Set a field as a function of other fields

This function has been **renamed** with the **JWT 3.0** release.

Find the new documentation at:

Update field based on rules

On this page

Purpose

In versions previous to 2.1.16, this post-function was called "Set a field from a set of rules based on regular expressions".

Post-function **Set a field as a function of other fields** is used for **updating the value of a field depending on the value of other fields** in the issue, by means of a set of setting rules.
Each setting rule is formed by a pair condition-value. There are two types of setting rules:

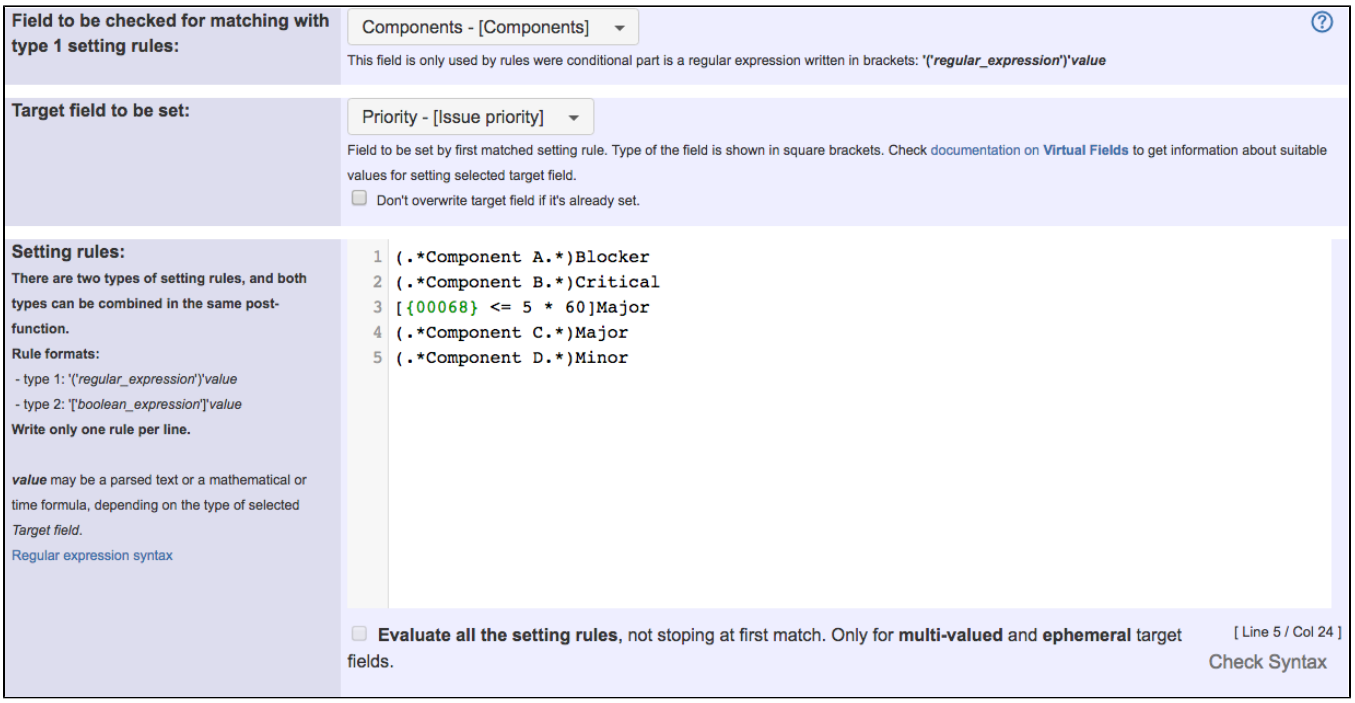
- **type 1**: uses syntax **'('regular_expression')'value** . This kind of setting rule uses a regular expression in the condition part. When the value of field to be checked matches regular expression in a setting rule, the value part of the setting rule will be evaluated and it's value assigned to target field.
- **type 2**: uses syntax **'['boolean_condition']'value** . This kind of setting rule is much more powerful than type 1, since condition part of the rule is a generic **boolean condition**, which can depend on more than one field value, and can contain mathematical, date-time, and text-string terms connected by logical connectives AND, OR and NOT.

