

Add and remove a single or a set of items from multi valued fields

On this page

- [Features used to implement the examples](#)
- [Example 1: Remove "Affects Version/s" from "Fix Version/s" in current issue](#)
- [Example 2: Add components in sub-tasks to components in parent issue](#)
- [Alternative implementation](#)
- [Example 3: Remove reporters of blocking and cloned issues from a "Multi User" field called "Team"](#)
- [Alternative implementation](#)
- [Other examples of that functions](#)
- [Related Usage Examples](#)

Features used to implement the examples

- [Copy parsed text to a field](#)
- [Read fields from linked issues or sub-tasks](#)

Multi Valued fields are fields that allow you to select more than one item:

- Checkboxes
- Select List (multiple choices)
- User Picker (multiple users)
- Group Picker (multiple groups)
- Version Picker (multiple versions)
- Fixed versions
- Affected versions
- Labels
- Components
- Request Participants
- [Issue Picker](#)
- [Insight - Asset Management for Jira's Multi Select](#)

You can use operators + and - to **add** or **remove** lone or set of items from Multi Valued fields.

Adding/Removing values from Current issue or Parent issue

- We use post-function [Copy parsed text to a field](#) to compose the text with operators + and -, and copy it into a multi-valued field of **current issue** or **parent issue** (in the case of sub-tasks).

Adding/Removing values from other issues

- To add or remove a single item or a set items from a multi-valued fields in **other issues (linked issues, sub-tasks, or issues returned by a JQL query)** different from current issue, we use any of the following post-functions:
 - [Write field on linked issues or sub-tasks](#)
 - [Update issue fields](#)

Adding and removing values in a single post-function

A string like + item_1, item_2, item_3, - item_4, item_5 will **add** items **1, 2, 3** and **remove** items **4, 5**. You can also **inject field codes** in the string, and they will be **replaced by its corresponding values** when parsed by post-function at run time.

Example 1: Remove "Affects Version/s" from "Fix Version/s" in current issue

Since we are setting a field in current issue, we can do it in a single step using post-function [Copy parsed text to a field](#) with the following configuration:

Target field:

Fixed versions - [Versions]

Field to be written with the resulting parsed text.

☐ Don't overwrite target field if it's already set.

Parsing Mode:

☒ Basic

Basic mode: Insert field codes anywhere in the text, and they will be replaced with corresponding field values. Field code formats are `%{nnnn}`, and `%{nnnn.i}` for Cascading Select fields (i = 0 for base level).

☐ Advanced

Advanced mode: Strings literals are written in double quotes ("This is a string."). Operator '+' is used to concatenate strings, and field codes are like in basic mode, e.g., "Issue key is " + `%{00015}` + ". ". More information at [parser syntax documentation](#).

Text to be parsed and then copied to target field:

[Line 1 / Col 11] [Syntax Specification](#) [Check Syntax](#)

1

- `%{00077}`

Text to be parsed is: - `%{00077}`

Note that:

- `%{00077}` is field code for "Affects Version/s"

Once configured, the transition looks like this:

The following will be processed after the transition occurs

[Add post function](#)

1. The following text parsed in **basic** mode will be copied to **Fixed versions**:

- `%{Affected versions}`

This feature will be run as user in field **Current user**.

Example 2: Add components in sub-tasks to components in parent issue

We will explain 2 different configurations to implement this same behavior.

Use post-function [Read fields from linked issues or sub-tasks](#) to read **Components** from sub-tasks and write it into **Ephemeral string 1** virtual field:

Target fields and Source values: ?

Select the target fields that will be set and the source values for each of them.

Target field:

Summary - [Text] ▼

Add

Add a field to be set in current issue.

Target Field	Type of Value	Source Value	Calculated Value	Don't Overwrite	Actions
Ephemeral string 1	Field in selected issues	Components			Edit Remove

Filtering by issue link type:

- ☐ is blocked by
- ☐ blocks
- ☐ is cloned by
- ☐ clones
- ☐ is duplicated by
- ☐ duplicates
- ☐ relates to
- ☐ relates to

Only issues linked to current issue by selected issue link types will be read.








Read also subtasks fulfilling condition on issue type, status and project:

This option only makes sense when current issue itself is not a subtask.

Read also sibling subtasks fulfilling condition on issue type, status and project:

Sibling subtasks are understood as subtasks with the same parent as current issue. This option only makes sense when current issue is itself a subtask.

