@enterprise *9.0*

Reporting

January 2023

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 9.0.33982

Contents

1 Introduction

With the Reporting component (Report designer) you can create complex statistics and analyses on runtime data using a simple to use search mask. Reports 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 "Report designer" allows you to create reports, the link "Reports" 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. The search mask is divided into following tabs (see fig. 2.1):

- Attributes (display fields): see section 2.1
- Conditions: see section 2.2
- Joins: see section 2.3
- SQL report: This tab is available only, if the toolbar function *SQL report* within the Report Designer has been selected (see section 2.6). The tabs *Attributes*, *Conditions* and *Joins* are not available in this case!
- Export: see section 2.4
- Options: see section 2.5
- Preview: This tab contains a preview of the current report

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 "Add" button. 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.
- **Active tasks information**: Tasks and available due dats, and agents of the currently running steps of the process.
- **Subject**: Subject of the process instance.
- **Application**: The application the process belongs to.

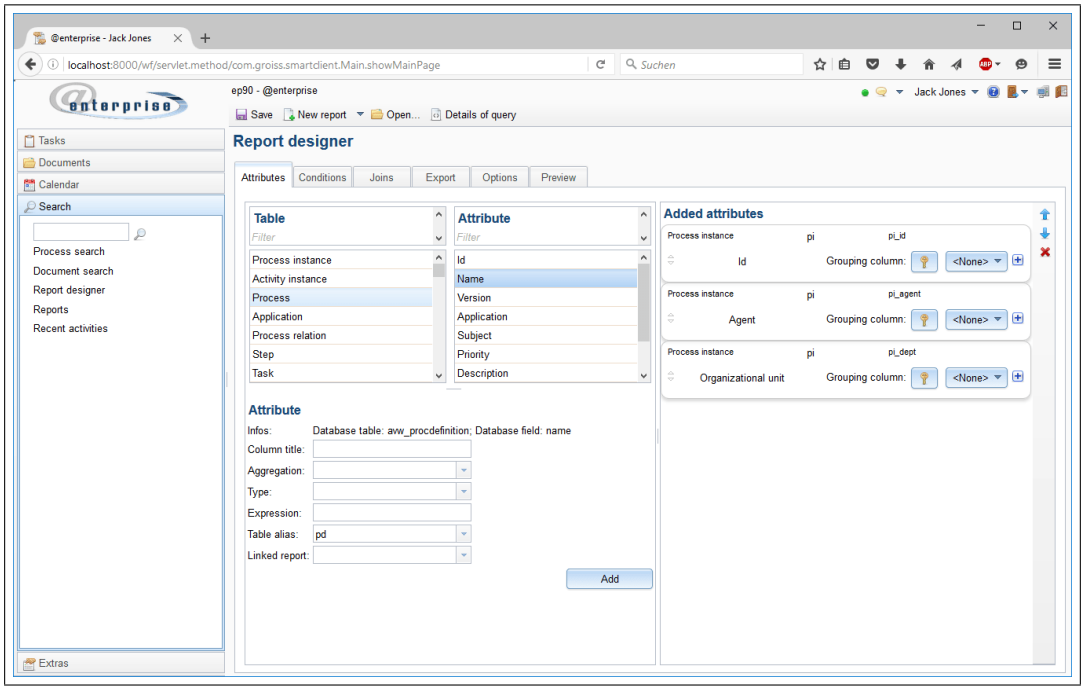

Figure 2.1: **Search Mask**

- **Process**: Name of the process.
- **Agent**: Agent who started the process.
- **Organizational Unit**: Organizational Unit where the process has been started.
- **Priority**: Value which has bee defined at process start or during run-time via appropriate task function for the process.
- **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.
- **Folder**: A link to the tab *Documents* of a process is displayed in report result.

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.

- **Comments:** Step comment of an activity instance which has been created in **@enterprise** versions less than 9.0 (e.g. created at change agent). Step comments in **@enterprise** 9.0 are stored in notes which can be displayed in report result by adding an attribute/condition of the entity *Note*. The JOIN selection offers the possibility to make a join between activity instance and note.

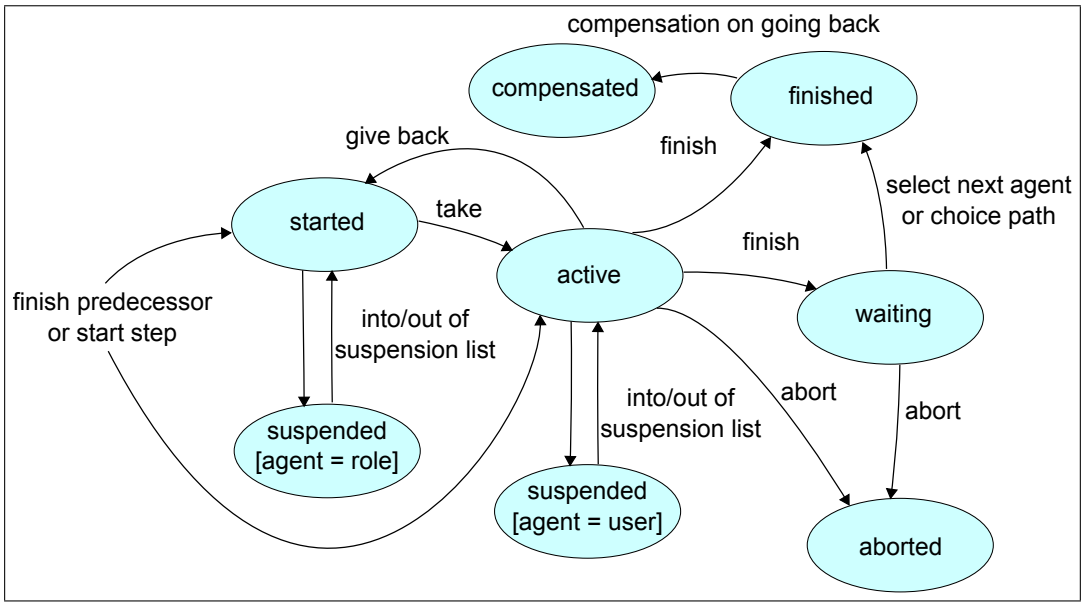

Figure 2.2: **Activityinstance states**

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).
- **Subject**: Subject of the process.
- **Priority**: Priority of the process.
- **Description**: Process description.
- **Active**: Activity-status of the process (active, inactive).
- **Object Id**: The id of the current object (in this case process) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

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.
- **Object Id**: The id of the current object (in this case application) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

2.1.5 Display Fields of Process Relation

- **Process 1** and **Process 2**: Process-Id of the process which is in relation to the searched process.
- **Process Relation Type**: Type of the process relation which can be defined in **@enterprise** administration under Configuration/Search/Process relations.

2.1.6 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 or a custom date format pattern which needs to fit the JAVA SimpleDateFormat patterns (see section 2.1.23): Format for Timestamps.

Example:
Proz-ID =1, execution order: order → order → a_task → 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

2.1.7 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 of report until *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.

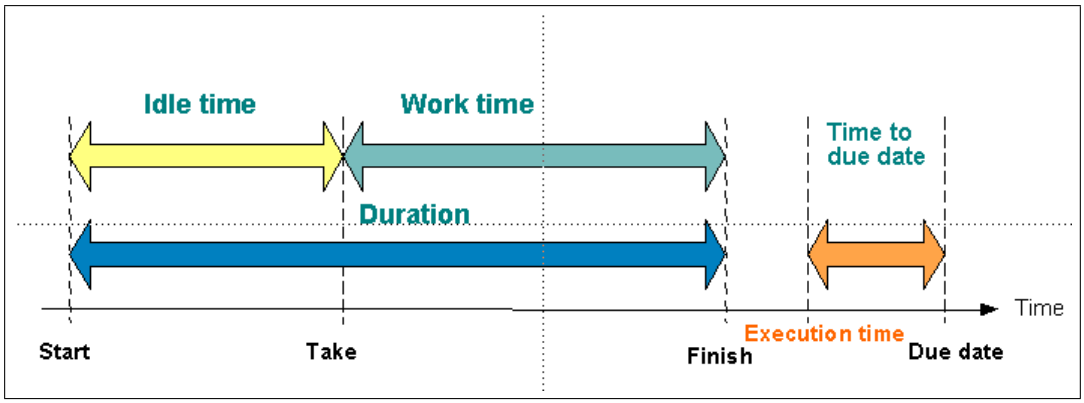

Figure 2.3: **Time intervals for tasks**

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.

