query? The process is given as parameter (specified at run-time). Sort the result by the week.
Select the following display fields:

- Task:Name
- ActivityInstance:Id with aggregation "count"
- ActivityInstance:Taken with date format "week" and sort order "ascending"

Then add the following conditions:

- ActivityInstance:Taken is not empty .... select only tasks which has been taken
- ProcessInstance:Id = PARAM ... process is specified when query is executed
- ActivityInstance:Agent = Agent at execution

Fig. 3.3 shows the search mask, Fig. 3.4 the result as diagram.

3.3 Example 3 (Null-Values: grouping, aggregation; form fields)

Which user has started how many processes, where in the form "Jobform" the subject field is empty and the recipient is "James".
Additionally show the field "description" of the "Jobform" form.

Select the following display fields:

- ProcessInstance:Agent ... start agent of a process,
- ProcessInstance:Id with aggregation "‘count"’ ... instead of count(Process:agent) to count processes without agents also,
- Jobform:description ... the form field description of the form Jobform.

Then add the following conditions:

- Jobform_1:recipient = Berger
- Jobform_1:subj is null

Fig. 3.5 shows the search mask.

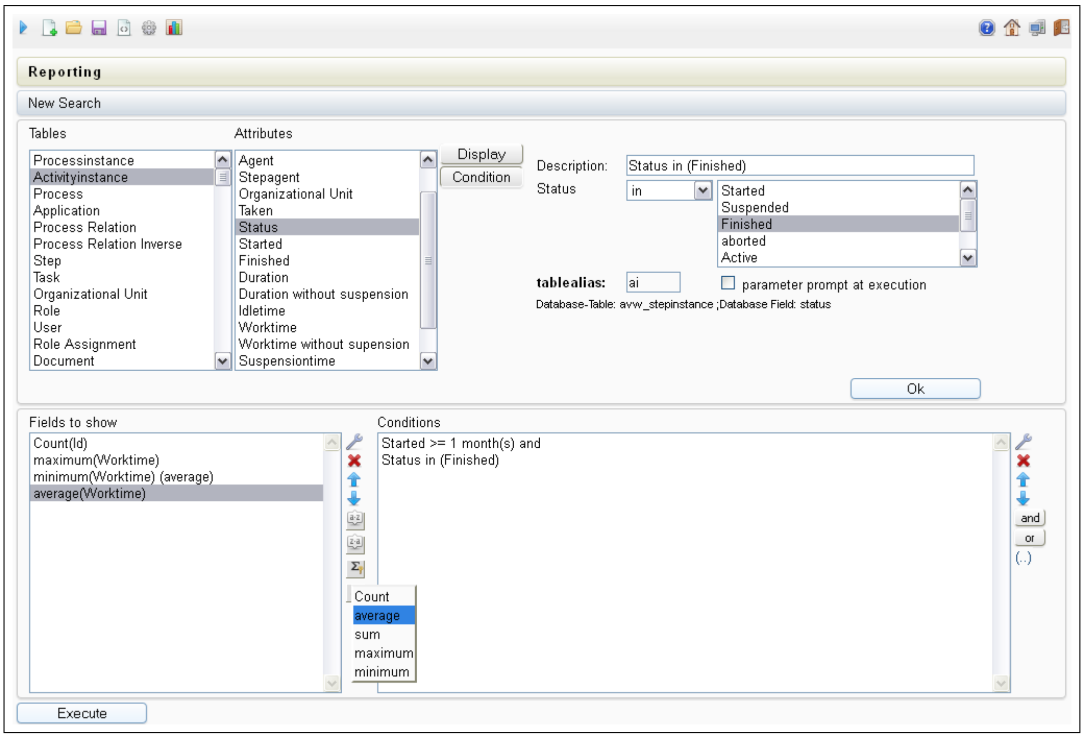
3.4 Example 4 (User-defined condition)

Which agents of aborted tasks can be reached via email?

We select the display field "ActivityInstance:Agent" and the following conditions:

- ActivityInstance:Status in (aborted)
- ai.agent in (select oid from avw_user where email is not null) ... a user-defined condition using a sub-query with the table avw_user. First select the entry *User defined* from the list *Tables*. Then activate the button *Condition* and select the entry *SQL-Command* from the dropdown-list *Operator*. Now you can enter the sql-statement in the field *Expression*.

Fig. 3.6 shows the search mask.



Tables

- Processinstance
- Activityinstance
- Process
- Application
- Process Relation
- Process Relation Inverse
- Step
- Task
- Organizational Unit
- Role
- User
- Role Assignment
- Document

Attributes

- Agent
- Stepagent
- Organizational Unit
- Taken
- Status
- Started
- Finished
- Duration
- Duration without suspension
- Idletime
- Worktime
- Worktime without supension
- Suspensiontime

Description: Status in (Finished)

Status: in — Started, Suspended, Finished, aborted, Active

tablealias: ai ☐ parameter prompt at execution

Database-Table: avw_stepinstance ;Database Field: status

Fields to show

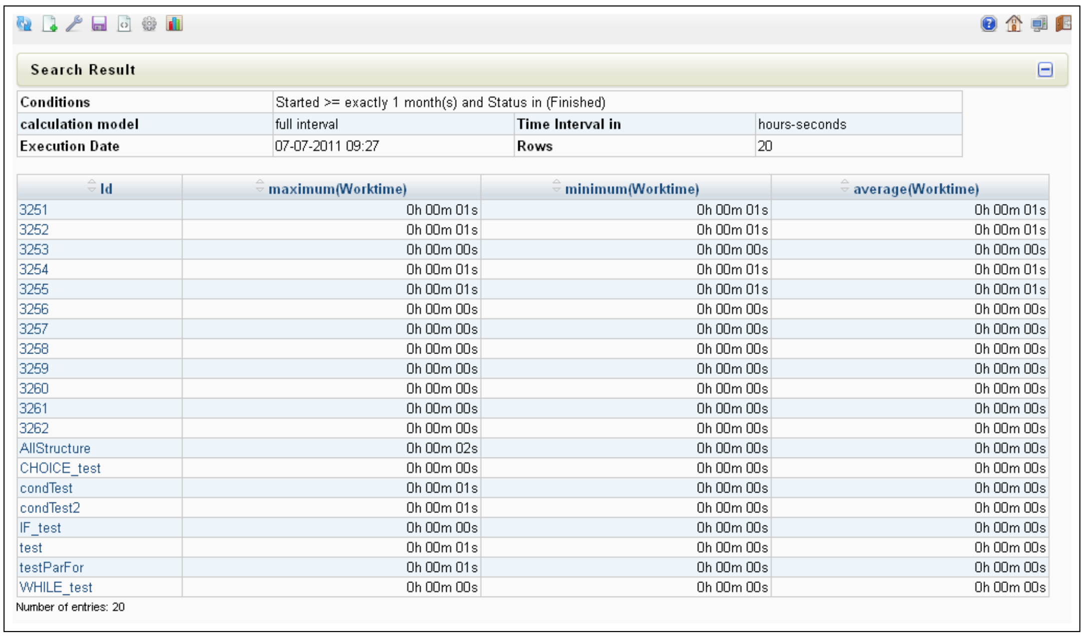
- Count(Id)
- maximum(Worktime)
- minimum(Worktime) (average)
- average(Worktime)

Count, average, sum, maximum, minimum

Conditions

Started >= 1 month(s) and
Status in (Finished)