Setting rules are evaluated in the same order they are written (one rule per line), and the evaluation stops at first rule matching. The resulting value is copied into target field and the rest of the rules are skipped.

Both kinds of setting rules (type 1 and type 2) can be combined in a same post-function.

Example 1: Set issue priority based on fields Components and Original Estimate

The following screenshot shows an **example of post-function configuration** for setting issue **Priority** depending on fields **Components** and **Original estimate**.
It must be taken into account that multi-select fields (like **Components**) return a list of comma separated values, e.g., Component B, Component D, Component E, and field "**Original estimate (minutes)**" (field code **{00068}**) returns the number of minutes estimated for resolving the issue.
This example combines both types of setting rules:



The behavior implemented by this example is:

- If component Component_A is selected, then priority will be set to Blocker.
- If component Component_B is selected, then priority will be set to Critical.
- If Original estimate is less or equal to 5 hours, then priority will be set to Major.
- If component Component_C is selected, then priority will be set to Major.
- If component Component_D is selected, then priority will be set to Minor.

Example 2: Assign issues at creation to project roles based on priority and number of components

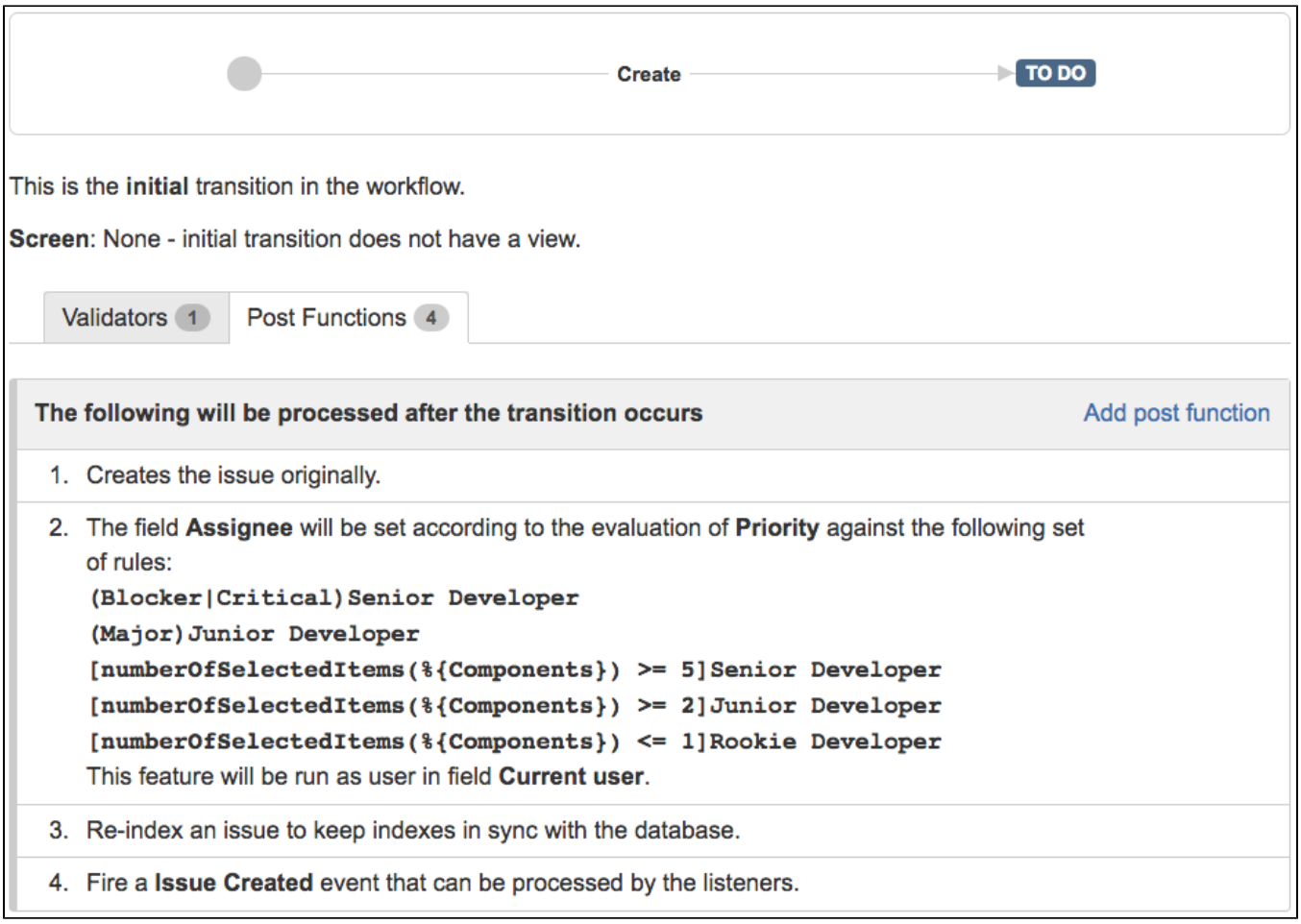
- If Priority is Blocker or Critical, assign issue to project role Senior Developers.
- If Priority is Major, assign issue to project role Junior Developers.
- For the rest of priorities:
    - If number of affected Components is equal or greater than 5, assign issue to project role Senior Developers.
    - If number of affected Components is between 2 and 4, assign issue to project role Junior Developers.
    - If number of affected Components is equal or lower than 1, assign issue to project role Rookie Developers.