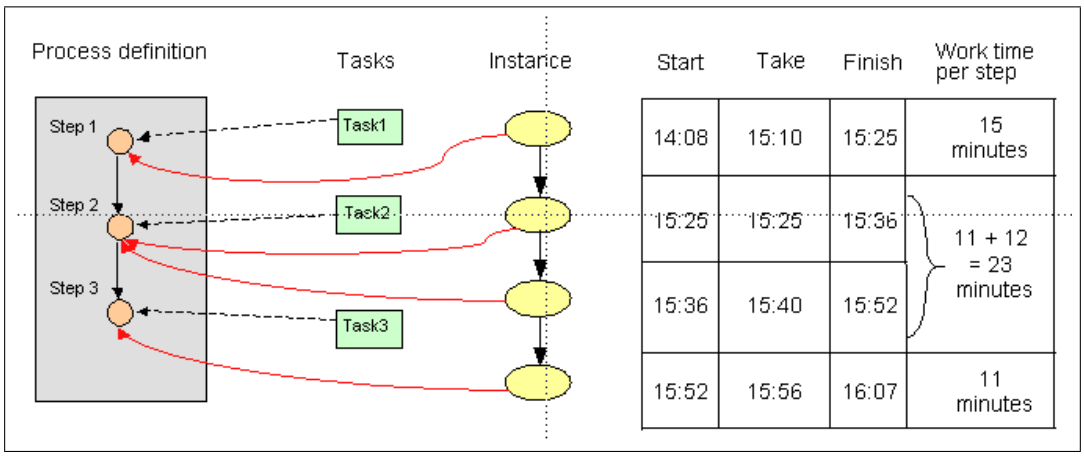

Figure 2.4: **Computation of time intervals per step**

2.1.8 Display fields of Tasks

- **Id**: Task-Id.

- **Name**: Name of the task.
- **Version**: Version of the task.
- **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).
- **Costs**: The costs of the task. This field is not used from **@enterprise**.
- **Effort**: The effort of the task in minutes. This field is not used from **@enterprise**.
- **Object Id**: The id of the current object (in this case task) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

2.1.9 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.
- **Object Id**: The id of the current object (in this case OU) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

2.1.10 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.
- **Object Id**: The id of the current object (in this case role) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

2.1.11 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.

- **Object Id**: The id of the current object (in this case user) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

2.1.12 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.

2.1.13 Display fields of Documents

- **Name**: Name 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.
- **Form type**: Form type of the document (see chapter *Forms* in administration guide).

2.1.14 Display fields of Documentfolder

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

2.1.15 Display fields of Link

- **Name**: Name of link.
- **Linked object**: Name of linked object.
- **Created at**: Timestamp of creation of this link.
- **Created by**: User who created this link.
- **Last changed at**: Timestamp when link was changed last time.
- **Last changed by**: User who changed this link.
- **Object Id**: The id of the current object (in this case Link) can be displayed in report result. This attribute can be used for e.g. the drilldown functionality of Reporting component.

2.1.16 Display fields of Keyword

- **Keyword**: Keyword label.

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 Display fields of user session

- **User**: The first and last name of the user.
- **IP address**: The IP-adress of the user.
- **Date of initialization**: The initialization-date of the user-session (login date of the user).
- **Date of logout**: The date when the user was logged-of from **@enterprise**.
- **Last access**: The date when the user was active in the system.

2.1.19 Display fields of Recycle Bin content

The **@enterprise** recycle bin is a temporary storage location for deleted DMSobjects which is displayed per user and is active by default. DMSobjects are moved to recycle bin by selecting them in DMSObjectTable and activating toolbar function *Delete* or pressing Del on keyboard.

- **Origin**: Information of the origin storage location of the DMSobject (DMSfolder, process instance).
- **Deleted at**: The timestamp when DMSobject has been deleted/moved to recycle bin.
- **Deleted with**: Information, if the element has been deleted directly or indirectly. Indirect means that a folder has been deleted and the element is part of this folder

2.1.20 Form fields as display fields

For adding form fields click on a form in the list *Tables*, select a formfield of the list *Attributes* and activate the button "Add". 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).

2.1.21 User defined display fields

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

- **Scheme:** schema of a database

- **Table:** name of database table
- **Table alias:** a table alias for this query
- **Attribute:** name of field of table
- **Column title:** the column header, can include I18N keys (e.g: @@@key@@)
- **Aggregation:** maximum, minimum, count, average, sum, Aggregation in expression
- **Type:** Data type (e.g. String, Date, etc.)
- **Expression:** SQL-expression, used as selection to the table
- **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.

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

Table: avw_user
Table alias: u
Attribute: surname

After activating the button *Ok* the attribute will be added. You can change the column title to *Surname* by clicking on text *avw_user.surname*. After this action following settings must be done in section *Extended Options* of the attribute:

Type: String
Expression: u.id like 'user

Example 2: Create a list of tasks with id and oid:
Attribute 1:
Table: avw_task
Table alias: t
Attribute: oid

After activating the button *Ok* the attribute will be added. You can change the column title to *oid* by clicking on text *avw_task.oid*. After this action following settings must be done in section *Extended Options* of the attribute:
Type: String

Attribute 2:
Table: avw_task
Table alias: t
Attribute: id

After activating the button *Ok* the attribute will be added. You can change the column title to *id* by clicking on text *avw_task.id*. After this action following settings must be done in

section *Extended Options* of the attribute:
Type: String

2.1.22 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*).

2.1.23 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. Furthermore it is possible to enter a custom date format which needs to fit the JAVA SimpleDateFormat patterns. If such custom date formats should be available by default (selectable), the schema file *reporting.xml* must be overwritten and the mapping *dateformats* extended like in following example:

```
<!--  extend this mapping to add dateformats -->
 <mapping xmlid="dateformats">
  <mapentry key="yyyy.MM.dd" value="My date format" />
 </mapping>
```

The entry "My date format" should be available beneath the entry "Year" after reloading all reporting schema files.

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):

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

2.1.24 Options for fields

For each attribute (display field) a set of options can be defined:

- Infos: Shows the database table and field of selected attribute.
- Column title: The column header, can include I18N keys (e.g: @@@key@@).
- Date format: Is displayed only, if type *Date* is selected and allows the definition of hour, day, week, month, quarter, year or a custom date format pattern which needs to fit the JAVA SimpleDateFormat patterns (see section 2.1.23): grouped by time spans
- Aggregation: Select an aggregation as described in 2.1.22.
- 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.
- 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)*).
- Table alias: 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 2.3. Once the attribute is added to the report, the alias can not be modified in edit mode to ensure that selected joins are correct.
- 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.

After adding an attribute by using button *Add*, another options are available:

- Sorting: By activating the icon on the left side of attribute block the sorting direction can be changed. An upward arrow indicates ascending, downward arrow indicates descending order of the attribute in result table. If no arrow is displayed, no sort direction for this attribute is used. 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.
- Grouping column and -key: Grouping of columns enables reporting to do a second level aggregation, which groups all display fields. Any aggregation-function of the specific column data type is usable as 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.
- Order of attributes: The order of attributes can be changed by activating an attribute block and moving it to another location or by using the arrow up/down function in the toolbar.

- Remove: By using the remove-function (displayed as red X in toolbar) the attribute can be removed from report.

Fig. 2.5 shows how this options can be 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*.

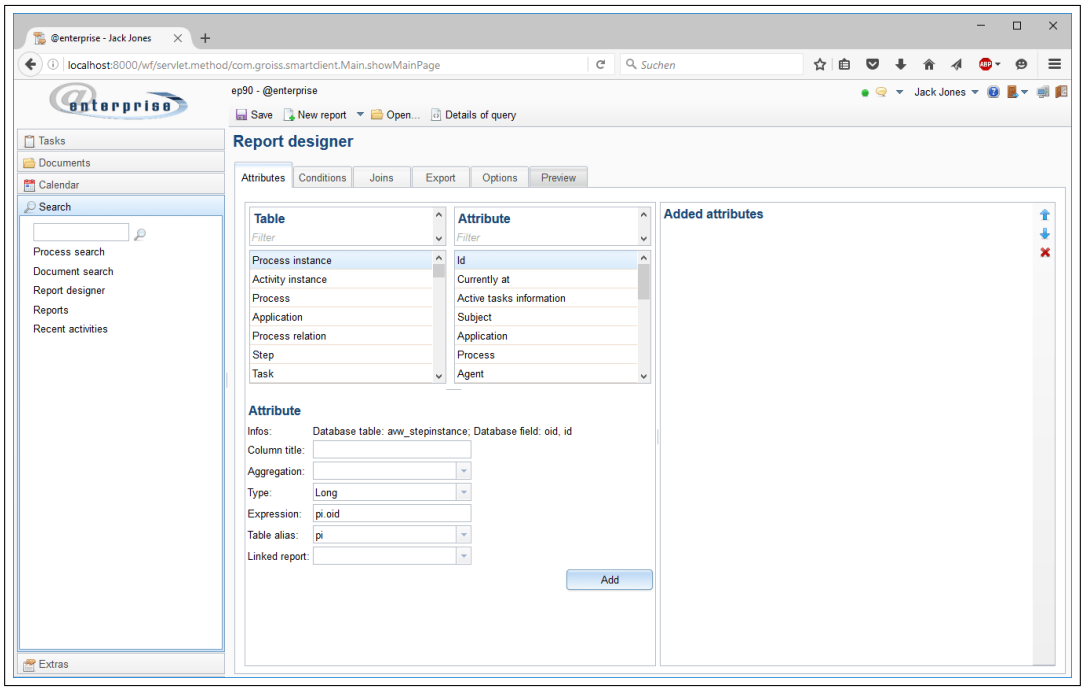

Figure 2.5: **Selecting OID of ProcessInstance via Extended Options**

2.2 Query conditions

With conditions you can reduce the amount of lines in the result set. Click the tab "Conditions" 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. 2.6. 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 close the object selection, click on *Close*.