Figure 3.1: **Search-mask to example 1**

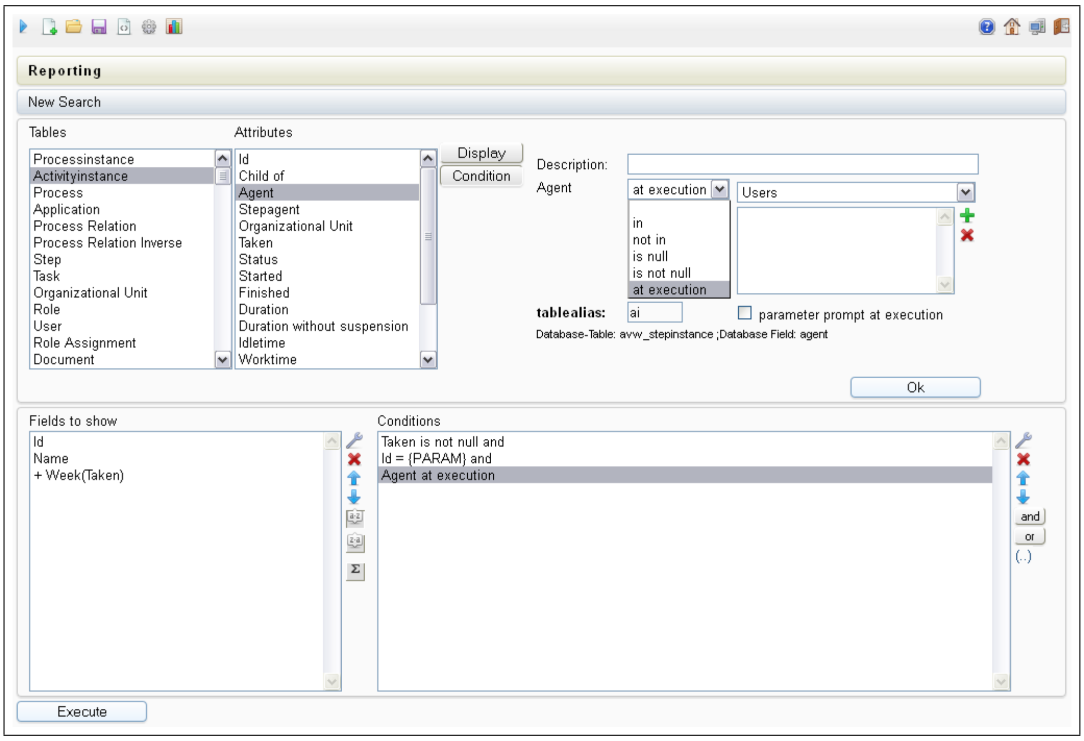


Conditions	Started >= exactly 1 month(s) and Status in (Finished)		
calculation model	full interval	**Time Interval in**	hours-seconds
Execution Date	07-07-2011 09:27	**Rows**	20

Id	maximum(Worktime)	minimum(Worktime)	average(Worktime)
3251	0h 00m 01s	0h 00m 01s	0h 00m 01s
3252	0h 00m 01s	0h 00m 01s	0h 00m 01s
3253	0h 00m 00s	0h 00m 00s	0h 00m 00s
3254	0h 00m 01s	0h 00m 01s	0h 00m 01s
3255	0h 00m 01s	0h 00m 01s	0h 00m 01s
3256	0h 00m 00s	0h 00m 00s	0h 00m 00s
3257	0h 00m 00s	0h 00m 00s	0h 00m 00s
3258	0h 00m 00s	0h 00m 00s	0h 00m 00s
3259	0h 00m 00s	0h 00m 00s	0h 00m 00s
3260	0h 00m 00s	0h 00m 00s	0h 00m 00s
3261	0h 00m 00s	0h 00m 00s	0h 00m 00s
3262	0h 00m 00s	0h 00m 00s	0h 00m 00s
AllStructure	0h 00m 02s	0h 00m 00s	0h 00m 00s
CHOICE_test	0h 00m 00s	0h 00m 00s	0h 00m 00s
condTest	0h 00m 01s	0h 00m 00s	0h 00m 00s
condTest2	0h 00m 01s	0h 00m 00s	0h 00m 00s
IF_test	0h 00m 00s	0h 00m 00s	0h 00m 00s
test	0h 00m 01s	0h 00m 00s	0h 00m 00s
testParFor	0h 00m 01s	0h 00m 00s	0h 00m 00s
WHILE_test	0h 00m 00s	0h 00m 00s	0h 00m 00s

Number of entries: 20

Figure 3.2: **Result to example 1**



Tables: Processinstance, Activityinstance, Process, Application, Process Relation, Process Relation Inverse, Step, Task, Organizational Unit, Role, User, Role Assignment, Document

Attributes: Id, Child of, Agent, Stepagent, Organizational Unit, Taken, Status, Started, Finished, Duration, Duration without suspension, Idletime, Worktime

Description:

Agent: at execution — Users

in, not in, is null, is not null, at execution

tablealias: ai — parameter prompt at execution

Database-Table: avw_stepinstance ;Database Field: agent

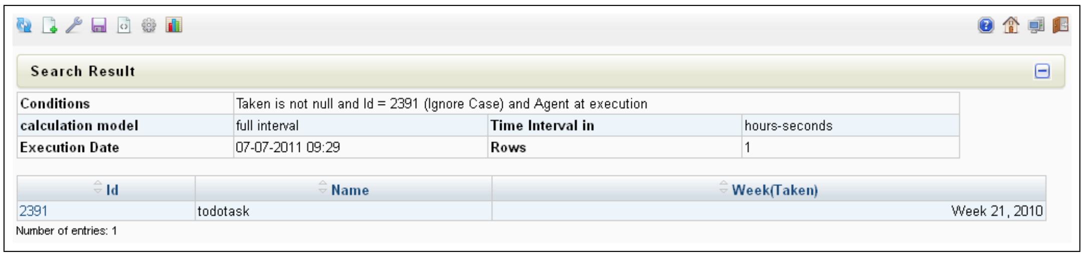
Fields to show:

Id
Name
+ Week(Taken)

Conditions:

Taken is not null and
Id = {PARAM} and
Agent at execution

Figure 3.3: **Search-mask to example 2**

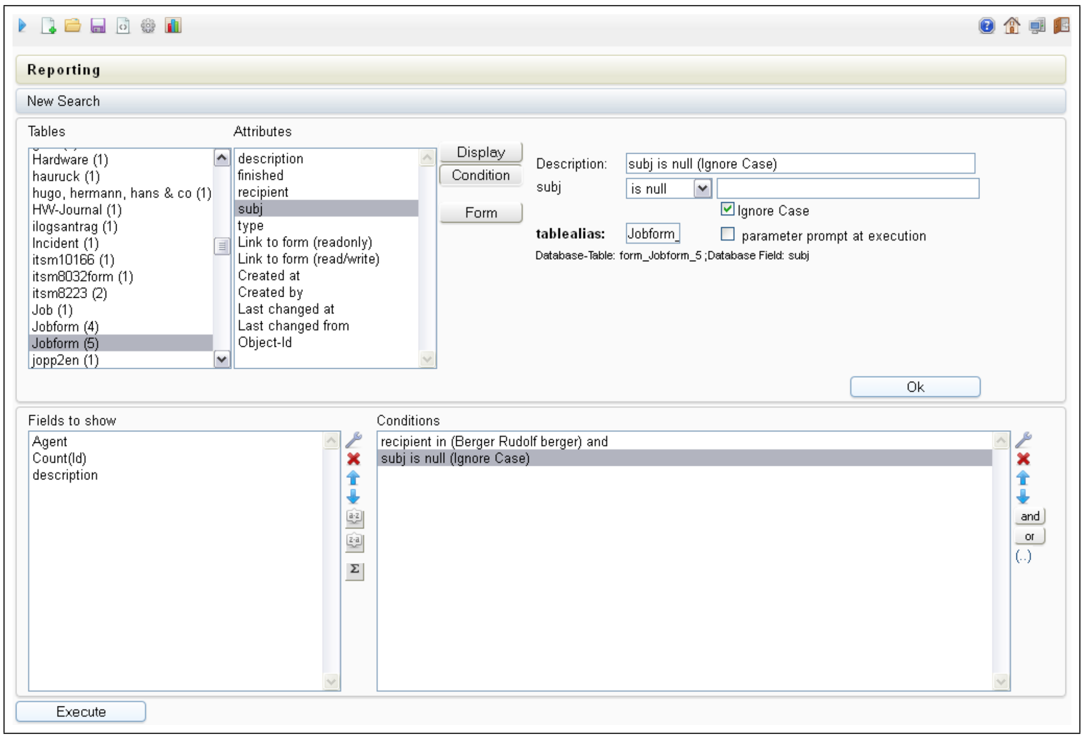


Conditions	Taken is not null and Id = 2391 (Ignore Case) and Agent at execution		
calculation model	full interval	**Time Interval in**	hours-seconds
Execution Date	07-07-2011 09:29	**Rows**	1

