Wikiprint Book

Title: Trac Reports

Subject: YAM - Yet Another Mailer - TracReports

Version: 9

Date: 06/25/2016 05:40:06 AM

Table of Contents

Trac Reports **3**
Changing Sort Order 3
Changing Report Numbering 3
Navigating Tickets 3
Alternative Download Formats 4
Comma-delimited - CSV (Comma Separated Values) 4
Tab-delimited 4
RSS - XML Content Syndication 4
Creating Custom Reports 4
Ticket columns 4
Advanced Reports: Dynamic Variables 5
Using Variables in a Query 5
Special/Constant Variables 5
Advanced Reports: Custom Formatting 5
Special Columns 5
Automatically formatted columns 5
Custom formatting columns 6
Changing layout of report rows 6
Reporting on custom fields 6
A note about SQL rewriting 7

Trac Reports

The Trac reports module provides a simple, yet powerful reporting facility to present information about tickets in the Trac database.

Rather than have its own report definition format, TracReports relies on standard SQL `SELECT` statements for custom report definition.

> **Note:** The report module is being phased out in its current form because it seriously limits the ability of the Trac team to make adjustments to the underlying database schema. We believe that the query module is a good replacement that provides more flexibility and better usability. While there are certain reports that cannot yet be handled by the query module, we intend to further enhance it so that at some point the reports module can be completely removed. This also means that there will be no major enhancements to the report module anymore.
>
> You can already completely replace the reports module by the query module simply by disabling the former in trac.ini:
>
> ```
> [components]
> trac.ticket.report.* = disabled
> ```
>
> This will make the query module the default handler for the ?View Tickets? navigation item. We encourage you to try this configuration and report back what kind of features of reports you are missing, if any.

A report consists of these basic parts:

- **ID** ? Unique (sequential) identifier
- **Title** ? Descriptive title
- **Description** ? A brief description of the report, in WikiFormatting text.
- **Report Body** ? List of results from report query, formatted according to the methods described below.
- **Footer** ? Links to alternative download formats for this report.

Changing Sort Order

Simple reports - ungrouped reports to be specific - can be changed to be sorted by any column simply by clicking the column header.

If a column header is a hyperlink (red), click the column you would like to sort by. Clicking the same header again reverses the order.

Changing Report Numbering

There may be instances where you need to change the ID of the report, perhaps to organize the reports better. At present this requires changes to the trac database. The *report* table has the following schema:

- id integer PRIMARY KEY
- author text
- title text
- query text
- description text

Changing the ID changes the shown order and number in the *Available Reports* list and the report's perma-link. This is done by running something like:

```
UPDATE report SET id = 5 WHERE id = 3;
```

Keep in mind that the integrity has to be maintained, ie ID has to be unique, and you don't want to exceed the max, since that's managed by SQLite someplace.

You may also need to update or remove the report number stored in the report or query.

Navigating Tickets

Clicking on one of the report results will take you to that ticket. You can navigate through the results by clicking the *Next Ticket* or *Previous Ticket* links just below the main menu bar, or click the *Back to Report* link to return to the report page.

You can safely edit any of the tickets and continue to navigate through the results using the *Next/Previous/Back to Report* links after saving your results, but when you return to the report, there will be no hint about what has changed, as would happen if you were navigating a list of tickets obtained from a query (see TracQuery#NavigatingTickets).

Alternative Download Formats

Aside from the default HTML view, reports can also be exported in a number of alternative formats. At the bottom of the report page, you will find a list of available data formats. Click the desired link to download the alternative report format.

Comma-delimited - CSV (Comma Separated Values)

Export the report as plain text, each row on its own line, columns separated by a single comma (','). **Note:** The output is fully escaped so carriage returns, line feeds, and commas will be preserved in the output.

Tab-delimited

Like above, but uses tabs (\t) instead of comma.

RSS - XML Content Syndication

All reports support syndication using XML/RSS 2.0. To subscribe to an RSS feed, click the orange 'XML' icon at the bottom of the page. See TracRss for general information on RSS support in Trac.

Creating Custom Reports

Creating a custom report requires a comfortable knowledge of SQL.

Note that you need to set up permissions in order to see the buttons for adding or editing reports.

A report is basically a single named SQL query, executed and presented by Trac. Reports can be viewed and created from a custom SQL expression directly in the web interface.

Typically, a report consists of a SELECT-expression from the 'ticket' table, using the available columns and sorting the way you want it.