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.

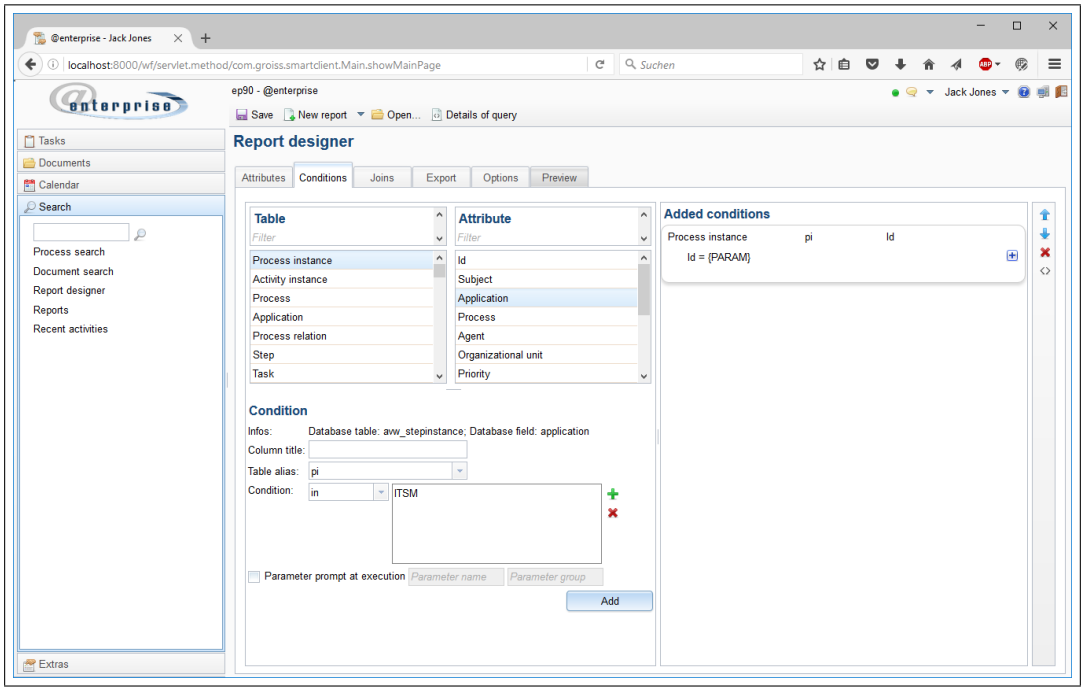

Figure 2.6: **Condition for objects**

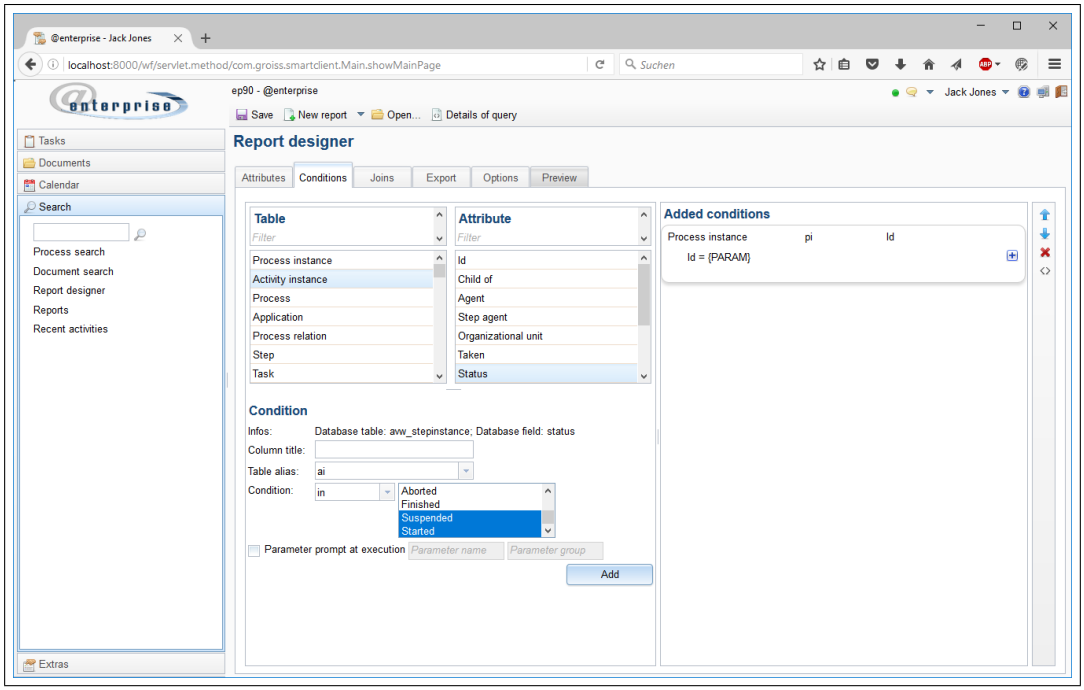

Figure 2.7: **Conditions for states**

States (Processinstance state, Activityinstance state) Select a item from the lists *Status* (see Fig. 2.7).

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

- Specify a date, where the attribute value is greater (after) or equal to this date,
- 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. 2.9 show some examples of date conditions.

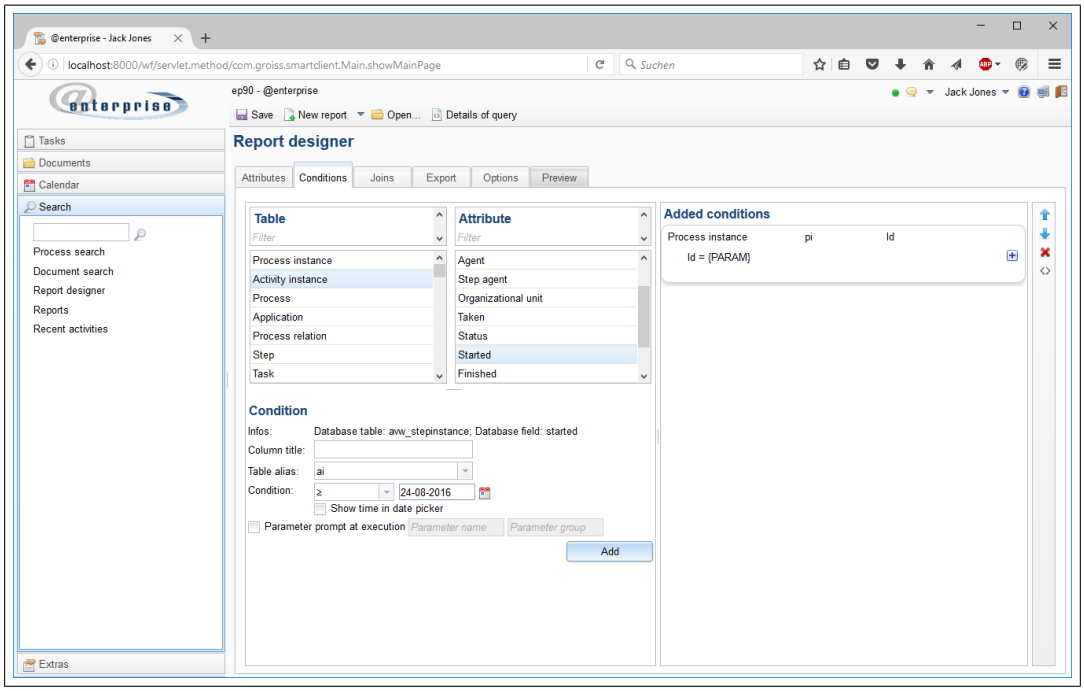

Figure 2.8: **Conditions for date fields**

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 icon *Form* of an attribute, 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 tab *Conditions*.
Otherwise you can specify a formfield condition like any other condition. see Fig.