Note that:

- **%{00094}** is field code for field virtual field "**Components**"

You have previously set the default user for each project role, the same way as in post-function **Assign to project role**

Once configured, **Create Issue** transition would look like this:

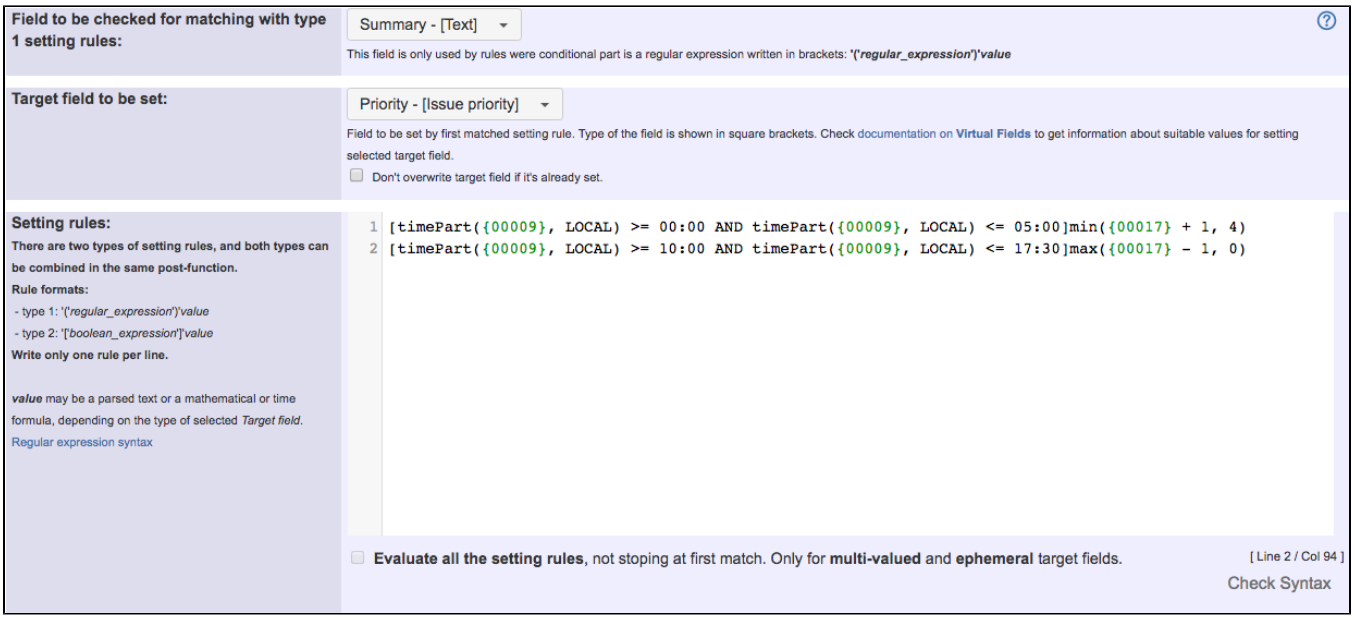


Example 3: Modulate the priority of an issue depending on the time when it is created