Filtering linked issues or subtasks by issue type:

- ☐  Car
- ☐  Bug
- ☐  Epic
- ☐  Improvement
- ☐  New Feature
- ☐  Story
- ☐ ☒ Task
- ☐  Sub-task

Selected issue types will be read, but if you don't select any, it won't be applied any filter by issue type. In that case all the issue types will be read.

Filtering linked issues or subtasks by status:

- ☐ Open
- ☐ In Progress
- ☐ Reopened
- ☐ Resolved
- ☐ Closed
- ☐ To Do
- ☐ Done
- ☐ Acceptance
- ☐ Fail
- ☐ Pass
- ☐ Retest
- ☐ Active
- ☐ Inactive

Selected statuses will be read, but if you don't select any, it won't be applied any filter by status. In that case issues in any status will be read.

Linked issues or subtasks belong to:

- ☒ any project
- ☐ current project
- ☐ any but current project

Filtering by field values:

Optional boolean expression that should be satisfied by linked issues and subtasks. [\(Syntax Specification\)](#)

1

Leave field empty for no filtering.

[Line 1 / Col 1]

Logical connectives: **or**, **and** and **not**. Alternatively you can also use **|**, **&** and **!**.

Comparison operators: **=**, **!=**, **>**, **>=**, **<** and **<=**. Operators **~**, **!~**, **in**, **not in**, **any in** and **none in** can be used with **strings**, **multi-valued fields** and **lists**.

Logical literals: **true** and **false**. Literal **null** is used with **=** and **!=** to check whether a field is initialized, e.g. `{00012} != null` checks whether **Due Date** is initialized.

[Check Syntax](#)

String Field Code Injector:

Summary - [Text] - %{00000} ▾

Field Code for **Current Issue**

Field Code for **Linked Issues / Subtasks**

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068} ▾

Field Code for **Current Issue**

Field Code for **Linked Issues / Subtasks**

Example 1: `{00012} <= ^{00012}` will require that linked issues and subtasks have *Due Date* equal or later than current issue's *Due Date*.

Example 2: `%{00074} ~ ^%{00074} AND ^%{00017} in ["Blocker", "Critical"]` will require that linked issues and subtasks have *Fixed versions* contained in current issue's *Fixed versions* and *Priority* is *Blocker* or *Critical*.

Read linked issues and subtasks recursively:

☐

Issues and subtasks transitively linked will also be read, provided they fulfill stated filtering conditions. Issues are read recursively without depth limit, but each selected issue is read only once.

Read also current issue:

☐

Current issue will be included in the issue selection, i.e., current issue's field value will also be read.

Conditional execution:
Optional boolean expression that should be satisfied in order to actually execute the post-function.
[\(Syntax Specification\)](#)

1

Leave the field empty for executing the post-function unconditionally.

Collection of Examples

[Line 1 / Col 1]

[Logical connectives:](#) and, or and not. Alternatively you can also use &, | and !.
[Comparison operators:](#) =, !=, >, >=, < and <=. Operators in, not in, any in, none in, ~ and !~ can be used with *strings*, *multi-valued fields* and *lists*.
[Logical literals:](#) true and false. Literal null is used with = and != to check whether a field is initialized, e.g. {00012} != null checks whether *Due Date* is initialized.

String Field Code Injector:
Summary - [Text] - %{00000}

Numeric/Date Field Code Injector:
Original estimate (minutes) - [Number] - {00068}

Check Syntax

Run as:
Select the user that will be used to execute this feature. JIRA will apply restrictions according to the permissions, project roles and groups of the selected user.

Current user

User defined by a **field**.
Input a **specific user**.

Use post-function **Copy parsed text to a field** to compose string with operator + and **Components** of sub-tasks read in previous step and write it into main issue's field **Components**:

Target field:

Components - [Components]

Field to be written with the resulting parsed text.

☐ Don't overwrite target field if it's already set.

Parsing Mode:

☒ Basic

Basic mode: Insert field codes anywhere in the text, and they will be replaced with corresponding field values. Field code formats are `%{nnnnn}`, and `%{nnnnn.i}` for Cascading Select fields (`i` = 0 for base level).