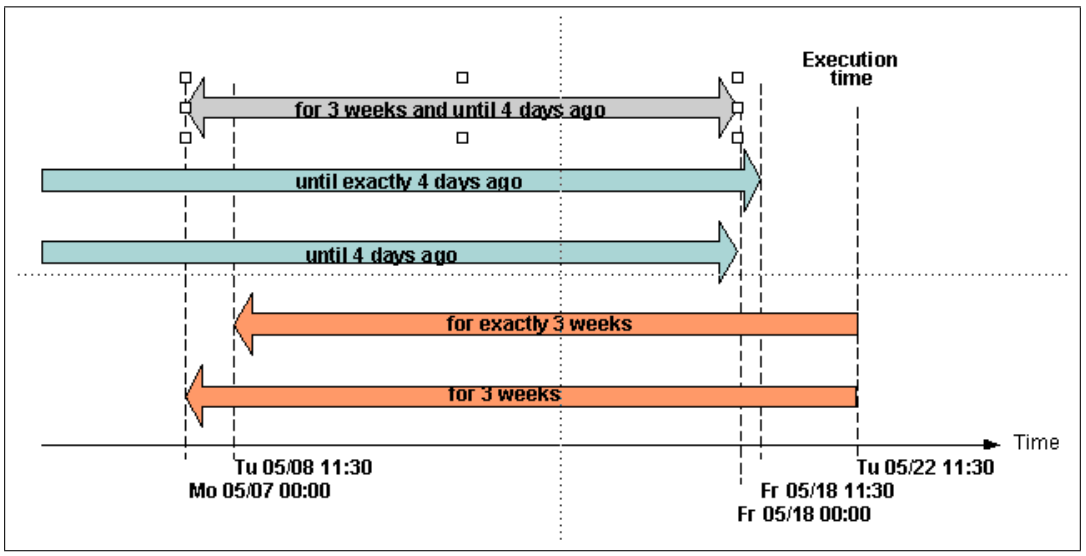

Figure 2.9: **Date condition examples**

2.11.

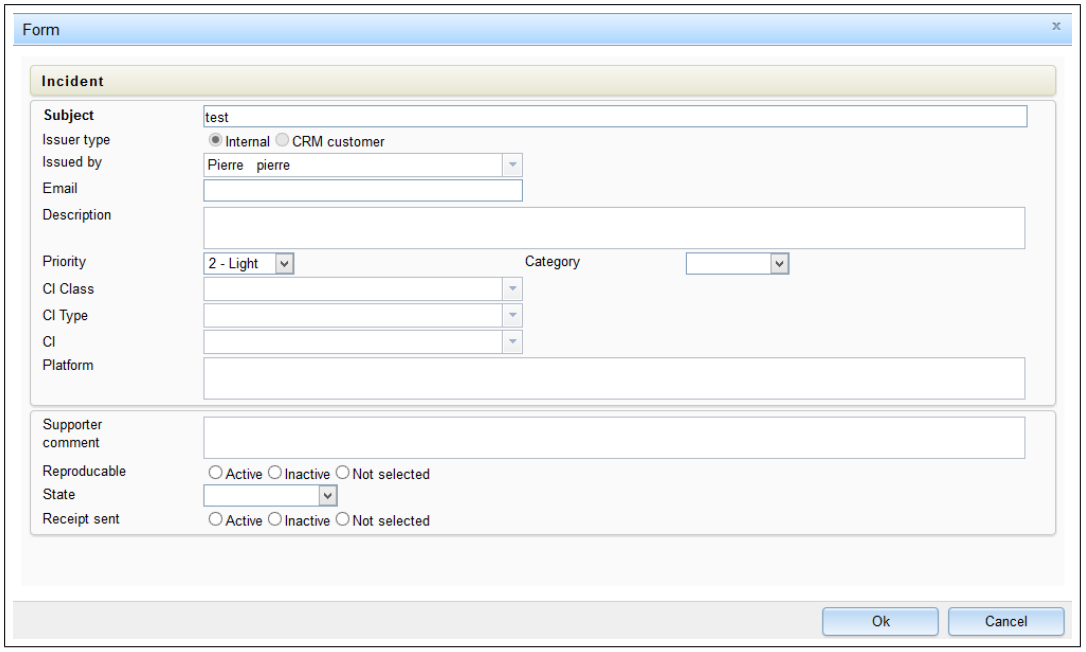

Figure 2.10: **Set condition by entering values in the formfields**

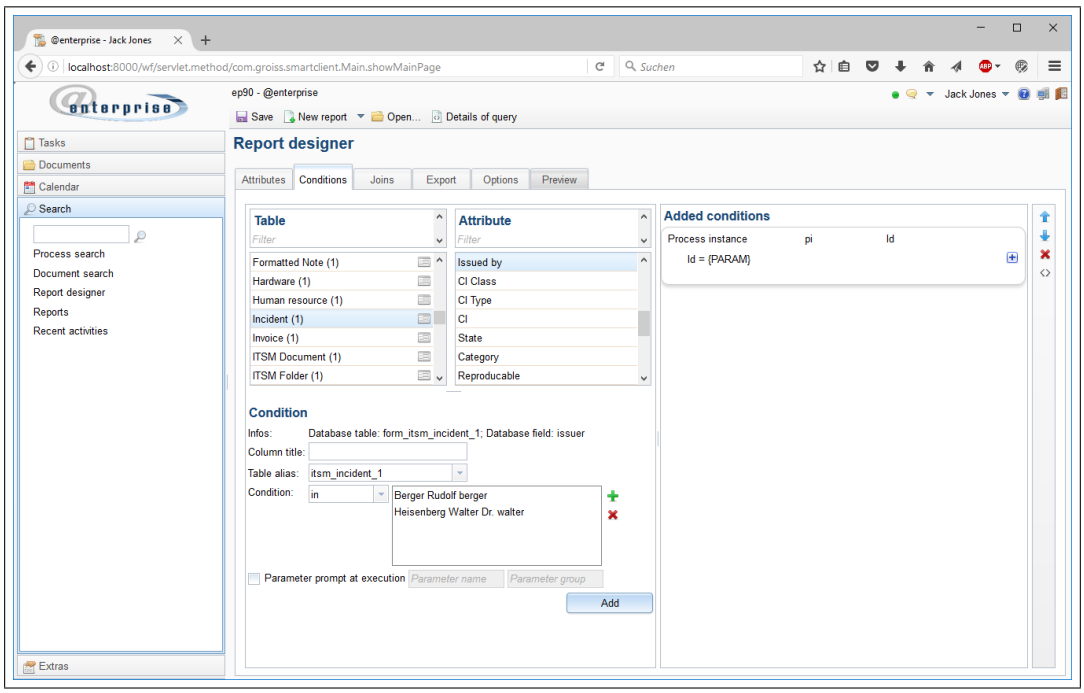

Figure 2.11: **Set formfield condition by entering values in condition form**

2.2.1 Connection of conditions

All conditions you add can be connected with "AND" or "OR" (AND is "stronger" than OR). Furthermore, you can set parentheses by selecting one or more conditions and click on the corresponding toolbar function *(..) Set Parentheses*.

2.2.2 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. 2.12).
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. The label (= condition text) is created automatically by report designer with placeholder {PARAM}. This label is displayed on dialog *Result details* (see section 4) and the placeholder is replaced.
Attention: If the label has been changed manually, the placeholder is substituted by the value only. It is assumed that the operator is part of the label!

Another feature is the possibility to group parameters by entering an arbitrary text into field *Parameter group* (also possible to internationalize text by entering @@@). Parameters with the same parameter group text are displayed on param mask in the same group.
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 executes the report.
The following implicit parameter conditions are possible:

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

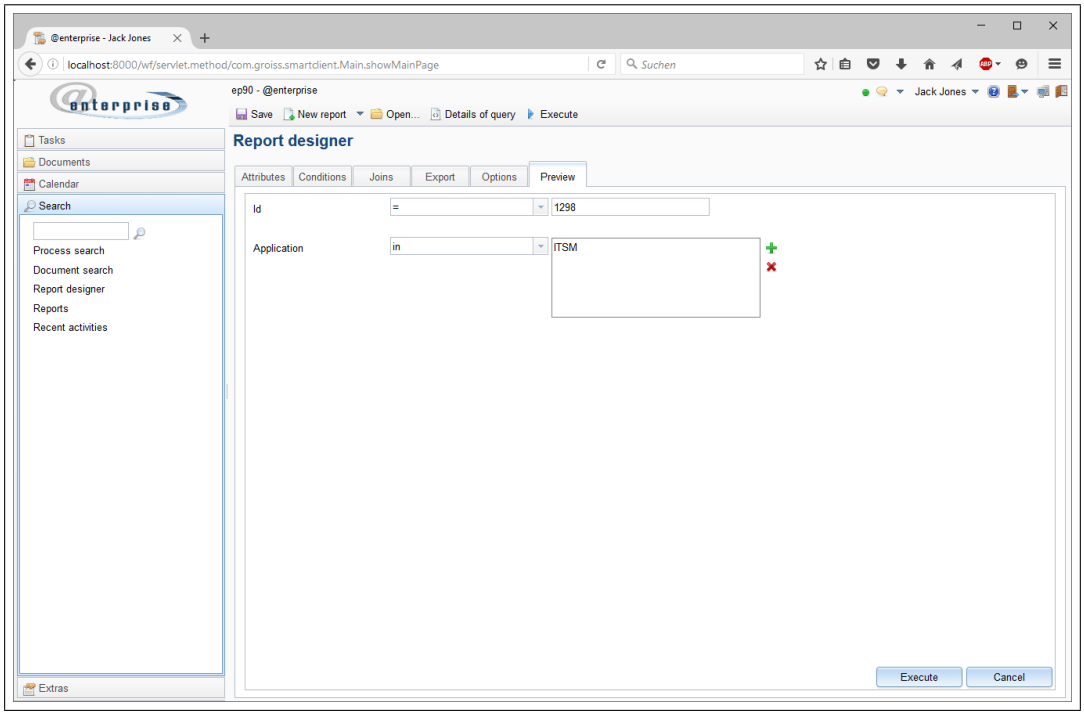

Figure 2.12: **Setting the parameters**

2.2.3 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.