Id	Name	Week(Taken)
2391	todotask	Week 21, 2010

Number of entries: 1

Figure 3.4: **Result to example 2**



Tables:
Hardware (1)
hauruck (1)
hugo, hermann, hans & co (1)
HW-Journal (1)
ilogsantrag (1)
Incident (1)
itsm10166 (1)
itsm8032form (1)
itsm8223 (2)
Job (1)
Jobform (4)
Jobform (5)
jopp2en (1)

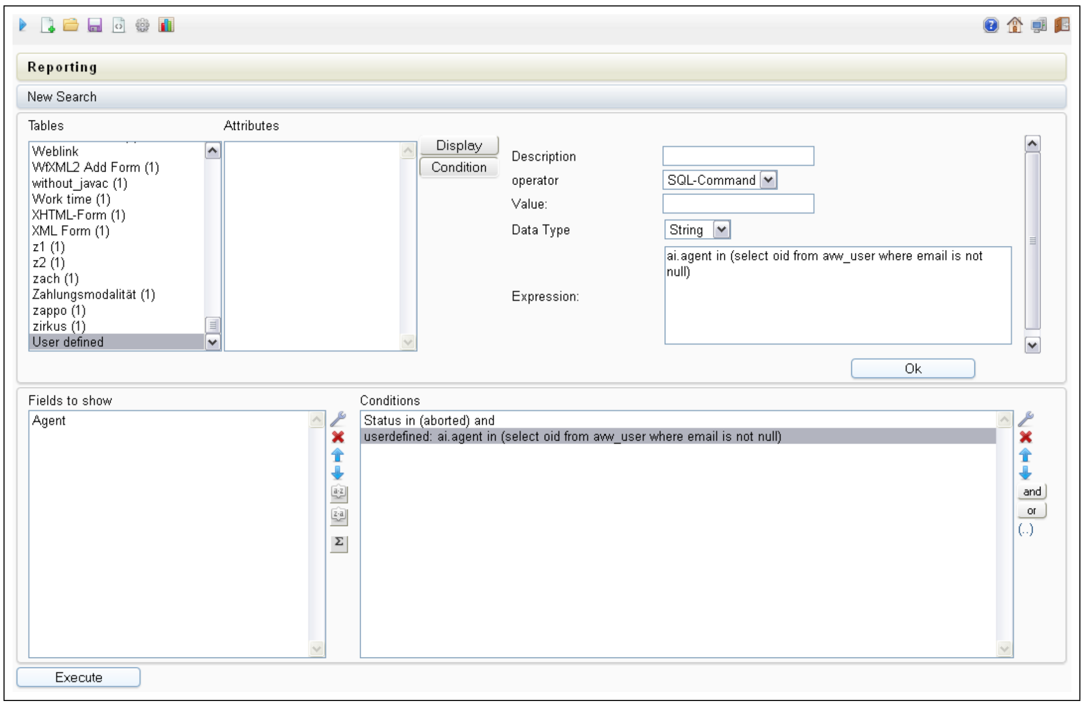
Attributes:
description
finished
recipient
subj
type
Link to form (readonly)
Link to form (read/write)
Created at
Created by
Last changed at
Last changed from
Object-Id

Description: subj is null (Ignore Case)
subj: is null
☑ Ignore Case
tablealias: Jobform
☐ parameter prompt at execution
Database-Table: form_Jobform_5 ;Database Field: subj

Fields to show:
Agent
Count(Id)
description

Conditions:
recipient in (Berger Rudolf berger) and
subj is null (Ignore Case)

Figure 3.5: **Search-mask to example 3**



Tables:
Weblink
WfXML2 Add Form (1)
without_javac (1)
Work time (1)
XHTML-Form (1)
XML Form (1)
z1 (1)
z2 (1)
zach (1)
Zahlungsmodalität (1)
zappo (1)
zirkus (1)
User defined

Attributes:

Description:
operator: SQL-Command
Value:
Data Type: String
Expression: ai.agent in (select oid from aw_user where email is not null)

Fields to show:
Agent

Conditions:
Status in (aborted) and
userdefined: ai.agent in (select oid from aw_user where email is not null)

Figure 3.6: **Search-mask to example 4**

4 Showing results graphically

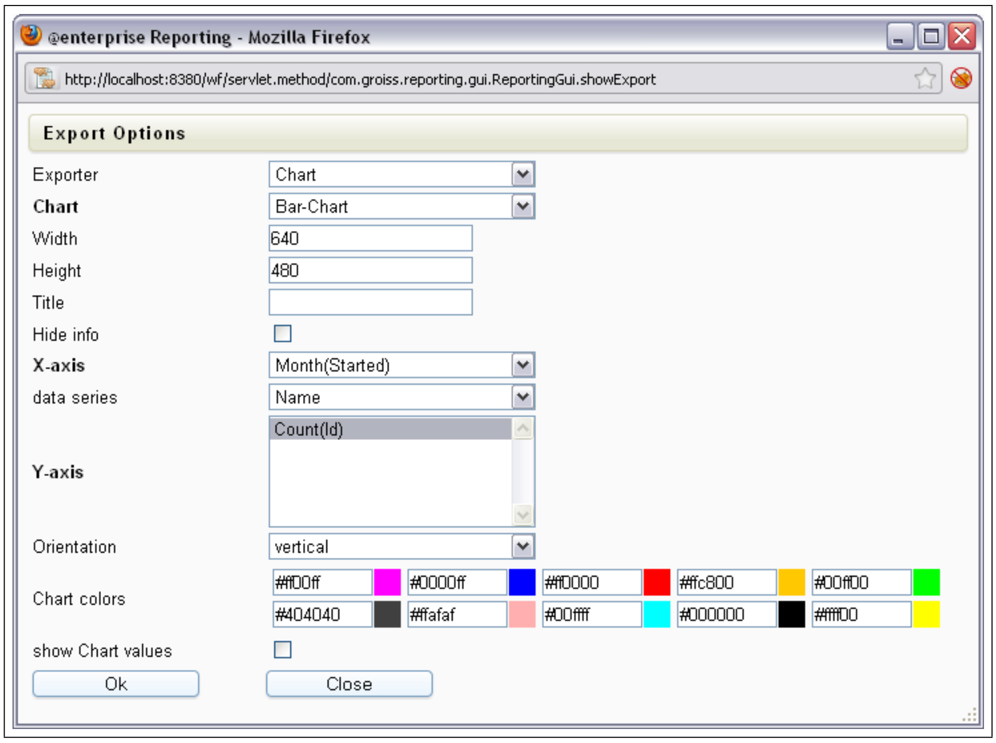
Additionally to the tabular output you can show the query results graphically. You can determine the chart settings before or after the execution.

A chart consists of categories and data series (see Fig. 4.2). Each row of the tabular view is a series. The categories are the values of the numerical display fields. Chosen categories are displayed as colored areas in a pie-chart.

With this function you can determine the charttype (see Fig. 4.1). Select the entry *Chart* from the dropdown-list *Exporter*. You can enter a chart-title and specify the *height* and *width* of the chart (is equivalent to the area inside the x- and y-axis). You can choose between following charttypes:

- Bar-Chart
- Pie-Chart
- Line-Chart
- Pie-Chart 3D
- Bar-Chart 3D
- Stacked-Bar-Chart
- Multiple-Pie-Chart

For all bar-charts you can select the *Orientation* (vertical or horizontal). You can also determine the data series or categories for the chart.