We use the following criteria:

- If issue is created from 00:00 to 5:00 we will lower the priority by 1 level.
- if issue is created from 10:00 to 17:30 we will raise the priority by 1 level.

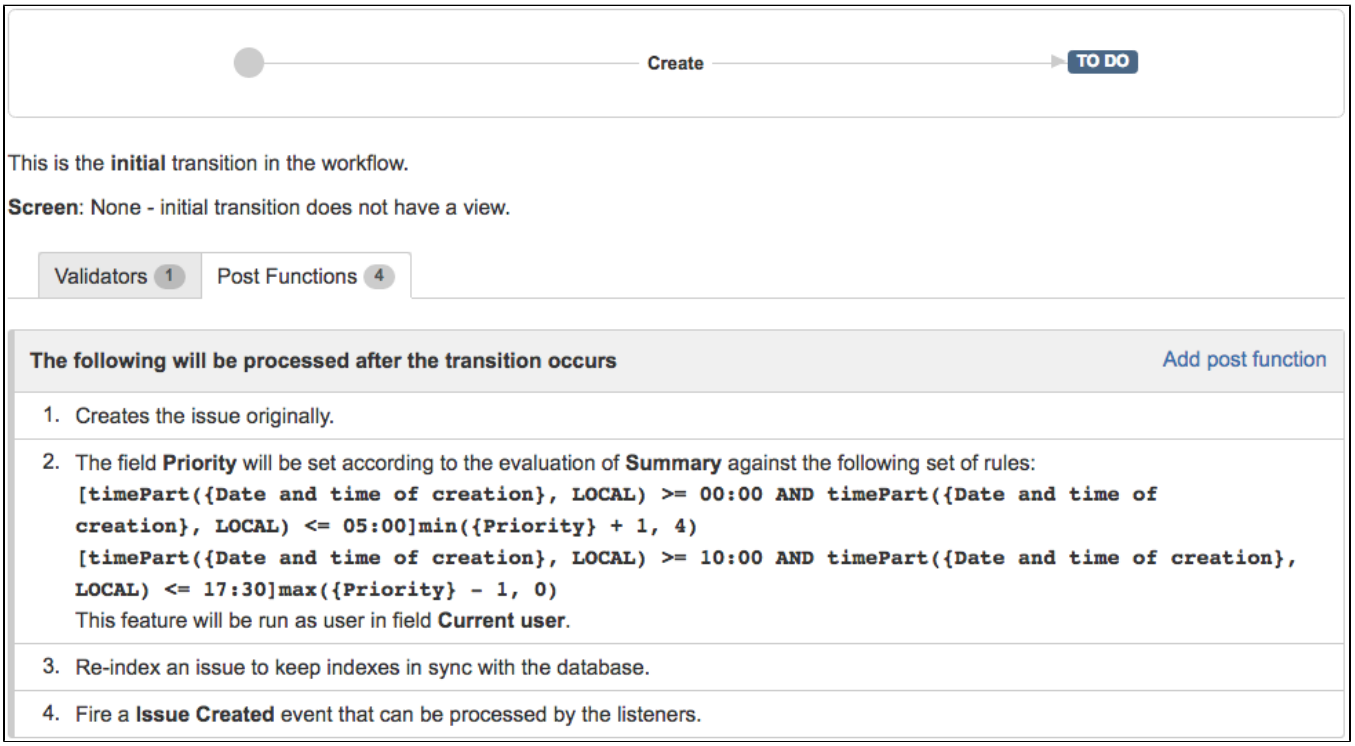
We have 5 priorities, with a numeric value associated to each one of them: **Blocker (0), Critical (1), Major (2), Minor (3) and Trivial (4)**.
In this example parameter field to be checked hasn't any function since we are not using setting rules with regular expressions:



Note that:

- **{00009}** is code for numeric value of "**Date and time of creation**"

Once configured, **Create Issue** transition would look like this:



Configuration Parameters

Field to be checked for matching with type 1 setting rules

Field whose value is going to be evaluated against **regular expressions** in the setting rules. The first matched regular expression will stop the evaluation of the remaining ones, and its associated **setting value** will be written into **Target field**.