2.3 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. In tab *Joins* the existing joins can be adapted.

These possible joinways are predefined in schema. Fig. 2.13 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 *Ok*. The popup will close and the attribute/condition will be added to the report.

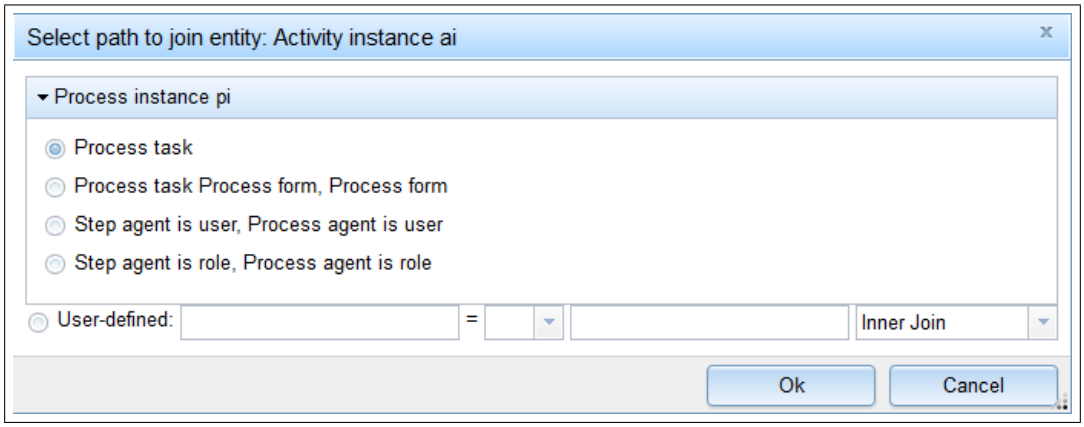

Figure 2.13: **Selecting Joinpath for Processinstance and Activityinstance**

Fig. 2.14 shows how join selection works if there are more than one entity already 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*) the triangle icon is located. Click on this icon to see the possible paths, choose the path and click on *Ok*.

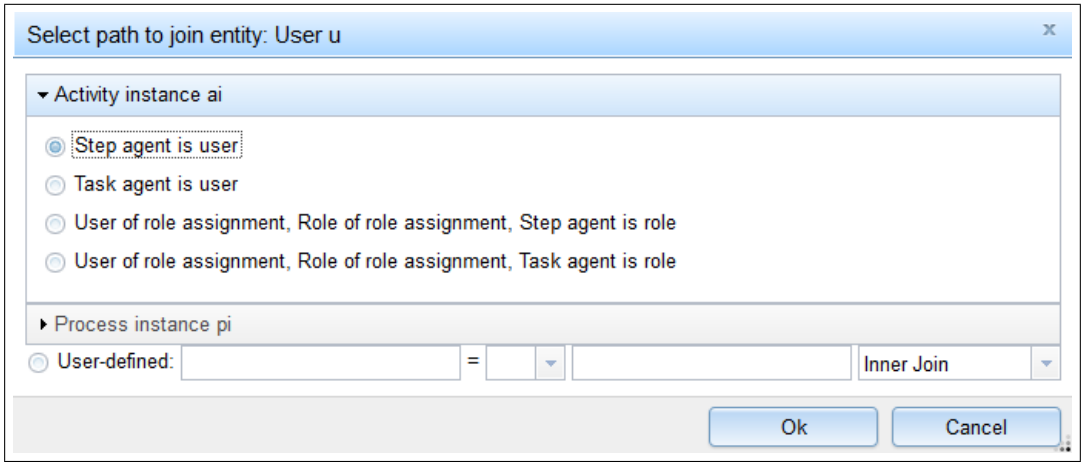

Figure 2.14: **Joining User either to ProcessInstance or ActivityInstance**

If the predefined joins do not include the needed join, user-defined joins can be declared. Fig. 2.15 shows how 2 entities (processinstance pi, processinstance pi2) can be joined via

id to get all subprocesses.

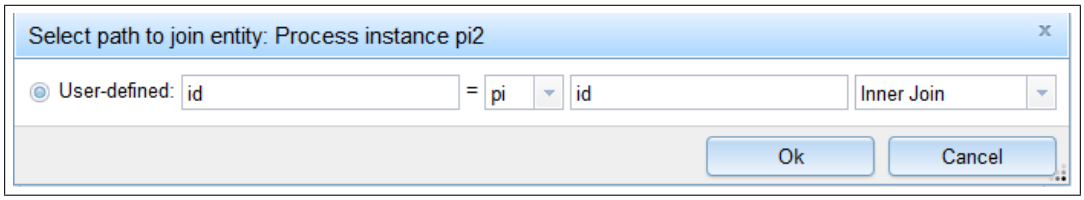

Figure 2.15: **User defined Joins**

2.4 Export of query results

In this tab it is also possible to define the exporter for query results. You can select between following exporters:

- **HTML Table**: The result is shown in the browser (default). The fields *Clickable value*, *Type of selection* and *Hide clickable value in result table* are relevant, if toolbar functions are defined for the report result (see section 2.5.11). It is also possible to define a fix width of the result table (in pixel) and the widths of the columns (in percent depending on result table size).
- **Chart**: See chapter 2.4.1
- **Export to Excel**: The query result will be stored in a XLS-file, which can be downloaded. If checkbox *Hide info* is activated, the query result details (e.g. condition details, execution time, etc.) are not written into file.
- **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@*. The checkbox *Use UTF-16 Little Endian Encoding* can be used, if result contains special characters (e.g. umlauts) to get correct representation in file. If checkbox *Hide info* is activated, the query result details (e.g. condition details, execution time, etc.) are not written into file.
- **XML export**: Similar to *Export to Excel*, whereas you can determine a style sheet. The target file is a XML-file. If checkbox *Hide info* is activated, the query result details (e.g. condition details, execution time, etc.) are not written into file.
- **PDF-Exporter**: 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. 2.16). 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. If checkbox *Hide info* is activated, the query result details (e.g. condition details, execution time, etc.) are not written into file.

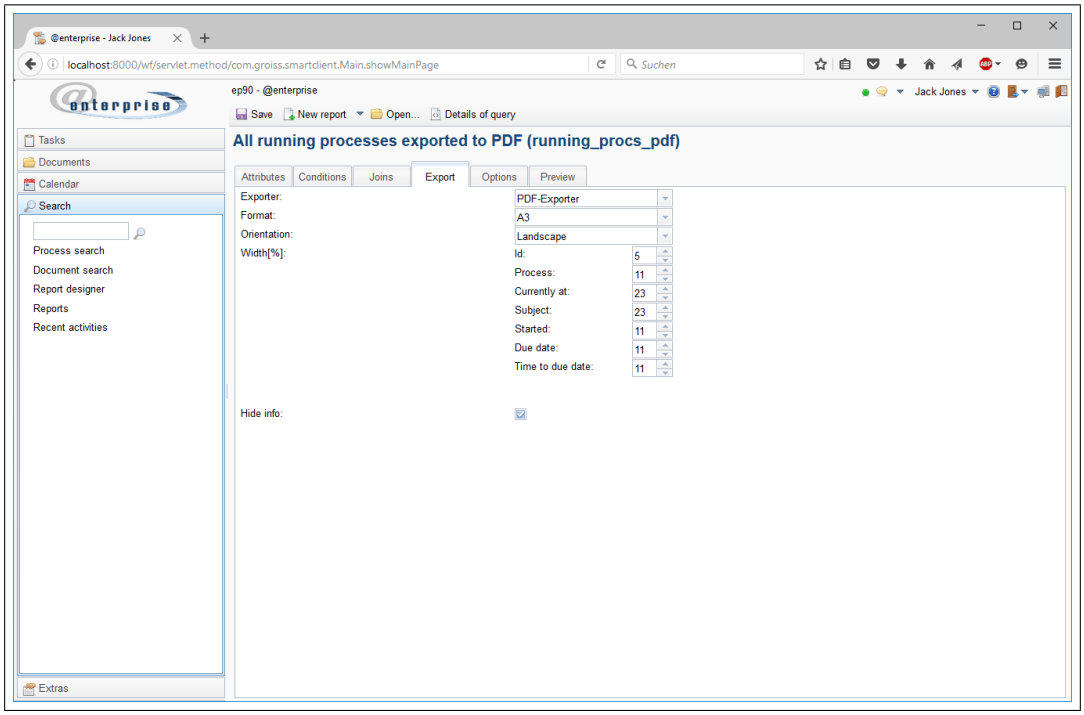

Figure 2.16: **Options of PDF-Exporter**

- **Open-Office Export:** This exporter allows to export in diverse output formats by defining an Open-Office template (ODT template). The output formates are PDF, ODT, DOC and DOCX. Please note that DOCX (Office Open XML) is useable with LibreOffice only! The template must be placed in the classpath of **@enterprise** and consists of placeholders in XPath syntax (see *Application Development Guide* - section *Office Templates*). In the following an example of an ODT template is shown:

```
${REPEAT $row in $data}
 ${com.groiss.reporting.export.OpenOfficeExporter.
   getRowData($query,$row,0)}
 ${com.groiss.reporting.export.OpenOfficeExporter.
   getRowData($query,$row,1)}
${END}
```

  The method *com.groiss.reporting.export.OpenOfficeExporter.getRowData* allows to read one row of the whole result table. The first parameter $query defines that the report result should be read. The second parameter $row indicates that a row should be read and the third parameter indicates which cell of the row should be read. If checkbox *Hide info* is activated, the query result details (e.g. condition details, execution time, etc.) are not written into file.