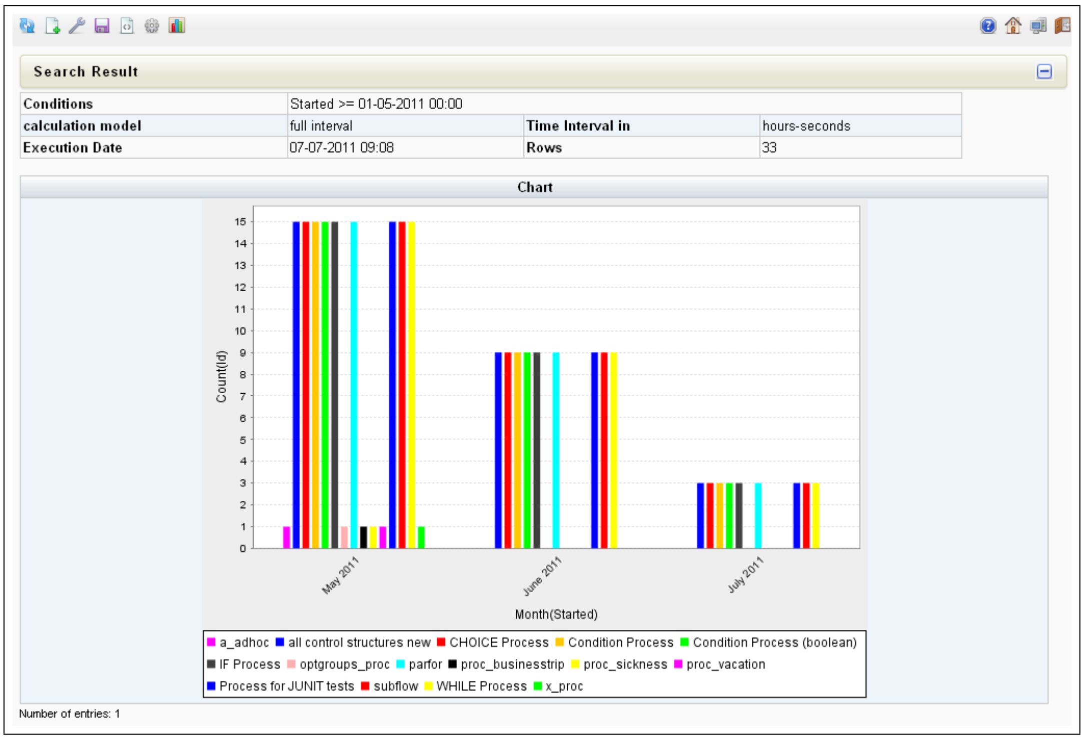

@enterprise Reporting - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.showExport

Export Options

Exporter	Chart
Chart	Bar-Chart
Width	640
Height	480
Title	
Hide info	☐
X-axis	Month(Started)
data series	Name
Y-axis	Count(Id)
Orientation	vertical
Chart colors	#ff00ff, #0000ff, #ff0000, #ffc800, #00ff00, #404040, #ffafaf, #00ffff, #000000, #ffff00
show Chart values	☐

Figure 4.1: **Diagram settings**


Search Result

Conditions	Started >= 01-05-2011 00:00		
calculation model	full interval	**Time Interval in**	hours-seconds
Execution Date	07-07-2011 09:08	**Rows**	33

Chart

Number of entries: 1

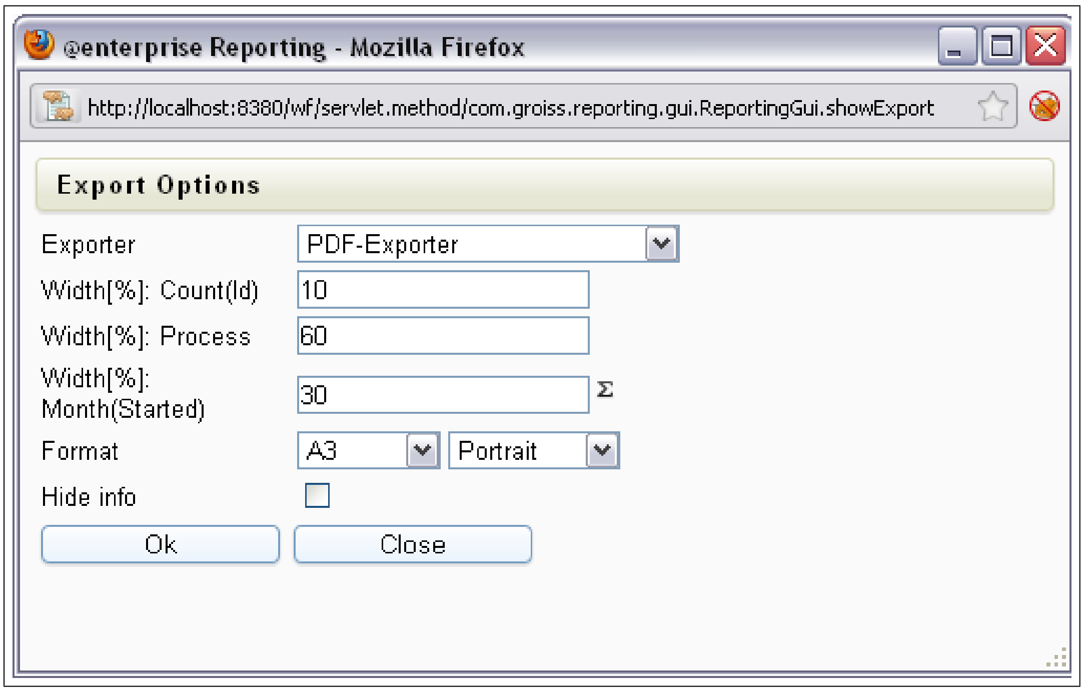
Figure 4.2: **Example diagram**

5 Export of query results

With this function it is also possible to export query results. You can select between following exporters:

- **HTML Table**: The result is shown in the browser (default). You can specify your own result page.
- **Chart**: see chapter 4
- **Export to Excel**: After activating the button *Ok*, the query will be executed and the result will be stored in a XLS-file, which can be downloaded.
- **Delimiter Separated Values - Exporter**: Similar to *Export to Excel*. Additionally you have to enter a *Delimiter* to separate the results. The target file is a CSV-file, which can be downloaded. If data fields contain the delimiter, its replaced by *@delimiter_removed@*
- **XML Export**: Similar to *Export to Excel*, whereas you can determine a style sheet. The target file is a XML-file.
- **PDF Export**: Report result will be written into a pdf file using the *itext* library. Page format an the widths of the columns (in percent of pagesize) are configurable. see fig. 5.1

Note that stored queries save the chosen export type. If you save the report e.g. with export excel, the report will be exported to excel by default.