Ticket columns

The *ticket* table has the following columns:

- id
- type
- time
- changetime
- component
- severity
- priority
- owner
- reporter
- cc
- version
- milestone
- status
- resolution
- summary
- description
- keywords

See TracTickets for a detailed description of the column fields.

Example: **All active tickets, sorted by priority and time**

```
SELECT id AS ticket, status, severity, priority, owner, time AS created, summary
FROM ticket
WHERE status IN ('new', 'assigned', 'reopened')
```

```
ORDER BY priority, time
```

Advanced Reports: Dynamic Variables

For more flexible reports, Trac supports the use of *dynamic variables* in report SQL statements. In short, dynamic variables are *special* strings that are replaced by custom data before query execution.

Using Variables in a Query

The syntax for dynamic variables is simple, any upper case word beginning with '$' is considered a variable.

Example:

```
SELECT id AS ticket,summary FROM ticket WHERE priority=$PRIORITY
```

To assign a value to $PRIORITY when viewing the report, you must define it as an argument in the report URL, leaving out the leading '$':

```
http://trac.edgewall.org/reports/14?PRIORITY=high
```

To use multiple variables, separate them with an '&':

```
http://trac.edgewall.org/reports/14?PRIORITY=high&SEVERITY=critical
```

Special/Constant Variables

There is one dynamic variable whose value is set automatically (the URL does not have to be changed) to allow practical reports.

- $USER ? Username of logged in user.

Example: List all tickets assigned to me:

```
SELECT id AS ticket,summary FROM ticket WHERE owner=$USER
```

Advanced Reports: Custom Formatting

Trac is also capable of more advanced reports, including custom layouts, result grouping and user-defined CSS styles. To create such reports, we will use specialized SQL statements to control the output of the Trac report engine.

Special Columns

To format reports, TracReports looks for 'magic' column names in the query result. These 'magic' names are processed and affect the layout and style of the final report.

Automatically formatted columns

- **ticket** ? Ticket ID number. Becomes a hyperlink to that ticket.
- **id** ? same as **ticket** above when **realm** is not set

  realm ? together with **id**, can be used to create links to other resources than tickets (e.g. a realm of *wiki* and an *id* to a page name will create a link to that wiki page)
  - for some kind of resources, it may be necessary to specify their *parent* resources (e.g. for *changeset*, which *repos*) and this can be achieved using the **parent_realm** and **parent_id** columns
- **created, modified, date, time** ? Format cell as a date and/or time.
- **description** ? Ticket description field, parsed through the wiki engine.

Example:

```
SELECT id AS ticket, created, status, summary FROM ticket
```

Those columns can also be defined but marked as hidden, see below.

See ?trac:wiki/CookBook/Configuration/Reports for some example of creating reports for realms other than *ticket*.

Custom formatting columns

Columns whose names begin and end with 2 underscores (Example: **__color__**) are assumed to be *formatting hints*, affecting the appearance of the row.

- **__group__** ? Group results based on values in this column. Each group will have its own header and table.
- **__grouplink__** ? Make the header of each group a link to the specified URL. The URL is taken from the first row of each group.
- **__color__** ? Should be a numeric value ranging from 1 to 5 to select a pre-defined row color. Typically used to color rows by issue priority.

  Defaults: Color 1 Color 2 Color 3 Color 4 Color 5
- **__style__** ? A custom CSS style expression to use on the `<tr>` element of the current row.
- **__class__** ? Zero or more space-separated CSS class names to be set on the `<tr>` element of the current row. These classes are added to the class name derived from `__color__` and the odd / even indicator.

Example: *List active tickets, grouped by milestone, group header linked to milestone page, colored by priority*

```
SELECT p.value AS __color__,
    t.milestone AS __group__,
    '../milestone/' || t.milestone AS __grouplink__,
    (CASE owner WHEN 'daniel' THEN 'font-weight: bold; background: red;' ELSE '' END) AS __style__,
    t.id AS ticket, summary
FROM ticket t,enum p
WHERE t.status IN ('new', 'assigned', 'reopened')
 AND p.name = t.priority AND p.type = 'priority'
ORDER BY t.milestone, p.value, t.severity, t.time
```

Note: A table join is used to match *ticket* priorities with their numeric representation from the *enum* table.

Changing layout of report rows

By default, all columns on each row are display on a single row in the HTML report, possibly formatted according to the descriptions above. However, it is also possible to create multi-line report entries.