- **Escalation Exporter:** With this exporter escalations can be fired for processes of the reporting result. For this purpose you have to select a *Process Id Attribute* which

identifies a process (should be process instance id). Furthermore an *Action* can be defined which is executed, if escalation is fired:

- Send an email: Sends an email to current agents or a self-defined list of email recipients
- Call method: The entered Java method is executed

The report must be stored first before this exporter can be used! Escalations are fired one time for each process of reporting result. If escalation has been fired correctly for a process, this one will be skipped in future.

- **SVG chart:** Analog to option *Chart* (see also section 2.4.1). In addition to ordinary charts it is possible to define a drill-down report (linked report): If a display attribute is linked to another report, the data value of of the clicked display value 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.

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

2.4.1 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. 2.18). 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 exporter you can determine the charttype (see Fig. 2.17). 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 chart types:

- 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.

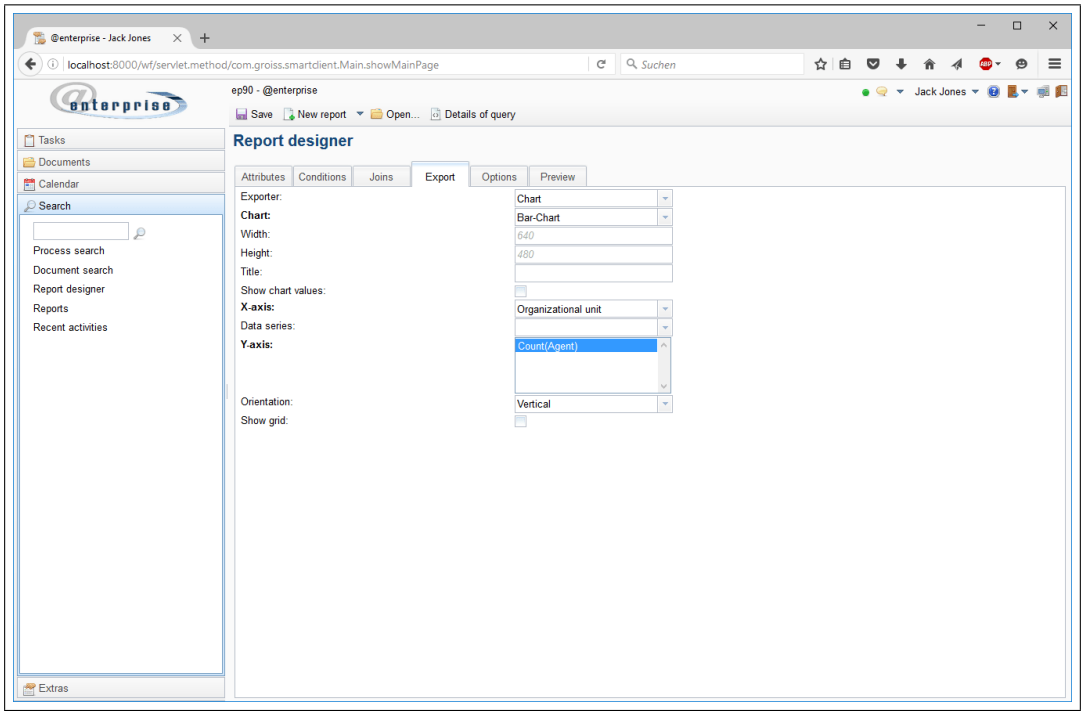

Figure 2.17: **Diagram settings**

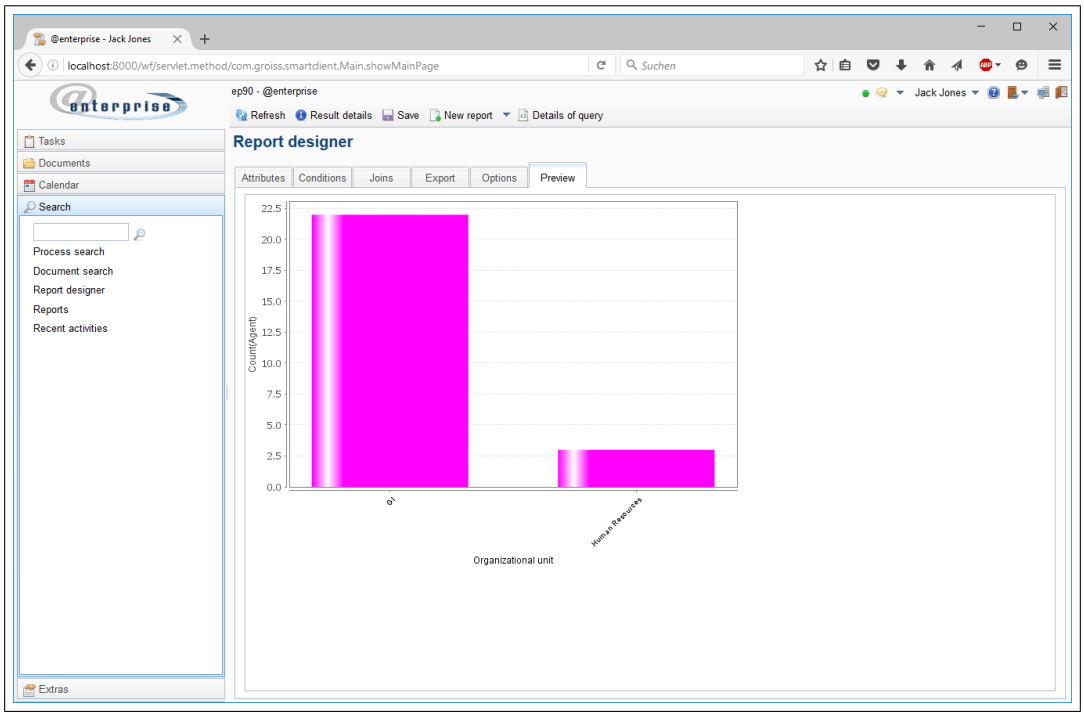

Figure 2.18: **Example diagram**

2.5 Report options

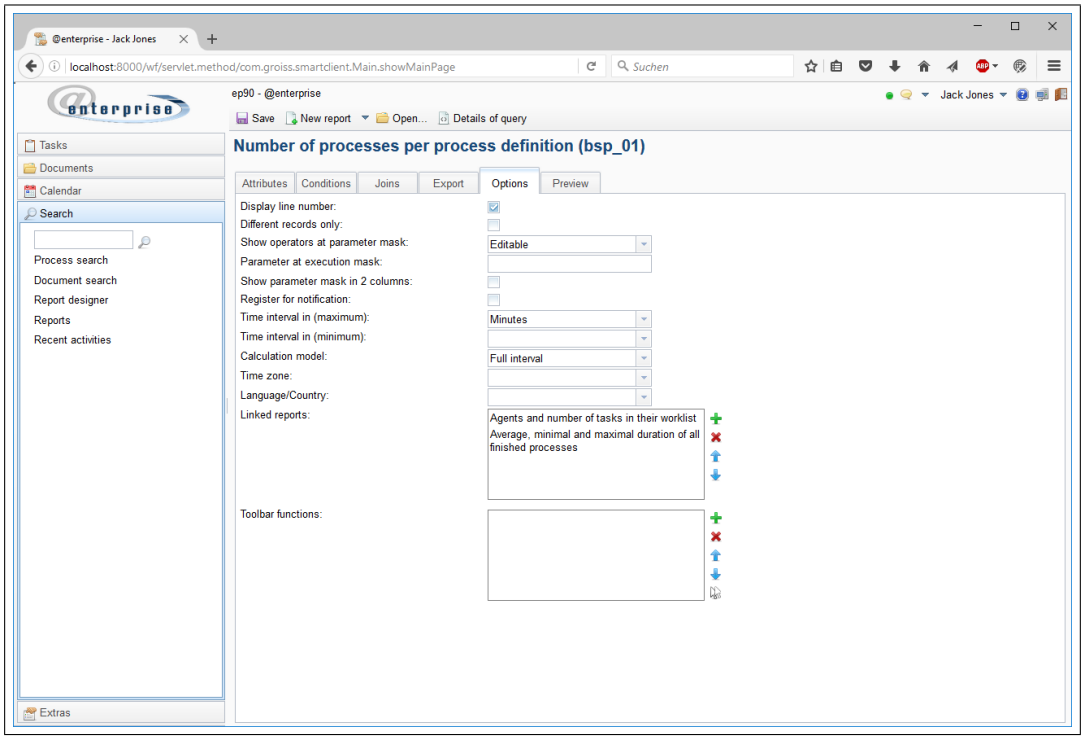

Figure 2.19: **Report options**

2.5.1 Checkbox *Display line number*

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

2.5.2 Checkbox *Different records only*

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

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

2.5.3 Show operators at parameter mask

Default values for operator and value of parameter at execution conditions can be chosen since **@enterprise** 8.0. Depending on the selection of following value the representation of operators can be manipulated:

- Editable: Operators are editable.
- Disable: Operators are not editable, but are displayed.