@enterprise Reporting - Mozilla Firefox

http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.showExport

Export Options

Exporter	PDF-Exporter
Width[%]: Count(Id)	10
Width[%]: Process	60
Width[%]: Month(Started)	30 Σ
Format	A3 Portrait
Hide info	☐

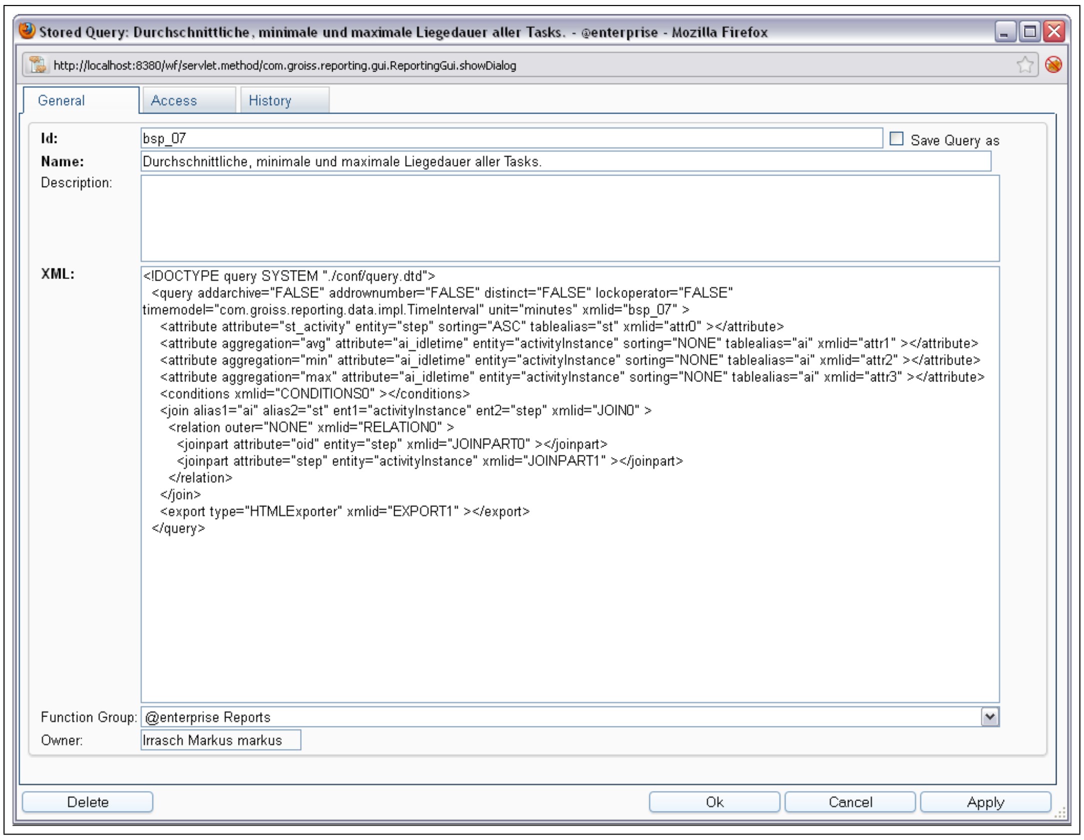
Figure 5.1: **Options of PdfExporter**

6 Administrate reports

If you want to reuse queries, it is possible to store it. Then you can open the query for executing, editing or deleting. Following functions are available:

- **New Search**: With this function you can create a new search. All fields will be cleared.
- **Open Stored Query**: After activating this function, a new HTML-page will be opened where you can select a stored query. Select a query and press the *OK* button to edit this query.
- **Save**: With this function you can save your query. Enter an *Id* and a *Name* and activate the button *Insert* (see Fig. 6.1). If you have already opened a stored query and activate this function, you can store this query with the same name (=*Update*) or delete it by activating the button *Delete*.
- **Details of Query**: In this mask the build query is shown in XML-format and as SQL-statement. If you want to see full details of a query, you have to activate this function in the result-mask.
- **Reportoptions**: This will open the report options mask. see chapter 2.3.
- **Chart**: See chapters 4 and 5.
- **Execute**: If you have activated this function, the created search will be executed and finally the result will be shown.
- **Edit Query**: This function is available in the creation- and parameter-mask only. With this function you can edit your build query.
- **Refresh**: This function is available in the result-mask only and refreshes the result.If queries with user defined parameters are refreshed with this function, the parameter input mask will not be displayed anymore.
- **Reload after new parameter input:** This function is available in the result-mask only when a query with user defined parameter inputs has been started and refreshes the result with the possibility to use other parameters.

- **Hide info**: This function is available in the result-mask only and fades in/out details of the query result. The Pdf Exporter uses the font *Helvetica*. This can be configured by a hidden configuration parameter. If the result shows Unicode signs, the configured font has to be a truetype font.



Stored Query: Durchschnittliche, minimale und maximale Liegedauer aller Tasks. - @enterprise - Mozilla Firefox
http://localhost:8380/wf/servlet.method/com.groiss.reporting.gui.ReportingGui.showDialog
General
Access
History
Save Query as
Delete
Ok
Cancel
Apply

Id: bsp_07

Name: Durchschnittliche, minimale und maximale Liegedauer aller Tasks.

Description:

XML:

```
<!DOCTYPE query SYSTEM "./conf/query.dtd">
  <query addarchive="FALSE" addrownumber="FALSE" distinct="FALSE" lockoperator="FALSE"
timemodel="com.groiss.reporting.data.impl.TimeInterval" unit="minutes" xmlid="bsp_07" >
    <attribute attribute="st_activity" entity="step" sorting="ASC" tablealias="st" xmlid="attr0" ></attribute>
    <attribute aggregation="avg" attribute="ai_idletime" entity="activityInstance" sorting="NONE" tablealias="ai" xmlid="attr1" ></attribute>
    <attribute aggregation="min" attribute="ai_idletime" entity="activityInstance" sorting="NONE" tablealias="ai" xmlid="attr2" ></attribute>
    <attribute aggregation="max" attribute="ai_idletime" entity="activityInstance" sorting="NONE" tablealias="ai" xmlid="attr3" ></attribute>
    <conditions xmlid="CONDITIONS0" ></conditions>
    <join alias1="ai" alias2="st" ent1="activityInstance" ent2="step" xmlid="JOIN0" >
      <relation outer="NONE" xmlid="RELATION0" >
        <joinpart attribute="oid" entity="step" xmlid="JOINPART0" ></joinpart>
        <joinpart attribute="step" entity="activityInstance" xmlid="JOINPART1" ></joinpart>
      </relation>
    </join>
    <export type="HTMLExporter" xmlid="EXPORT1" ></export>
  </query>
```

Function Group: @enterprise Reports

Owner: Irrasch Markus markus

Figure 6.1: **Save query**

7 Configuration of the reporting component

7.1 Permissions

You need the right *Statistic* for creating reports and select all fields.
You can define which user may execute which query. One possibility to define this, is when you save the query (see above). It you want to give a user or a role the right to execute a stored query, go the the list of users or roles, click on permissions and add a new permission: Select the right "execute", the object class "Stored Queries" and select the query you want. To enable user to set permissions to stored queries, the user needs the right "edit permissions".
Note that in the list of stored queries every user sees only the queries he may execute. The creator of a report has the permission to edit, delete, execute and assign permissions to his report.

7.2 Public execution (without login)

Reports are executable without being logged in if the user *guest* is allowed to execute the report. The URL below will show you a list of executable reports. Log out before calling the url to check public queries.

```
http://<host>:<port>/wf/servlet.method/com.groiss.reporting.
HTMLStoredQuery.listQueries
```

Add the parameter `groupId=<idOfFunctionGrpup>` to get a list of only one functiongroup.

7.3 Version independent views for forms

If you want to create queries on forms version-independent, you need to create views over the form versions. To do so, go to the form administration, select a form and click the button "Create View". You will see the SQL-statement you can now execute. For further information how to create a view, please take a look in the *System Administration Guide*.

7.4 Server Configuration

The following configuration parameter affect the reporting component.

- Maximum Table Size on Server (rows): Due to the possible very large amount of report results, reporting may cause performance problems. To avoid this, you can specify a maximum amount of lines. If the report returns more results, the report is canceled and a exception is thrown.
- Cache Interval (minutes): To avoid high server load its able to cache query results. This parameter defines how long a query should be cached.
- Maximum Number of Cached Queries: This parameter defines the size of the cache.
- Maximum Number of Simultaneous Queries: To ensure that reporting does not cost too much performance, this parameter limits the number of queries which can be executed concurrently.
- Maximum Number of Startable Queries: Defines the amount of queries which are queued if the maximum number of simultaneous queries is exceeded.
- Order Process-Ids by OID: When checked, ProcessIds are sorted by the Processoid.
- Show all rows, even when no View Right: DMSObjects in result set will not be displayed if the user who executes the report has no view right on the DMSObject. This feature can be disabled by activating this checkbox.
- Open Forms in Edit Mode: If checked forms links (attribute: oid) open the forms editable.