- **column_** ? *Break row after this*. By appending an underscore ('_') to the column name, the remaining columns will be continued on a second line.
- **_column_** ? *Full row*. By adding an underscore ('_') both at the beginning and the end of a column name, the data will be shown on a separate row.
- **_column** ? *Hide data*. Prepending an underscore ('_') to a column name instructs Trac to hide the contents from the HTML output. This is useful for information to be visible only if downloaded in other formats (like CSV or RSS/XML). This can be used to hide any kind of column, even important ones required for identifying the resource, e.g. `id as _id` will hide the **Id** column but the link to the ticket will be present.

Example: *List active tickets, grouped by milestone, colored by priority, with description and multi-line layout*

```
SELECT p.value AS __color__,
      t.milestone AS __group__,
      (CASE owner
         WHEN 'daniel' THEN 'font-weight: bold; background: red;'
         ELSE '' END) AS __style__,
      t.id AS ticket, summary AS summary_,             -- ## Break line here
      component,version, severity, milestone, status, owner,
      time AS created, changetime AS modified,         -- ## Dates are formatted
      description AS _description_,                    -- ## Uses a full row
      changetime AS _changetime, reporter AS _reporter -- ## Hidden from HTML output
FROM ticket t,enum p
WHERE t.status IN ('new', 'assigned', 'reopened')
 AND p.name = t.priority AND p.type = 'priority'
ORDER BY t.milestone, p.value, t.severity, t.time
```

Reporting on custom fields

If you have added custom fields to your tickets (see TracTicketsCustomFields), you can write a SQL query to cover them. You'll need to make a join on the ticket_custom table, but this isn't especially easy.

If you have tickets in the database *before* you declare the extra fields in trac.ini, there will be no associated data in the ticket_custom table. To get around this, use SQL's "LEFT OUTER JOIN" clauses. See ?TracIniReportCustomFieldSample for some examples.

A note about SQL rewriting

Beyond the relatively trivial replacement of dynamic variables, the SQL query is also altered in order to support two features of the reports:

i. changing the sort order
ii. pagination support (limitation of the number of result rows displayed on each page)

In order to support the first feature, the sort column is inserted in the `ORDER BY` clause in the first position or in the second position if a `__group__` column is specified (an `ORDER BY` clause is created if needed). In order to support pagination, a `LIMIT ... OFFSET ...` clause is appended. The query might be too complex for the automatic rewrite to work correctly, resulting in an erroneous query. In this case you still have the possibility to control exactly how the rewrite is done by manually inserting the following tokens:

- `@SORT_COLUMN@`, the place where the name of the selected sort column will be inserted,
- `@LIMIT_OFFSET@`, the place where the pagination support clause will be added

Note that if you write them after an SQL comment, `--`, you'll effectively disable rewriting if this is what you want!

Let's take an example, consider the following SQL query:

```
-- ## 4: Assigned, Active Tickets by Owner ## --

--
-- List assigned tickets, group by ticket owner, sorted by priority.
--

SELECT p.value AS __color__,
  owner AS __group__,
  id AS ticket, summary, component, milestone, t.type AS type, severity, time AS created,
  changetime AS _changetime, description AS _description,
  reporter AS _reporter
FROM ticket t,enum p
WHERE status = 'assigned'
 AND p.name = t.priority AND p.type = 'priority'
ORDER BY __group__, p.value, severity, time
```

The automatic rewrite will be the following (4 rows per page, page 2, sorted by `component`):

```
SELECT p.value AS __color__,
  owner AS __group__,
  id AS ticket, summary, component, milestone, t.type AS type, severity, time AS created,
  changetime AS _changetime, description AS _description,
  reporter AS _reporter
FROM ticket t,enum p
WHERE status = 'assigned'
 AND p.name = t.priority AND p.type = 'priority'
ORDER BY __group__ ASC, `component` ASC,  __group__, p.value, severity, time
LIMIT 4 OFFSET 4
```

The equivalent SQL query with the rewrite tokens would have been:

```
SELECT p.value AS __color__,
  owner AS __group__,
  id AS ticket, summary, component, milestone, t.type AS type, severity, time AS created,
  changetime AS _changetime, description AS _description,
  reporter AS _reporter
FROM ticket t,enum p
WHERE status = 'assigned'
 AND p.name = t.priority AND p.type = 'priority'
ORDER BY __group__, @SORT_COLUMN@, p.value, severity, time
@LIMIT_OFFSET@
```

If you want to always sort first by priority and only then by the user selected sort column, simply use the following `ORDER BY` clause:

```
ORDER BY __group__, p.value, @SORT_COLUMN@, severity, time
```

See also: TracTickets, TracQuery, TracGuide, ?Query Language Understood by SQLite