Read field from issues returned by JQL query or issue list

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

Find the new documentation at:

Copy field values from multiple issues

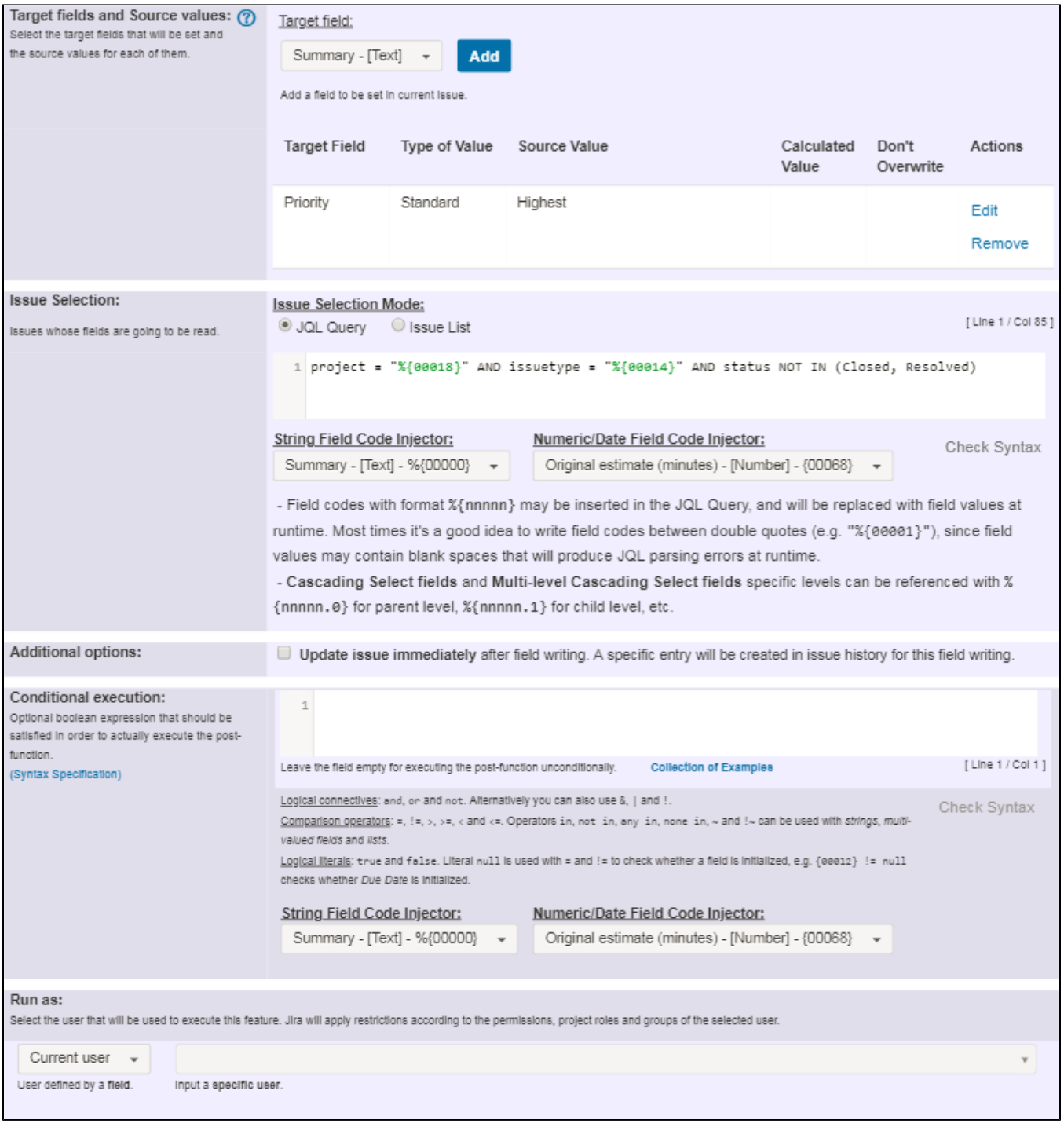
On this page

Purpose

This post-function is used for **reading field values** from issues selected by a **JQL Query** or an **Issue List** expression. The values read are written into a field in current issue.