Hidden configuration parameters are documented in chapter 8.

7.5 Reporting-Cache

Changes in Persisten-Objects will not effect reporting results if the result was found in reporting cache. In this situation reexecute the report result with the reload toolbarfunction or reinitalize the reporting engine (Administration -> Admin Tasks -> Server Control)

8 Developers Guide

This part of document shall give an overview about the possibilities to customize the reporting component.

8.1 Hidden Configuration

Reporting uses 2 hidden parameter in configuration file.

- itext.pdf.font: Path to font file to use in pdf files. If report includes Unicode characters, this font has to be a truetype font.
- avw.reporting.schemaxml: Path to the system schema xml file. Needed to ensure compatibility to version 7.0 when reporting.xml was edited to fit application requirements. If possible don't overwrite schema xml but use the new merging of system schema xml and application schemas.

8.2 XML Configuration

The reporting uses XML-documents to declare the choiceable data on the one hand and the chosen query on the other hand. To understand, how to customize the reporting, a closer look on the XML specification is needed.

8.2.1 Schema

The schema file contains all needed information about the pool of data, which can be used in reports. Reporting merges the configured schema file with any file named ”reporting.xml” in the configured application paths.

```
<!ELEMENT Schema (mapping*,entity+,relation*)>
<!ATTLIST Schema xmlid ID #REQUIRED
name CDATA #IMPLIED
furtherHops CDATA #IMPLIED
defaultTimemodel CDATA #IMPLIED
defaultTimeUnit CDATA #IMPLIED
addForms (TRUE | FALSE) "FALSE">
```

Example:

```
<schema xmlid="avw" name="@@@reporting@@"
    furtherHops="2"
    defaultTimeUnit="hours"
    defaultTimemodel="com.groiss.reporting.data.impl.TimeInterval"
    addForms="TRUE">
....
</schema>
```

Mappings

Mappings are used to translate data into their natural meaning, e.g. ActivityInstance Status is a number and each number means a status. The reporting user does not want to know the number but the statusname. So this translation is made with a mapping. Mappings are defined once and can be referenced often by their id. Mappings are used to define which exporter, charts and timemodels shall be available for users. Even the possible implementation of an HasSubClass Persistent are defined by a mapping.

```
<!ELEMENT mapping (mapentry+)>
  <!ATTLIST mapping xmlid ID #REQUIRED>
<!ELEMENT mapentry EMPTY>
  <!ATTLIST mapentry
    key CDATA #REQUIRED
    value CDATA #REQUIRED>
```

Example 1: Mapping for Translating status keys

```
<mapping
  xmlid ="aiStatus">
  <mapentry key="0" value="@@@started@@" />
  <mapentry key="1" value="@@@suspended@@"/>
  <mapentry key="2" value="@@@finished@@"/>
  <mapentry key="4" value="@@@aborted@@"/>
  <mapentry key="5" value="@@@active@@"/>
  <mapentry key="6" value="@@@waiting@@"/>
  <mapentry key="7" value="@@@compensated@@"/>
</mapping>
```

Example 2: Mapping for Subclasses of Agent

```
<mapping xmlid="agentSubclasses">
  <mapentry key="c1" value="com.dec.avw.core.User" />
  <mapentry key="c2" value="com.dec.avw.core.Role" />
  </mapping>
```

Entity

Entities are representing selections of tables in the database. One table can be defined as several entities if needed. In the system scheme the avw_stepinstance table is defined once as ProcessInstance (with selection type=22) and once as ActivityInstance (with selection type=20). Entities contains at least one attribute and several selections.

```
<!ELEMENT entity (attribute*,selection*)>
  <!ATTLIST entity xmlid ID #REQUIRED
    table CDATA #REQUIRED
    class CDATA #REQUIRED
    name CDATA #IMPLIED
    tablealias CDATA #REQUIRE>D
<!ELEMENT selection EMPTY>
  <!ATTLIST selection
    attribute CDATA #REQUIRED
    operator CDATA #REQUIRED
    value CDATA #REQUIRED>
```

Example: Entity of Processdefinition

```
<entity
    xmlid="process"
    name="@@@objectname_processdefinition@@"
    class="com.dec.avw.core.ProcessDefinition"
    table="avw_procdefinition"
    tablealias="pd">
....
</entity>
```

Attribute

Attributes can be attached to the query as select attribute or as condition. The way a attribute is stored in report result, is defined in an implementation of the ReportingData Interface. Attributes may contain several selects to gain data from the database (e.g. a TimeInterval needs at least two timestamps).

```
<!ELEMENT attribute (select*)>
  <!ATTLIST attribute
    xmlid CDATA #REQUIRED
    name CDATA #IMPLIED
    mapping IDREF #IMPLIED
    aggrs CDATA #IMPLIED
    class CDATA #IMPLIED>
<!ELEMENT select (#PCDATA)>
  <!ATTLIST select entity CDATA #IMPLIED
     tablealias CDATA #IMPLIED>
```

Relations

Relations define how entities can be joined. A relation has a name (which is displayed in join select mask) and two joinparts. Additionally a outer join can be specified.

```
<!ELEMENT relation (joinpart,joinpart)>
  <!ATTLIST relation name CDATA #IMPLIED
    outer (LEFT | RIGHT | NONE) "NONE">
<!ELEMENT joinpart EMPTY>
  <!ATTLIST joinpart entity CDATA #REQUIRED
    attribute CDATA #REQUIRED>
```

8.2.2 Query

Queries are defined in XML documents, too. This XML tree is generated while configuring the query options at the extended search mask. But you can even write it manually. A query consists of 3 parts: The select attributes, a condition tree and the joins. Joins have to be declared because the reporting engine provides the possibility to join entities over several ways.

```
<!ELEMENT query (attribute*,conditions?,join*,export?)>
<!ATTLIST query xmlid ID #REQUIRED
 unit CDATA #IMPLIED
 minunit CDATA #IMPLIED
 timemodel CDATA #IMPLIED
 timezone CDATA #IMPLIED
 locale CDATA #IMPLIED
 parammask CDATA #IMPLIED
 lockoperator (TRUE|FALSE) "FALSE"
 distinct (TRUE|FALSE) "FALSE"
 addarchive (TRUE|FALSE) "FALSE"
 addrownumber (TRUE|FALSE) "FALSE">
```

Parameter

Parameters are used as child nodes of attributes and exporters. Its a key value pair which may be used to store additional data.

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter xmlid ID #IMPLIED
  key CDATA #REQUIRED
  value CDATA #REQUIRED>
```

Attributes

Attributes in Query-XML refer to attributes in the schema.

```
<!ELEMENT attribute (parameter*)>
<!ATTLIST attribute
xmlid ID #IMPLIED
displayname CDATA #IMPLIED
entity CDATA #IMPLIED
tablealias CDATA #IMPLIED
attribute CDATA #REQUIRED
aggregation CDATA #IMPLIED
type CDATA #IMPLIED
dateformat CDATA #IMPLIED
select CDATA #IMPLIED
others CDATA #IMPLIED
sorting (ASC |DESC |NONE) "NONE">
```

Conditions

The conditions of the query are defined as a tree of condition-elements, connectors and parentheses. The defined root element is the Conditions-element, which contains one or no condition or parentheses element and several pairs of connectors and conditions/parentheses.