☐ Advanced

Advanced mode: Strings literals are written in double quotes (`"This is a string."`). Operator `+` is used to concatenate strings, and field codes are like in basic mode, e.g., `"Issue key is " + %{00015} + ",."`. More information at [parser syntax documentation](#).

Text to be parsed and then copied to target field:

[Line 1 / Col 11]

Syntax Specification

Check Syntax

1

+ %{00061}

Ephemeral string 1 - [Text] - %{00061}

Insert String Value

Original estimate (minutes) - [Number] - {00068}

Insert Numeric Value

- **Compose dynamic text** by inserting field codes (`%{nnnnn}`) that will be replaced with corresponding field values prior to be copied to target field.

- You can reference parent and child values of **cascading select fields** writing `%{nnnnn.0}` for parent value, and `%{nnnnn.1}` for child value. Use greater indexes to read **multi-level cascading select** custom fields.

- You can change **reporter**, **assignee**, **due date**, **issue status**, **priority**, **resolution**, **labels**, **components**, **fixed versions**, **affected versions**, **original estimate**, **estimated**, **time spent**, and **security level** by choosing the suitable target field and value to be assigned.

- To assign **cascading selects**, **multi selects**, **multi checkboxes**, **components**, **labels**, **fixed versions** and **affected versions** you should use comma or semicolon separated values.

- Additionally, fields of type **Select list**, **Radio button**, **Multi select**, **Multicheck box**, **Multi user**, **Multi groups**, **Components** and **Versions** can be set through regular expressions: options that matches a regular expression can be set by writing `/(regular_expression)/`, and options that doesn't match a regular expression can be set by writing `!(regular_expression)/`.

- **Setting and unsetting individual values** in multi-valued fields, leaving the rest untouched, can be achieved simply by inserting a character `'` or `!` preceding the value or the list of values. You can insert more than one `'` / `!` character in a sole setting operation.

- Fields **Attachments** (**only new attachments will be added**) and **Attachments** (**all current attachments will be replaced**) expect one or more issue keys whose attachments will be copied to current issue.

- You can also use this post-function to **cast a string into a number**.

Conditional execution:

Optional boolean expression that should be satisfied in order to actually execute the post-function.

(Syntax Specification)

1

Leave the field empty for executing the post-function unconditionally.

Collection of Examples

[Line 1 / Col 1]

Check Syntax

Logical connectives:

and, or and not. Alternatively you can also use `&`, `|` and `!`.

Comparison operators:

`=`, `!=`, `>`, `>=`, `<` and `<=`. Operators `in`, `not in`, `any in`, `none in`, `~` and `!~` can be used with *strings*, *multi-valued fields* and *lists*.

Logical literals:

true and false. Literal null is used with `=` and `!=` to check whether a field is initialized, e.g. `{00012} != null` checks whether *Due Date* is initialized.

String Field Code Injector:

Summary - [Text] - %{00000}

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068}

Run as:

Select the user that will be used to execute this feature. JIRA will apply restrictions according to the permissions, project roles and groups of the selected user.

Current user

User defined by a field.

Input a specific user.

Text to be parsed is: + %{00061}

Note that:

- `%{00061}` is field code for "Ephemeral string 1"

Once configured, the transition looks like this:

The following will be processed after the transition occurs

Add post function

1. Fields from the following issues will be read:

Inward issue link types: **none**

Outward issue link types: **none**

Subtasks fulfilling conditions on issue type, status and project **will be read**.

Sibling subtasks won't be read.

Issue types: **any**

Statuses: **any**

Linked issues or subtasks may belong to **any** project.

Target fields and Source values:

Target Field	Type of Value	Source Value	Calculated Value	Don't Overwrite
Ephemeral string 1	Field in selected issues	Components		

This feature will be run as user in field **Current user**.

by JWT

2. The following text parsed in **basic** mode will be copied to **Components**:

+ %{Ephemeral string 1}

This feature will be run as user in field **Current user**.

by JWT

Alternative implementation

Using one single post-function, use post-function [Copy parsed text to a field](#) with the following configuration:

Target field:

Components - [Components]

Field to be written with the resulting parsed text.

☐ Don't overwrite target field if it's already set.

Parsing Mode:

☐ Basic

Basic mode: Insert field codes anywhere in the text, and they will be replaced with corresponding field values. Field code formats are `{nnnnn}`, and `{nnnnn.i}` for Cascading Select fields (`i = 0` for base level).