Target field to be set

Field that will be set by the post-function. Its type will determine whether the expected Setting value is a **math-time formula** (when Target field is a **Number**, **Date** or **Date-Time**), or a **parsed text** (in the rest of cases).

Setting rules

This parameter is a multiline text field containing **one setting rule per line**. There are 2 types of setting rules, with the following formats:

- **Type 1** setting rules: **optional_prefixes (regexp)value**, where regexp is a **regular expression**, and value is the **setting value** to be copied into target field when Field to be checked matches regexp.
  Field codes with format **%{nnnnn}** (like in post-function **Copy parsed text to a field**) can be inserted in the regular expression. Regular expression will be preprocessed before being used, and inserted field codes will be replaced by the value of the corresponding fields. This way you can create dynamic regular expressions, which depends on the value of fields in the issue.
- **Type 2** setting rules: **optional_prefixes [boolean_expression]value**, where boolean_expression is a **logical expression** containing terms with numeric, text strings and date-time terms. Logical connectives AND, OR and NOT, and parentheses can be used to associate and group terms. This kind of rules are extremely powerful, since you can include more than one field in the boolean expression, so that you can make your rule dependent on various fields in the issue.

Value part of the setting rules: is written just after the closing bracket character (**')'** for type 1 rules, or **']'** for type 2 rules). It represents the **value to be written** into **Target field** when its associated regular expression is matched by the value stored in **field to be checked**, or the boolean expression is evaluated as **true**.
The value part of the setting rules can be a **parsed text** or a **math-time formula** depending on the type of **Target field to be set**:

- **Math or Time formula**: when **Target field to be set** is a field of types **Number**, **Date**, **Date-Time** or **Priority**. The format of these formulas is the same as used in post-function **Mathematical and date-time expression calculator**. Field codes in math-time formulas uses format **{nnnnn}**, unlike format in parsed text **%{nnnnn}**. Obviously you can simply use a **number**, a **date literal** (format YYYY/MM/DD or YYYY-MM-DD) or a **date-time literal** (format YYYY/MM/DD hh:mm or YYYY-MM-DD hh:mm).
- **Parsed Text**: when **Target field to be set** is a field of type different from **Number**, **Date** or **Date-Time**. A string value where you can insert field codes in format **%{nnnnn}**, that will be replaced by its field value.
  Setting rules are **evaluated in order**, and evaluation stops at the first match, and so its important to **order setting rules correctly** if you are writing **not mutually exclusive** regular expressions.

Optional prefixes are single characters that can preceed setting rules for changing someway its behavior:

- **a** : makes value part in setting rules to be parsed in **advanced parsing mode**.
- **i** : in type 1 setting rules, makes regular expression to be evaluated in **ignore case mode**.
- **l** : in type 1 setting rules, makes regular expression to be treated as a **literal string**.

Evaluate all the setting rules (Checkbox)

When this parameter is **unchecked**, once a setting rule is matched, the rest of setting rules are discarded, but when **checked**, all setting rules are evaluated, and this way all value assignments of matching rules are executed. This checkbox is by default unchecked, and only enabled and allowed to be checked, when target field is a multi-valued field, i.e., fields that can contain more than one value: Multi-Select, Checkbox, Multi-User picker, Multi-Group picker, Affected versions, Fixed versions, Components, Labels, Issue picker and Attachments.

Tips and tricks

Don't know much about **regular expressions**?