```
<!ELEMENT conditions ((condition|parentheses)?,
                (connector,(condition|parentheses))*)>
<!ATTLIST conditions
xmlid ID #IMPLIED>
<!ELEMENT parentheses ((condition|parentheses)?,
                (connector,(condition|parentheses))*)>
<!ATTLIST parentheses
xmlid ID #IMPLIED>
<!ELEMENT connector EMPTY>
<!ATTLIST connector
xmlid ID #IMPLIED
type (AND | OR ) "AND">
<!ELEMENT condition EMPTY>
<!ATTLIST condition xmlid ID #IMPLIED
entity CDATA #REQUIRED
tablealias CDATA #REQUIRED
attribute CDATA #REQUIRED
displayname CDATA #IMPLIED
operator CDATA #IMPLIED
value CDATA #IMPLIED
type CDATA #IMPLIED
paramatexec (TRUE|FALSE) "FALSE"
others CDATA #IMPLIED>
```

Join

The selected joinpaths for the usage in report are stored in the xml. The joins have to build a graph so that every entity is connected to an other entity. A join can include several relations, which can include entities which are not referenced at the attributes or conditions of the report, too. Every relation is used to do the correct join. If the entity of a joinpart is not used in the report, the standard alias defined in schema is used.

```
<!ATTLIST join
  xmlid ID #IMPLIED
  ent1 CDATA #REQUIRED
  ent2 CDATA #REQUIRED
  alias1 CDATA #REQUIRED
  alias2 CDATA #REQUIRED>
<!ELEMENT relation (joinpart,joinpart)>
<!ATTLIST relation
  xmlid ID #IMPLIED
  outer (LEFT | RIGHT | NONE) "NONE">
<!ELEMENT joinpart EMPTY>
<!ATTLIST joinpart entity CDATA #REQUIRED
  xmlid ID #IMPLIED
  attribute CDATA #REQUIRED>
```

Export

As described in Chapter 5 the export options are stored in xml, too. The export element can have several parameters (key-value pairs) to configure export options and drill-down functions. The type attribute includes the classname, if no package is defined the default package will be used.

```
<!ELEMENT export (parameter*)>
<!ATTLIST export type CDATA #REQUIRED
  xmlid ID #IMPLIED>
```

8.3 API

Sometimes the engine has to be extended to fit requirements application . Therefore an API was designed to enable project specific reports.

8.3.1 com.groiss.reporting.data.TimeModel

TimeModels are calculating TimeIntervals between 2 Date objects. Implement the getInterval Method, which returns the interval in milliseconds as a long value.

```
public interface TimeModel {

   public long getInterval(Date start,Date end);
   public String getModelName();
}
```

Register your TimeModel in reporting schema in the mapping *timemodels* to mark it as selectable in reporting designer.

Example: DemoTimeModel Calculates the milliseconds between two date objects.

```
public long getInterval (Date start, Date end) {

  long time1 = start.getTime();
  long time2 = end.getTime();
  return (time2-time1);
}
public String getModelName() {
  return "DefaultTimeModel" ;
}
```

8.3.2 com.groiss.reporting.data.ReportingExportable

Exporters of reporting must handle objects implementing this interface. The getValue Method returns the valueholder object, toHTML the HTML representation (has to be a `com.groiss.gui.Component` or a `String` Object) of the value and toText a text representation to use for CSV or Excelexport.

```
public interface ReportingExportable extends Comparable {
   public String toText();
   public Object toHtml();
   public Object getValue ();
}
```

Example will be given in Chapter 8.3.3.

8.3.3 com.groiss.reporting.data.ReportingData

This interface extends the ReportingExportable and provides methods to overwrite the attribute's behavior when added to a report. We recommend to extend default implementation *com.groiss.reporting.data.impl.DefaultReportingData*.

```
public interface ReportingData extends ReportingExportable {
   public Page getAttributePage(String id, Document query);
   public Page getConditionPage(String id, Document query);
   public List addParamAtExecuteRow(int count, Element c,
        StringBuffer javascript,Query q);
   public Attribute getAttribute();
   public void setAttribute(Attribute a);
   public Entity getEntity();
   public void setEntity(Entity e);
   public void addSelectAttributeToQuery(Query q, Element select);
   public void addConditionToQuery(Query q, Element c);
   public void setValue (ResultSet rs, Element selectAttribute,
               Query q);
}
```

getAttributePage has to return a HTML Page to specify attribute options. Has to include a javascript function *add()*, which calls *parent.ep.reporting.gui.addAttribute(id,entityId , attributeId , tableAlias , displayName , aggregation , datumformat , sorting, type,select,others)* or *parent.ep.reporting.gui.editAttribute(id,entityId , attributeId , tableAlias , displayName , aggregation , datumformat , sorting, type,select,others)* in edit mode.

getConditionPage has to return a HTML Page to specify condition options. Has to include a javascript function *add()*, which calls *parent.ep.reporting.gui.addCondition(id,entityId , attributeId , tableAlias , displayName , operator, paramValue , others, type ,paramAtExecute)* or
parent.ep.reporting.gui.editCondition(id,entityId , attributeId , tableAlias , displayName , operator, paramValue , others, type, paramAtExecute) in edit mode.

addParamAtExecuteRow method is called, if engine determines that the condition is not complete. Engine expects a 3 value list as return value which is used as row of a htmltable. The input components have to be named with "value"+count, "operator"+count and - if needed - "others"+count. Element c is the DOM Element of type condition, which is not complete. If some javascript statements to manipulate users input are needed you can add it to the StringBuffer *javascript*.

getAttribute,setAttribute,getEntity,setEntity set and return the entity and attribute object of remating scheme which are referenced this ReportingData Object.

addSelectAttributeToQuery has to implement, how attribute is added to the select statement of SQL Query. Has to call *com.groiss.reporting.engine.Query.addSelect* and *com.groiss.reporting.engine.Query.addSelectNameOfAttrib(Element, String)* to register the attribute!

addConditionToQuery has to implement the logic how attribute is added to the conditions of SQL Query. Has to call *com.groiss.reporting.engine.Query.addCondition*.

setValue gains the data from ResultSet and stores it.

Example: com.groiss.demo.DemoReportingData The Goal is to add a selectable attribute to reportingschema, which indicates the average progress of all orderitems Therefore you have to add following attribute to reporting schema in the entity "demo_order_1" The output shall be the average progress with a '%' sign afterwards. if its HTML it shall be a link which does a javascript alert.
We want the engine to allow a field which calculates the product of amount and price at execution time of report. Therefore you have to add following attribute to reporting schema in the entity "bill":

```
<attribute xmlid="averageprocess"
   name="@@@averageprocess@@"
   class="com.groiss.demo.ProgressReportingData">
   <select entity="demo_orderitem_1" tablealias="avg_oderitem_1">
```

```
avg(avg_oderitem_1.amount)
</select>
 </attribute>
```

This column shall also do conditions on the product of amount and price. The output shall be the product with a Dollar sign afterwards. His HTML representation shall be a Link with a javascript alert showing the product. Cause of extending DefaultReportingData we do not need to implement a getAttributePage, getConditionPage or addSelectToQuery method.
You will find a Demoimplementation in the demo package of @enterprise.
(see: *com.groiss.demo.reporting.ProgressReportingData*)

8.3.4 com.groiss.reporting.data.NumericValue

ReportingData Objects have to implement this interface, if the value is aggregateable but does not fit to the standard dataobjects.

```
public interface NumericValue extends ReportingData {
  public NumericValue add(NumericValue v2);
  public NumericValue avg(long count);
}
```

8.3.5 ReportingExporter

Reporting Engine returns a Tablemodel containing ReportingExportable Objects as result of a report. The output format of this Tablemodel can be modified by implementing an own Exporter. To mark exporter as selectable in reporting designer, add it to the mapping "exporter" in reporting.xml.

