# Custom Ticket Fields

Trac supports adding custom, user-defined fields to the ticket module. Using custom fields, you can add typed, site-specific properties to tickets.

# Configuration

Configuring custom ticket fields is done in the trac.ini file. All field definitions should be under a section named `[ticket-custom]`.

The syntax of each field definition is:

```
FIELD_NAME = TYPE
(FIELD_NAME.OPTION = VALUE)
...
```

The example below should help to explain the syntax.

## Available Field Types and Options

- **text**: A simple (one line) text field.
  - ♦ label: Descriptive label.
  - ♦ value: Default value.
  - ♦ order: Sort order placement. Determines relative placement in forms with respect to other custom fields.
  - ♦ format: One of:
    - ◊ `plain` for plain text
    - ◊ `wiki` to interpret the content as WikiFormatting
    - ◊ `reference` to treat the content as a queryable value (*since 1.0*)
    - ◊ `list` to interpret the content as a list of queryable values, separated by whitespace (*since 1.0*)
- **checkbox**: A boolean value check box.
  - ♦ label: Descriptive label.
  - ♦ value: Default value: 0 or 1.
  - ♦ order: Sort order placement.
- **select**: Drop-down select box. Uses a list of values.
  - ♦ label: Descriptive label.
  - ♦ options: List of values, separated by | (vertical pipe).
  - ♦ value: Default value (one of the values from options).
  - ♦ order: Sort order placement.
- **radio**: Radio buttons. Essentially the same as **select**.
  - ♦ label: Descriptive label.
  - ♦ options: List of values, separated by | (vertical pipe).
  - ♦ value: Default value (one of the values from options).
  - ♦ order: Sort order placement.
- **textarea**: Multi-line text area.
  - ♦ label: Descriptive label.
  - ♦ value: Default text.
  - ♦ cols: Width in columns

- rows: Height in lines.
- order: Sort order placement.
- format: Either `plain` for plain text or `wiki` to interpret the content as WikiFormatting.

Macros will be expanded when rendering `textarea` fields with format `wiki`, but not when rendering `text` fields with format `wiki`.

## Sample Configuration

```
[ticket-custom]

test_one = text
test_one.label = Just a text box

test_two = text
test_two.label = Another text-box
test_two.value = Default [mailto:joe@nospam.com owner]
test_two.format = wiki

test_three = checkbox
test_three.label = Some checkbox
test_three.value = 1

test_four = select
test_four.label = My selectbox
test_four.options = one|two|third option|four
test_four.value = two

test_five = radio
test_five.label = Radio buttons are fun
test_five.options = uno|dos|tres|cuatro|cinco
test_five.value = dos

test_six = textarea
test_six.label = This is a large textarea
test_six.value = Default text
test_six.cols = 60
test_six.rows = 30
```

**Note**: To make entering an option for a `select` type field optional, specify a leading `|` in the `fieldname.options` option.

## Reports Involving Custom Fields

Custom ticket fields are stored in the `ticket_custom` table, not in the `ticket` table. So to display the values from custom fields in a report, you will need a join on the 2 tables. Let's use an example with a custom ticket field called `progress`.

```
SELECT p.value AS __color__,
   id AS ticket, summary, owner, c.value AS progress
  FROM ticket t, enum p, ticket_custom c
  WHERE status IN ('assigned') AND t.id = c.ticket AND c.name = 'progress'
AND p.name = t.priority AND p.type = 'priority'
  ORDER BY p.value
```

**Note**: This will only show tickets that have progress set in them, which is **not the same as showing all tickets**. If you created this custom ticket field *after* you have already created some tickets, they will not have that field defined, and thus they will never show up on this ticket query. If you go back and modify those tickets, the field will be defined, and they will appear in the query. If that is all that is required, you're set.

However, if you want to show all ticket entries (with progress defined and without), you need to use a `JOIN` for every custom field that is in the query:

```
SELECT p.value AS __color__,
   id AS ticket, summary, component, version, milestone, severity,
   (CASE status WHEN 'assigned' THEN owner||' *' ELSE owner END) AS owner,
   time AS created,
   changetime AS _changetime, description AS _description,
   reporter AS _reporter,
   (CASE WHEN c.value = '0' THEN 'None' ELSE c.value END) AS progress
  FROM ticket t
     LEFT OUTER JOIN ticket_custom c ON (t.id = c.ticket AND c.name = 'progress')
     JOIN enum p ON p.name = t.priority AND p.type='priority'
  WHERE status IN ('new', 'assigned', 'reopened')
  ORDER BY p.value, milestone, severity, time
```

Note in particular the `LEFT OUTER JOIN` statement here.

Note that if your config file uses an **uppercase** name:

```
[ticket-custom]

Progress_Type = text
```

you would use **lowercase** in the SQL: `AND c.name = 'progress_type'`.

## Updating the database

As noted above, any tickets created before a custom field has been defined will not have a value for that field. Here is some SQL (tested with SQLite) that you can run directly on the Trac database to set an initial value for custom ticket fields. It inserts the default value of 'None' into a custom field called 'request_source' for all tickets that have no existing value:

```
INSERT INTO ticket_custom
   (ticket, name, value)
   SELECT
      id AS ticket,
      'request_source' AS name,
      'None' AS value
   FROM ticket
   WHERE id NOT IN (
      SELECT ticket FROM ticket_custom
   );
```

If you added multiple custom fields at different points in time, you should be more specific in the subquery on table `ticket` by adding the exact custom field name to the query:

```
INSERT INTO ticket_custom
   (ticket, name, value)
   SELECT
```

```
   id AS ticket,
   'request_source' AS name,
   'None' AS value
FROM ticket
WHERE id NOT IN (
   SELECT ticket FROM ticket_custom WHERE name = 'request_source'
);
```

---

See also: TracTickets, TracIni