- Hidden: Operators are hidden and therefor not editable.

More information about param mask is available in section 2.2.2.

2.5.4 Parameter at execution mask

This field allows to define an own parameter mask. This mask can be either a HTML file (must be valid XHTML), which is interpreted as DOJO widget (see example below) or a DOJO widget itself (*.js file). This field must be stored in classpath under `alllangs/scripts`, e.g. if file is stored under `classes/alllangs/scripts/ep/widget/MyParamMask.html`, then the value of this field must be `ep/widget/MyParamMask.html`.

All input fields on parameter mask require the attribute `data-dojo-attach-point`. If a parameter name has been entered for the condition (can be done via GUI and is recommended), the value of data-dojo-attach-point must be defined under this directive: `paramname_value` If no parameter name has been entered, the automatic assigned number (always starting with 0 for first condition) must be used under this directive: `value<NR>`

Example for parameter mask as HTML file:

```
<div>
 <h1>Customized param mask</h1>
 <table>
  <tr>
   <td>Process-ID: </td>
   <td><input data-dojo-type="dijit/form/ValidationTextBox"
        data-dojo-attach-point="value0"></input>
   </td>
  </tr>
  <tr>
   <td>Priority: </td>
   <td><input data-dojo-type="dijit/form/ValidationTextBox"
        data-dojo-attach-point="c1_value"></input>
   </td>
  </tr>
 </table>
</div>
```

The parameter mask is interpreted as DOJO widget, therefor no further HTML tags (e.g. html, body, etc.) are needed! In the example above the text `value` + the number of the condition (in this case the number of first condition in XML) is used as *data-dojo-attach-point* for *Process-ID*; the condition *Priority* uses the entered parameter name + `_value`.

Hint: The options *Show operators at parameter mask* and *Show parameter mask in 2 columns* have no effect for the own defined parameter mask!

2.5.5 Checkbox *Show parameter mask in 2 columns*

If this checkox is activated, the defined conditions are displayed in 2 columns on param mask.

More information about param mask is available in section 2.2.2.

2.5.6 Checkbox *Register for notification*

This checkbox defines, if the report should be refreshed by a notification. The notification is done with method `QueryEngine.sendReportRefreshToClient(String reportId)` - for more details see **@enterprise** APIDoc.

2.5.7 Time interval

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

2.5.8 Computing of time intervals (Calculation model)

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.

2.5.9 Timezone and Language/Country

To execute a report in a specific timezone and/or Language/Country, you can specify it here. If nothing is chosen, the timezone and locale of the executing user is taken.

2.5.10 Linked reports

Any stored report can be linked to a report. HTML links to execute the reports are displayed at the bottom of the result page.

2.5.11 Toolbar functions

Equivalent to tables in GUI configuration (see *Administration Guide*), toolbar functions can be configured. These functions can be declared as task functions in administration or as xml node in a config-file. The toolbar function can use the result data, if in tab *Export*

1. the exporter *HTML Table* is selected,
2. a column in field *Clickable value* is defined which is used as referenced column for the function and
3. the *Type of selection* is selected depending on the expectation of the function (e.g. *Set read/unread* expects only one selected entry).

One toolbar function may be marked as double-click action only which fires when the user double-clicks on the result line.

Usage of standard @enterprise functions

@enterprise offers by default functions which could be used in Reporting, but it is not guaranteed that the reporting result will be refreshed after executing the function! Following table contains some standard functions and which referenced column is needed:

Function	*Referenced column*
Set read/unread	Activity instance/Id
Set due date	Process instance/Id (only process due date can be set), Activity instance/Id (process and task due)
Into clipboard	Process instance/Id, Activity instance/Id
Copy to ...	Activity instance/Id
Add parfor steps	Activity instance/Id
Set priority	Process instance/Id, Activity instance/Id
Add process relation	Process instance/Id, Activity instance/Id
Follow the process	Process instance/Id, Activity instance/Id

Table 2.1: **@enterprise Functions as toolbar functions**

Following action id's can be used for entries in worklist:

Action id	*Name*	*Referenced column*
finish	Complete	Activity instance/Id
finishAndSelect	Complete and assign	Activity instance/Id
goBack	Go back	Activity instance/Id
seeLater	Suspend	Activity instance/Id
makeVersion	Make version	Activity instance/Id
setAgent	Reassign	Activity instance/Id
cut	Cut	Activity instance/Id

Table 2.2: **@enterprise Action id's as toolbar functions**

2.6 SQL report

The function *SQL report* is a special form of the Report Designer and allows the creation of reports with the aid of SQL statements. The display attributes must be added separately, the FROM-block allows the definition of an arbitrary SQL statement.
Display attributes can be added/delete with the appropriate toolbar functions and moved in their positions via drag & drop.
The display attribute are consists of following elements:

- SQL attribute: The column name of a table.
- Label: The column label (header) which is displayed in report result.

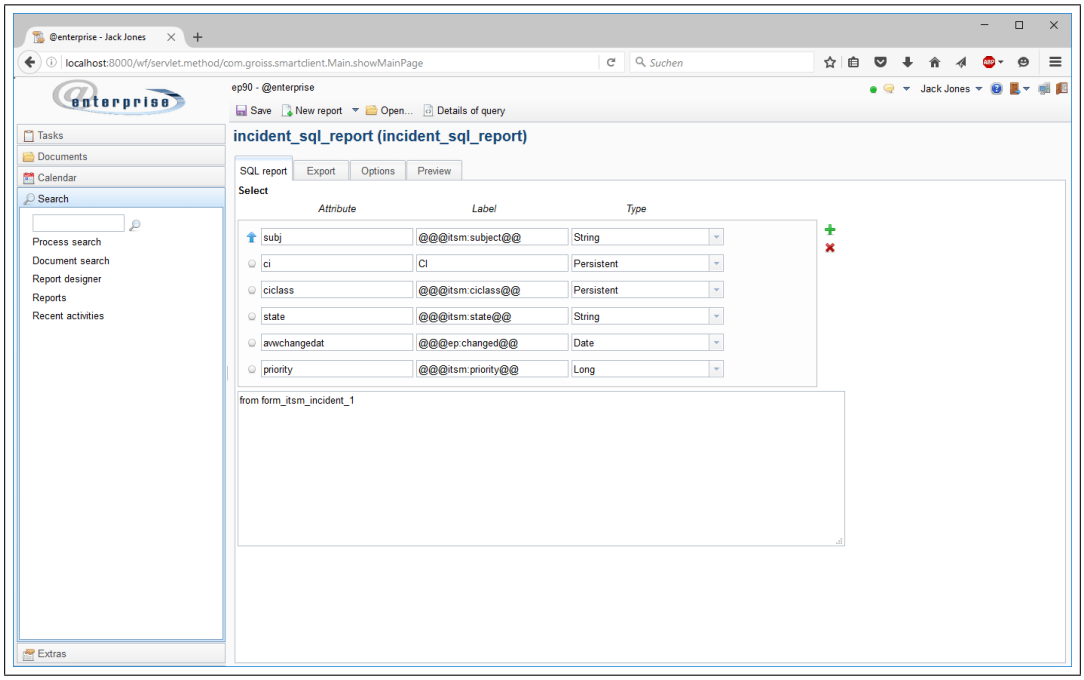

Figure 2.20: **SQL report**

- Data type: Selection of diverse data types:
  - *String*
    Description: any string
    SQL attribute: any varchar field of a database table
  - *Double*
    Description: rational number
    SQL attribute: any decimal field of a database table
  - *Long*
    Description: natural number
    SQL attribute: decimal field (without decimal places)
  - *Date*
    Description: date object
    SQL attribute: date field in database table
  - *TimeInterval*
    Description: time interval
    SQL attribute: 2 comma separated timestamps
  - *Persistent*
    Description: reference to another database table
    SQL attribute: referenced oid field, the corresponding _class field is added automatically
  - *Implementation of Persistent*
    Description: reference to another database table

SQL attribute: referenced oid field

- *com.groiss.reporting.data.impl.ProcessLink*
  Description: Link to process history, the id is used as link label
  SQL attribute: OID and ID (comma separated) of a process instance
- *Implementation of com.groiss.reporting.data.ReportingData*
  Description: formatted due to the implementation
  SQL attribute: expected attributes due to implementation

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 (Aggregation; 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.
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".