Even if you have poor knowledge about regular expressions you can find this post-function very useful knowing that **a string composed exclusively by letters, spaces, and numbers is a regular expression that matches itself**, e.g., if you have a custom field of type select list called **Colour** with options *red*, *yellow*, *green* and *blue*, you can evaluate for its selected value using simple setting rules like: `(red)value_for_red, (yellow)value_for_yellow, (green)value_for_green` and `(blue)value_for_blue`.
Example: **Set a custom field "Urgency" depending on a combined value of issue's Priority and "Impact" custom field**.

An 'ELSE' setting rule to manage "the rest of cases"

If none of the setting rules is matched by the value in **Field to be checked** the **Target field** will not be set. If you need to ensure that **Target field** is set in all cases, you should insert as **last setting rule** something like **(.*)setting_value**. This rule works as the **ELSE** clause in a classic IF-THEN-ELSE sentence, since it is evaluated only once all the previous setting rules have been evaluated and not matched, and the regular expression "**(.*)**" matches any string, even an empty one.

Setting fields depending on the value of more than one field

When you need to make the conditional part of a setting rule dependent of more than one field, you should use type 2 setting rules. For example, if you need to "activate" a rule when issue type is **Bug** and Resolution is **Duplicated**, you can use the following setting rule: **`[%{00014}="Bug" and %{00028}="Duplicated"]value_to_be_assigned`** , where **%{00014}** is field code for "**Issue type**" and **%{00028}** is the field code for "**Resolution**".

Combining with other post-functions

What we have described in this page is the basic functionality of the post-funcion, but you can combine it with other post-functions to attain more complex functionalities:

- **Read fields from linked issues or sub-tasks**: you can use this post-function to read fields from linked issues and sub-tasks and store its value in virtual fields **Ephemeral string X** or **Ephemeral number X**. Then using that virtual field in parameter **Field to be checked** you can set fields in current issue depending on the value of fields in linked issues or sub-tasks.
- **Read field from issues returned by JQL query or issue list**: same as **Read fields from linked issues or sub-tasks**, but in this case you can read fields on any issue returned by a JQL query or issue list expression.
- **Write field on linked issues or sub-tasks**: you can select an **Ephemeral** virtual field as **Target Field** and the use this field as source value to write into a field in a linked issued issue or sub-task.
- **Update issue fields**: same as Write field on linked issues or sub-tasks, but in this case you can write into any issue returned by a JQL query or issue list expression.

Usage Examples

Page: Add watcher depending on security level
Page: Add watchers based on issue type
Page: Add watchers depending on the value of a custom field
Page: Assign issue based on the value of a Cascading Select custom field
Page: Assign issue to a specific user based on a specific custom field value
Page: Assign issue to current user if assignee is empty
Page: Assign issue to current user if the user is not member of a certain project role
Page: Change assignee based on a custom field
Page: Change parent's status depending on sub-task's summary
Page: Changing issue priority depending on issue description
Page: Compose dynamic text by inserting field values in a text template
Page: Copy "Due date" into a date type custom field in a linked issue if it's greater than current issue's "Due date"
Page: Limit the number of hours a user can log per day
Page: Make parent issue progress through its workflow
Page: Rise priority if due date is less than 3 weeks away
Page: Set "Due date" depending on the value of other fields, in case it's uninitialized
Page: Set "Due date" to a specific day of next week no matter of date of creation this week
Page: Set "Due date" to current date at issue creation if not initialized
Page: Set a custom field "Urgency" depending on a combined value of issue's priority and "Impact" custom field
Page: Set a date based on current date
Page: Set a field based on reporter's email
Page: Set a watcher at ticket creation depending on custom field's value
Page: Set assignee depending on issue type
Page: Set security level based on groups and project roles the reporter or creator are in
Page: Set security level depending on reporter or creator
Page: Set the assignee based on a condition
Page: Set the value of a field of type "User Picker" depending on other field's value
Page: Set watchers depending on the value of a custom field
Page: Setting a custom field (User Picker) based on the value of another custom field (Text Field)
Page: Setting a field's default value depending on another field
Page: Setting the priority depending on the multiplication of custom fields
Page: Transition an issue automatically depending on the value of a field
Page: Unassign an issue when assigned to project leader
Page: Update checkboxes custom field if a file has been attached during a transition
Page: Using project properties to calculate custom sequence numbers

Related Features