☒ Advanced

Advanced mode: Strings literals are written in double quotes (`"This is a string."`). Operator `+` is used to concatenate strings, and field codes are like in basic mode, e.g., `"Issue key is " + {00015} + ",."`. More information at [parser syntax documentation](#).

Text to be parsed and then copied to target field:

[Line 1 / Col 51]

Syntax Specification

Check Syntax

1

" + " + toString(fieldValue({00094}, subtasks()))

Components - [Components] - {00094}

Insert String Value

Original estimate (minutes) - [Number] - {00068}

Insert Numeric Value

- **Compose dynamic text** by inserting field codes (`{nnnnn}`) that will be replaced with corresponding field values prior to be copied to target field.
- You can reference parent and child values of **cascading select fields** writing `{nnnnn.0}` for parent value, and `{nnnnn.1}` for child value. Use greater indexes to read **multi-level cascading select** custom fields.
- You can change **reporter**, **assignee**, **due date**, **issue status**, **priority**, **resolution**, **labels**, **components**, **fixed versions**, **affected versions**, **original estimate**, **estimated**, **time spent**, and **security level** by choosing the suitable target field and value to be assigned.
- To assign **cascading selects**, **multi selects**, **multi checkboxes**, **components**, **labels**, **fixed versions** and **affected versions** you should use comma or semicolon separated values.
- Additionally, fields of type **Select list**, **Radio button**, **Multi select**, **Multicheck box**, **Multi user**, **Multi groups**, **Components** and **Versions** can be set through regular expressions: options that matches a regular expression can be set by writing `/(regular_expression)/`, and options that doesn't match a regular expression can be set by writing `!(regular_expression)/`.
- **Setting and unsetting individual values** in multi-valued fields, leaving the rest untouched, can be achieved simply by inserting a character `'` or `'` preceding the value or the list of values. You can insert more than one `'` / `'` character in a sole setting operation.
- Fields **Attachments (only new attachments will be added)** and **Attachments (all current attachments will be replaced)** expect one or more issue keys whose attachments will be copied to current issue.
- You can also use this post-function to **cast a string into a number**.

Conditional execution:

Optional boolean expression that should be satisfied in order to actually execute the post-function.

(Syntax Specification)

1

Leave the field empty for executing the post-function unconditionally.

Collection of Examples

[Line 1 / Col 1]

Logical connectives:

and, or and not. Alternatively you can also use &, | and !.

Comparison operators:

=, !=, >, >=, <, <=.

Operators in, not in, any in, none in, ~ and !~ can be used with strings, multi-valued fields and lists.

Logical literals:

true and false. Literal null is used with = and != to check whether a field is initialized, e.g. {00012} != null checks whether Due Date is initialized.

String Field Code Injector:

Summary - [Text] - {00000}

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068}

Run as:

Select the user that will be used to execute this feature. JIRA will apply restrictions according to the permissions, project roles and groups of the selected user.

Current user

User defined by a field.

Input a specific user.

Text to be parsed: `" + " + toString(fieldValue({00094}, subtasks()))`

Once configured, the transition looks like this:

The following will be processed after the transition occurs

[Add post function](#)

1. The following text parsed in **advanced** mode will be copied to **Components**:

```
" + " + toString(fieldValue(%{Components}, subtasks()))
```

This feature will be run as user in field **Current user**. by JWT

Example 3: Remove reporters of blocking and cloned issues from a "Multi User" field called "Team"

We will explain 2 different configurations to implement this same behavior:

Use post-function [Read fields from linked issues or sub-tasks](#) to read field **Reporter** from linked issues and write it into **Ephemeral string 1**:

Target fields and Source values: ?

Select the target fields that will be set and the source values for each of them.

Target field:

Summary - [Text] ▼

Add

Add a field to be set in current issue.

Target Field	Type of Value	Source Value	Calculated Value	Don't Overwrite	Actions
Ephemeral string 1	Field in selected issues	Reporter			Edit Remove

Filtering by issue link type:

- ☒ is blocked by
- ☐ blocks
- ☒ is cloned by
- ☐ clones
- ☐ is duplicated by
- ☐ duplicates
- ☐ relates to
- ☐ relates to

Only issues linked to current issue by selected issue link types will be read.