Example 1: Setting priority on current issue based on other issues priority

We are going to set current issue's Priority with the **highest priority** among issues in the **same project** and with **same issue type** as current issue, which are also in **statuses different from "Resolved"** and **"Closed"**:

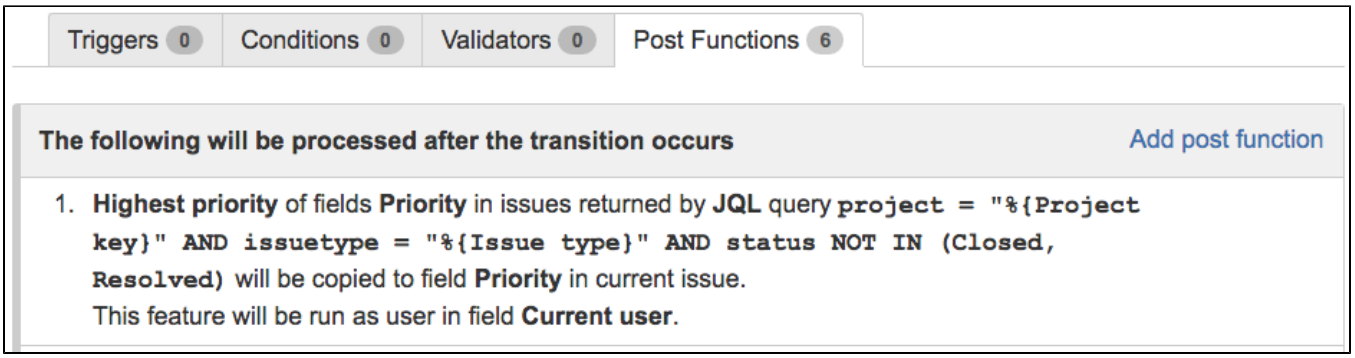


JQL query used is: **`project = "%{00018}" AND issuetype = "%{00014}" AND status NOT IN (Closed, Resolved)`**

Note that:

- **%{00018}** is field code for "**Project issue key**"
- **%{00014}** is field code for "**Issue type**"

Once configured, your post-function will look like this:

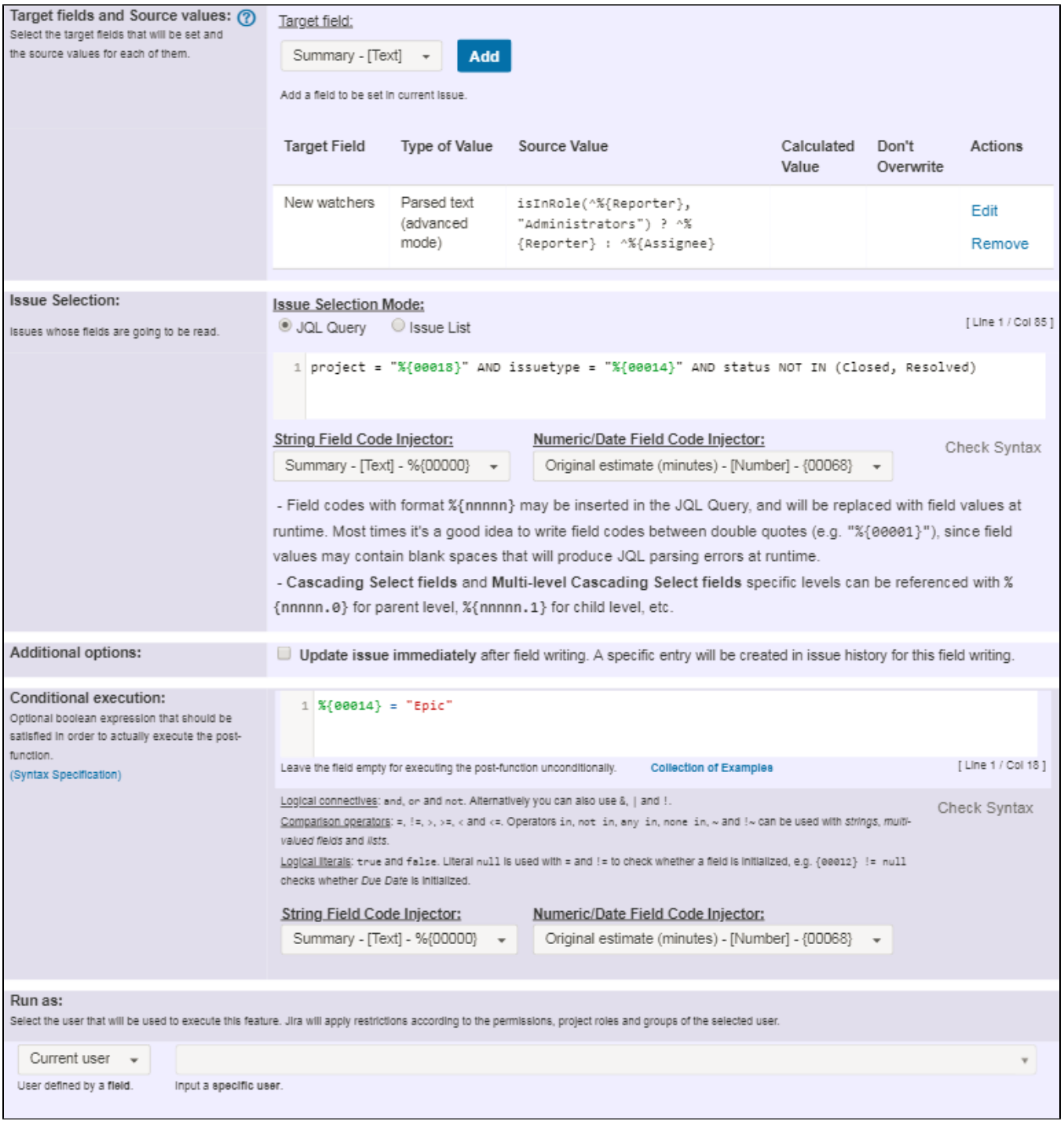


Example 2: Add new watchers into Epic based on assignees and reporters of its tasks and sub-tasks

We want to add as new watcher into **Epic** issue the **reporter** or **assignee** of its **task** and **sub-tasks** using the following logic:

- **Reporter** will be added as watcher if is in role **Administrators**, otherwise the **Assignee** will be added.
  In order to implement this logic we use as source value the following **text expression**:
  `isInRole(^%{00006}, "Administrators") ? ^%{00006} : ^%{00003}`, where **%{00006}** is field code for **Reporter**, and **%{00003}** the one for **Assignee**
- Only **tasks** and **sub-tasks** with **due date** earlier than **epic's due date**, or with **higher priority** than **epic's priority** will be selected.
  In order to implement the issue selection we use the following **issue list** expression:
  `filterByPredicate(subtasks() UNION linkedIssues("is Epic of"), ^{00017} < {00017} OR ^{00012} < {00012})`, where **{00017}** is code for numeric value of **Priority**, and **{00012}** the one for **Due date**. Note that highest priority is **0**
- This post-function will be executed only if current issue is an **Epic**. This way we can use this post-function in a workflow shared with other issue types.

The configuration for the described post-function is:

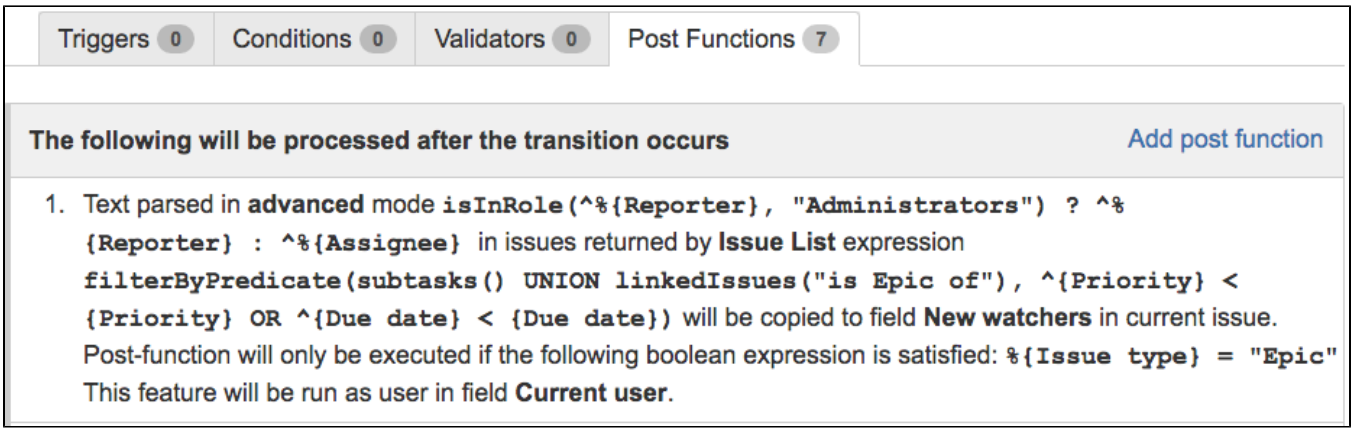


Text to be parsed is:

```
isInRole(^%{Reporter}, "Administrators") ? ^%{Reporter} : ^%{Assignee}

filterByPredicate(subtasks() UNION linkedIssues("is Epic of"), ^{00017} < {00017} OR ^{00012} < {00012})
```

Once configured, your post-function will look like this:



Configuration Parameters

Source Value

There are 3 types of source values available:

- **Field** in selected issues: the value of a field in JQL or Issue List selected issues
- **Parsed text** (**advanced** mode): a string expression where we can use values of fields in current issue (syntax **%{nnnnn}**), and in selected issues (syntax **^%{nnnnn}**). We can use all the functions available in the **Expression Parser**
- **Math** or **Date-Time** expression: an expression returning a numeric value where we can use values of fields in current issue (syntax **{nnnnn}**), and in selected issues (syntax **^{nnnnn}**). We can use all the functions available in the **Expression Parser**

Special Operations depending on Source Field Type

- **Date** and **Date-Time** fields:
  - **Lowest Date**: earliest date among those read.
  - **Highest Date**: latest date among those read.
- **Number** fields:
  - **Sum of Values**: sum of all the values read.
  - **Lowest Value**: minimum value among those read.
  - **Highest Value**: maximum value among those read.
  - **Average Value**: arithmetic mean of values read.
- **Priority** field:
  - **Highest Priority**
  - **Lowest Priority**

Issue Selection Modes

There are 2 different modes for selecting the issues whose field values are going to be read: **JQL query** and **Issue List** expression.

JQL Query

In this issue selection mode we use JQL, which is a language provided by Jira for doing **advanced issue searching**.

You can insert field codes with format **%{nnnnn}** in your JQL query. These field codes will be replaced with the values of the corresponding fields in current issue at execution time, and the resulting JQL query will be processed by Jira JQL Parser. This way you can write dynamic JQL queries that depend on values of fields of current issue.
Example: **`issuetype = "%{00014}" AND project = "%{00018}"`** will return issues in same project and with same issue type as current issue.

When you write your JQL for selecting the issues, take into account the following advices:

- If field values are expected to have **white spaces** or **JQL reserved words or characters**, you should write field code **between quotes** (doub le or simple). Example: **`summary ~ "%{00021}"`** will return issues with current user's full name. As full name can contain spaces, we have written the field code between double quotes.
- In general we will write field codes between quotation marks, since in most cases it doesn't hurt and it's useful for coping with field values containing white spaces or reserved JQL words. Anyway, there is an exception to this general rule: when our field contains a **comma separated list of values**, and we want to use it with JQL operator **IN**. In those cases we will not write the field code between quotes, since

we want the content of the field to be processed as a **list of values**, not as a single string value.

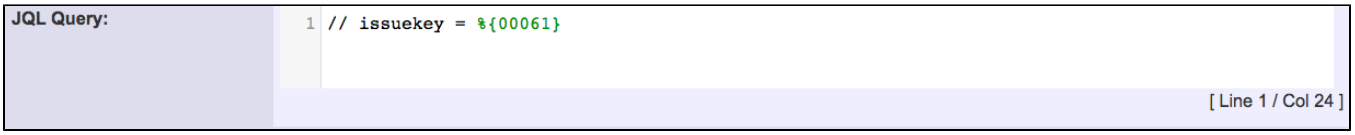
Example: Let's assume that "**Ephemeral string 1**" (field code **%{00061}**) contains a comma separate list of issue keys like **"CRM-1, HR-2, HR-3"**. JQL Query **`issuekey in ("%{00061}")`** will be rendered in runtime like **issuekey in ("CRM-1, HR-2, HR-3")**, which is syntactically incorrect. On the other hand, JQL Query **`issuekey in (%{00061})`** will be rendered in runtime like **issuekey in (CRM-1, HR-2, HR-3)**, which is correct.

Disabling JQL Syntax Pre-Checking

When we enter our JQL query, a syntax pre-checking is carried out in order to verify that it's correctly written. But when we insert field codes in our JQL query, the definitive form of the query that will be executed is unknown, since it depends on the actual values of the fields in runtime. In these cases the syntax pre-checking is done with speculative values given to the fields, and it might happen that fake syntax errors are reported.

In order to inhibit the JQL syntax pre-checking you should enter `//` at the beginning of the line. Those characters will be removed in the actual JQL query that will be executed.

Example:



Issue List expression

In this issue selection mode we use an **issue list expression** according to the **Expression Parser**. Here you can find **Examples of Issue List expressions**.

Additional Options:

- **Don't overwrite target field if it's already set**: when checked, this parameter will make the post-function do nothing in case target field is not empty in current issue.
- **Run as**: Jira user post-function is going to be executed as. This parameter can be set to a **fixed user** (e.g. "john.nash"), or to a **user field** (e.g. "Reporter", "Assignee", etc). This parameter is particularly important in this feature since JQL query will return issues according to the browse permission this user has in the different project of the instance of Jira.

Usage Examples

Page: Use field value as a key for referencing an issue in different project and reading field values in referenced issue

Related Features

- **Update issue fields**
- **Write field on linked issues or sub-tasks**
- **Read fields from linked issues or sub-tasks**