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

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

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 button *User-defined* in tab *Conditions* and select the entry *SQL command*. After activating the button *Ok* the *Extended Options* of this condition can be opened where field *SQL command* can be filled.

4 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:

- **Save**: This function allows you to save your report. Enter an *Id*, a *Name* and select an *Application* which the report should belongs to and activate the button *Save* or *Save and close* (see Fig. 4.1). If you have already opened a stored report and activate this function, you can store this report with the same name (=*Update*) or delete it by activating the button *Delete*.
  If the appropriate rights has been assigned to current user, beside the field *Name* the I18n-link is displayed with the translations of this key. The translations can be edited directly (e.g. in case of admin session) and stored with tooltip dialog button *Save* or viewed only. The changes are stored in the resource file of the given application.

- **New report**: With this function you can create a new search. All fields will be cleared. It is also possible to select an exisiting query and press the *OK* button to edit this query.

- **Open...**: By activating this function a dialog will be opened where a stored report can be selected and loaded into Report Designer.

- **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.

- **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.

- **Result details**: This function is available in the result-mask only and shows details of the query result.

- **Edit**: This function is available in search result only. With this function you can edit your build query.

- **Print...:** This function is available for HTML tables only and allows to print the displayed result.
- **Export Options**: See chapters 2.4.1 and 2.4.

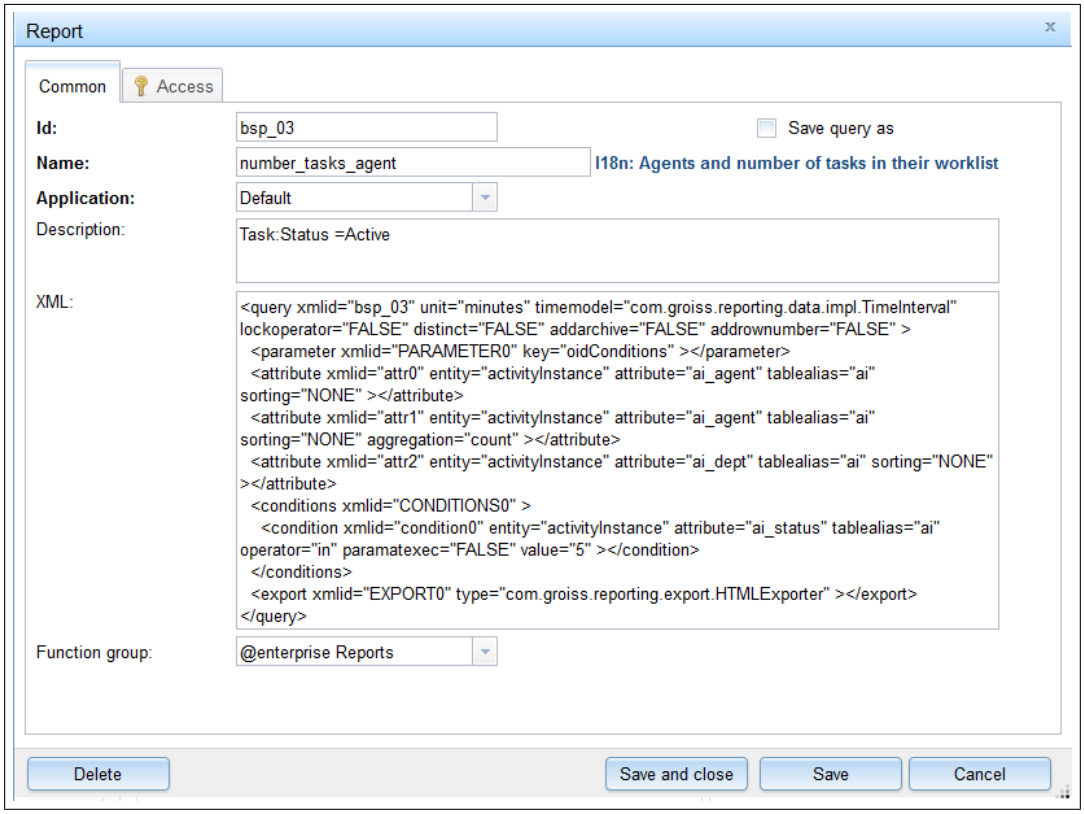

Figure 4.1: **Save query**

5 Configuration of the reporting component

5.1 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 report, go the the list of users or roles, click on permissions and add a new permission: Select the right "execute", the object class "Reports" and select the query you want. To enable user to set permissions to stored reports, the user needs the right "edit permissions".
Note that in the list of stored reports 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.

5.2 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>/<contextRoot>/servlet.method/`
`com.groiss.reporting.gui2.PublicReportingGui.listQueries`

Add the parameter `groupId=<idOfFunctionGroup>` or `group=<nameOfFunctionGroup>` to get a list of only one functiongroup.

5.3 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*.

5.4 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 6.

5.5 Reporting-Cache

Changes in Persistent objects will not effect reporting results, if the result was found in reporting cache. In this situation re-execute the report result with the toolbar function *Refresh* or re-initialize the reporting engine (Administration -> Admin Tasks -> Server Control)

6 Developers Guide

The following chapter shall give an overview about the possibilities to customize the reporting component.

6.1 Hidden Configuration

Reporting uses 2 hidden parameters 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.

6.2 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.

6.2.1 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>
```

6.2.2 Query

Queries are defined in XML documents, too. This XML tree is generated while configuring the query options at the Report designer 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 2.4 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>
```

6.3 API

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

6.3.1 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" ;
}
```

6.3.2 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. Due to new GUI implementation of **@enterprise** 9.0 any implementation has to implement toJson which may return a JSON Object or a String and getColumnRenderer which refers to a dojo widget implementing a ColumnRenderer. The result of toJson is passed to the ColumnRenderer on the client, so any formatting has to be done there.

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

Example will be given in Chapter 6.3.3.

6.3.3 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 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);
   public void addClientConditionWidgetOptions(
               JSONObject json) throws JSONException;
   public List<Pair<String,String>> getOperatorList(
               boolean comesFromParamMask);
}
```

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.Query.addSelect* and *com.groiss.reporting.Query.addSelectIndexOfAttrib(Element, int)* 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.Query.addCondition*.

setValue gains the data from ResultSet and stores it.

addClientConditionWidgetOptions adds condition widget information to attribute store. The implementing widget is stored as *condwidget* attribute to the json-object. Any additional Data will be passed to the widget at instantiation.

getOperatorList returns a pairlist of selectable operators. The key is the xml representation of the operator, the value the I18N string of the operator.

Example: com.groiss.demo.ProgressReportingData 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 "demo_order_1":

```
<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>
```

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, which is handeld by the implementing dojo widget . Cause of extending DefaultReportingData we do not need to implement every method.
You will find a Demoimplementation in the demo package of **@enterprise**.
(see: *com.groiss.demo.reporting.ProgressReportingData*)

6.3.4 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);
}
```

6.3.5 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 JSONArray getExportOptionsJSON() throws JSONException;
   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.

getExportOptionsJSON enables exporter to add option fields to the export option page. Return a jsonarray which contains jsonobjects which have at leaast an id, a label and an implementing widget.

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*)

6.3.6 ClientSideExporter

Due to the new GUI of **@enterprise** 9.0 a subinterface of ReportingExporter has been introduced. Implement this interface if you need to overwrite the exporting functions of a report.

```
public interface ClientSideExporter extends ReportingExporter{
   public String getClientSideRenderer();
   public List<Map<String,Object>> toJson(
    Query q, ReportingTableModel tm) throws JSONException;
   public JSONArray getResultDetailsJson(
    Query q, int count) throws JSONException;
}
```

getClientSideRenderer Returns the path to a renderer widget, which has to display the result.

toJson Transfers the Tablemodel data to a JSON Object which will be passed to the ClientSideRenderer

getResultDetailsJson Returns a JSON Array which holds all informations displayed as report details. Every label and value pair is handeld as a Json Array.

6.4 Implementing your own Search Mask

To implement your own search mask, just design a form with the needed input fields. When submitting the search, instantiate an *ep/widget/smartclient/reporting/ReportingResult* object and put it in the designated target. The ReportingResult object executes the report the passed report (passed in paramter *query.qxml* or *query.id* and adds the form data given in paramter *postParams* to the sql condition. The naming and handling of parameters is described in the follwoing section.
The recommended way is to build a stored report in Report designer, which includes *parameter-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*. Since **@enterprise** 9.0 you can specifiy the parameter name in the reporting mask of conditions too. Add a field *comesFromParamMask* with value "1" to the search mask, so that engine expects the parameter in the post parameters.

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 6.1: 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 6.2: 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 6.3: 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 *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 6.4: 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 6.5: 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 6.6: 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 6.7: 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 6.8: 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 6.2.1

Table 6.9: Description of element join

7 Support

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