```
public interface ReportingExporter {
   public String getExportName() ;
   public void addConfigOptions(Query q, XHTMLPage p);
   public void export(HttpServletResponse res, Query q,
            TableModel tm) throws Exception;
}
```

getExportName returns the Name of this Exporter. Is displayed in export option page to select exporter.

addConfigOptions enables exporter to add option fields to the export option page. These parameter can be read by calling *q.getExportParam(key)*.

export Iterate over the tablemodel and manipulate data like needed for export.
You will find a demoimplementation in the demo package of @enterprise.
(see *com.groiss.demo.reporting.FileSystemExporter*)

8.4 Implementing your own Search Mask

The recommended way is to build a stored query in reporting designer, which includes *paramenter-at-execution* conditions for each field of the search mask. In the search mask for each condition a *value*-field, an *operator*-field and in some cases an *others*-field are needed to complete conditions. The engine expects post parameters called *value0*, *operator0* and *others0* for explicit parameter substitution. 0 stands for the index of the condition in the conditions tree starting with 0. So if the first and the third condition of the report need explicit parameter input, the fields *value0*, *operator0*, *others0*, *value2*, *operator2* and *other2* are expected. If needed you may name the condition with a parameter tag (key="paramname"), so that you may reference the parameter with *paramname_value*, *paramname_operator* and *paramname_others*. Add a field *comesFromParamMask* with value "1" to the search mask, so that engine expects the parameter in the post parameters. A demoparammask, which fits the minimal requirements of reporting is placed in *demos/classes/alllangs/demo/reporting/demoparammask.html*. Fell free to use this file to start and extend it with the controls you need. Configure report options to set this search mask as param mask. Execute the query. and the param mask will be shown.

Attribute	**Description**
xmlid	The Id of the schema
name	The name of the schema, can be localized
furtherhops	To determine the pool of possible Joins for 2 entities, the shortest join path is searched! This parameter delimits the amount of selectable joins to all joins, which need not more join hops than the shortest path plus this parametervalue
defaultTimemodel	The default timemodel, which should be used for reports to calculate Timeintervals. Must implement the *com.groiss.reporting.data.TimeModel* interface.
defaultTimemodel	The default timeunit of timeintervals
addForms	Forms are not declared in the schema file by default. They are added automatically during the parsing, if this flag is set to true!

Table 8.1: Description of element Schema

Attribute	**Description**
xmlid	The Id of the map is used to reference it in the attributes.
key	The key of this entry. If Mapping is used for translation, the key is the value in the database.
value	The string representing the database value or the full classname of the exporter, charttype or timemodel.

Table 8.2: Description of element mapping

Attribute	**Description**
xmlid	The Id of the entity is used to reference it in the query xml tree.
table	The name of the database table.
class	The persistent implementation representing this table in @enterprise. This is needed to get information about the fieldtypes.
name	The name of this entity. May include I18N keys, so each string starting with @@@ and ending with @@ is replaced.
tablealias	The default tablealias for this entity. It can be overwritten by the query xml document.
selection	The selection defines a condition to restrict the data tuples of this entity. The selection consists of an (database-)attribute name, an operator and a value. (e.g: ActivityInstance: selection: attribute="type";operator="=";value="20")

Table 8.3: Description of element entity

Attribute	**Description**
xmlid	The Id of the attribute is used to reference it in the query xml tree.
name	The name of this attribute. May include I18N keys, so every string starting with @@@ and ending with @@ is replaced.
class	The implementation of the *com.groiss.reporting.data.ReportingData* Interface. If not specified the Default-Implementation is used.
mapping	The id of the referenced mapping to translate data or to know the possible implantations of HasSubclasses Persistents.
aggrs	The selectable aggregations for this attribute. If not specified, the aggregations are calculated depending on the field type. But sometimes it does not make sense to calculate an average of a numeric data (e.g. Process:Version).
select	The select Attribute contains the name of the database field. If a select is needed from an other entity, the attributes entity and tablealias are specified to gain this additional information. (e.g.: StepDuration needs start and end time of ActivityInstance)

Table 8.4: Description of element attribute in schema

Attribute	**Description**
entity	The entity id of this joinpart.
attribute	A DB-field which is used to join.

Table 8.5: Description of element relation

Attribute	**Description**
xmlid	The Id of the query
unit	The maximum timeunit of timeintervals. Possible values: ”seconds",”minutes",”hours",”days",”weeks"
minunit	The minimum timeunit of timeintervals.
timemodel	The timemodel, which should be used for this report to calculate Timeintervals. Has to implement the `com.groiss.reporting.data.TimeModel`!
locale	If a Report should be executed in a specific language, the shortname locale is defined here! (e.g. en_US)
timezone	If the report should be executed in a specific timezone, the id of the timezone is defined here
parammask	The path to an alternativ paramask. Customized Parameter mask has to fit all naming conventions.
lockoperator	If this flag is set to *TRUE*, the operator selectlist in parameter at execution mask are displayed readonly. This affects standard parameter mask but not a customized one.
distinct	If this flag is set to true, only equal tuples in the result-set are not displayed.
addarchive	This flag has to be true, if the report shall include avw_stepinstance records from the archive schema!
addrownumber	This flag has to be true, if the report shall include rownumbers in the first column. (Note that this needs a special treatment when implementing an Exporter!)

Table 8.6: Description of element query

Attribute	**Description**
xmlid	An optional field, which has to declare an unique id for this attribute. If not specified, the engine set a unique id automatically. The id is used for referencing the attribute at the edit mode.
displayname	The name of this column. If not specified, the default attribute name from schema is the column name. Can include I18N keys
entity	The id of the referenced entity, which contains the referenced attribute in schema. If its a user-defined attribute, the tablename in the database is stored in the entity field.
tablealias	The tablealias for this entity. Entity-Id and tablealias are the unique id of an entity in the query. For each entity a join has to be declared.
attribute	The id of the referenced attribute of the schema. If its an user-defined attribute, its set to ”userdefined"!
select	A sql-syntax fitting expression, which is copied in the select statement.
type	If select expression returns not the default type of the attribute, which is declared in schema, define full-qualified class name here. If its `com.groiss.reporting.data.impl.TimeInterval` 2 date type are suggested.
aggregation	The aggregation for this attribute. Aggregations are grouped by all non aggregated attributes in the result. If the select includes an sql aggregation, specify here ”sqlaggr".
sorting	Can be ”ASC” for ascending, ”DESC” for descending or none for no sorting. Sorting is done as the attribute are ordered, so the sorting of the first attribute has an higher priority than the second one.
others	This is optional wildcard attribute. It can be used for different things. The default data types interpret the others attribute as an user-defined selection due to the entity

Table 8.7: Description of element attribute in query

Attribute	**Description**
xmlid	An optional field, which has to declare an unique id for this attribute. If not specified, the engine will set an unique id automatically. The id is used for referencing the element at the edit mode.
connector: type	Type of boolean operation. Can be AND or OR, AND is default!
entity	The id of the referenced entity, which contains the referenced attribute in schema. If its a user-defined attribute, the tablename in the database is stored in the entity field.
tablealias	The tablealias for this entity. Entity-Id and tablealias are the unique id of an entity in the query. For each entity a join has to be declared.
attribute	The id of the referenced attribute of the schema. If it is an user-defined attribute, it will be set to ”userdefined"!
displayname	A description of this condition. May include I18N keys.
operator	The operator of this condition, for example in, like, = or >! If it is an user-defined SQL Condition, operator contains the expression, which may be written in PreparedStatement syntax.
value	Value for prepared Statements. Several values can be separated by a comma.
type	The type of the value string for parsing it to the fitting object.
others	This is optional wildcard attribute. It can be used for different things. DateReportingData Objects for example suggest unit here if a relative date condition is specified. Text condition store the ignorecase option here.
paramatexec	True if parameter at execution is needed. In this case the operator and the value is used as default parameters.

Table 8.8: Description of elements of conditions tree

Attribute	**Description**
ent1	The entity id of the source entity of the join
alias1	The tablealias of the source entity of the join
ent2	The entity id of the target entity of the join
alias2	The tablealias of the target entity of the join
other fields	see 8.2.1

Table 8.9: Description of element join

9 Support

For further questions, contact us under the email `support@groiss.com`.