Read also subtasks fulfilling condition on issue type, status and project:☐

This option only makes sense when current issue itself is not a subtask.

Read also sibling subtasks fulfilling condition on issue type, status and project:☐

Sibling subtasks are understood as subtasks with the same parent as current issue. This option only makes sense when current issue is itself a subtask.

Filtering linked issues or subtasks by issue type:

- ☐  Car
- ☐  Bug
- ☐  Epic
- ☐  Improvement
- ☐  New Feature
- ☐  Story
- ☐ ☒ Task
- ☐  Sub-task

Selected issue types will be read, but if you don't select any, it won't be applied any filter by issue type. In that case all the issue types will be read.

Filtering linked issues or subtasks by status:

- ☐ Open
- ☐ In Progress
- ☐ Reopened
- ☐ Resolved
- ☐ Closed
- ☐ To Do
- ☐ Done
- ☐ Acceptance
- ☐ Fail
- ☐ Pass
- ☐ Retest
- ☐ Active
- ☐ Inactive

Selected statuses will be read, but if you don't select any, it won't be applied any filter by status. In that case issues in any status will be read.

Linked issues or subtasks belong to:

- ☒ any project
- ☐ current project
- ☐ any but current project

Filtering by field values:

Optional boolean expression that should be satisfied by linked issues and subtasks. (Syntax Specification)

1

Leave field empty for no filtering.

[Line 1 / Col 1]

Logical connectives: **or**, **and** and **not**. Alternatively you can also use **|**, **&** and **!**.

Comparison operators: **=**, **!=**, **>**, **>=**, **<** and **<=**. Operators **~**, **!~**, **in**, **not in**, **any in** and **none in** can be used with **strings**, **multi-valued fields** and **lists**.

Logical literals: **true** and **false**. Literal **null** is used with **=** and **!=** to check whether a field is initialized, e.g. **{00012} != null** checks whether **Due Date** is initialized.

Check Syntax

String Field Code Injector:

Summary - [Text] - %{00000} ▾

Field Code for **Current Issue**

Field Code for **Linked Issues / Subtasks**

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068} ▾

Field Code for **Current Issue**

Field Code for **Linked Issues / Subtasks**

Example 1: **{00012} <= ^{00012}** will require that linked issues and subtasks have *Due Date* equal or later than current issue's *Due Date*.

Example 2: **!^{00074} ~ ^!^{00074} AND ^!^{00017} in ["Blocker", "Critical"]** will require that linked issues and subtasks have *Fixed versions* contained in current issue's *Fixed versions* and *Priority* is *Blocker* or *Critical*.

Read linked issues and subtasks recursively:

☐

Issues and subtasks transitively linked will also be read, provided they fulfill stated filtering conditions. Issues are read recursively without depth limit, but each selected issue is read only once.

Read also current issue:

☐

Current issue will be included in the issue selection, i.e., current issue's field value will also be read.

Conditional execution:
Optional boolean expression that should be satisfied in order to actually execute the post-function.
[\(Syntax Specification\)](#)

1

Leave the field empty for executing the post-function unconditionally.

[Collection of Examples](#)

[Line 1 / Col 1]

[Logical connectives:](#) and, or and not. Alternatively you can also use &, | and !.

[Comparison operators:](#) =, !=, >, >=, < and <=. Operators in, not in, any in, none in, ~ and !~ can be used with *strings*, *multi-valued fields* and *lists*.

[Logical literals:](#) true and false. Literal null is used with = and != to check whether a field is initialized, e.g. {00012} != null checks whether *Due Date* is initialized.

String Field Code Injector:

Summary - [Text] - %{00000} ▾

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068} ▾

Run as:
Select the user that will be used to execute this feature. JIRA will apply restrictions according to the permissions, project roles and groups of the selected user.

Current user ▾

User defined by a field.

Input a specific user.

Check Syntax

Use post-function **Copy parsed text to a field** to compose string with operator – and **Reporter** previously read in step 1 and write it into Multi User field **Team**:

Target field:

Team - [User Picker (multiple users)]

Field to be written with the resulting parsed text.

☐ Don't overwrite target field if it's already set.

Parsing Mode:

☒ Basic

Basic mode: Insert field codes anywhere in the text, and they will be replaced with corresponding field values. Field code formats are `%{nnnnn}`, and `%{nnnnn.i}` for Cascading Select fields (`i` = 0 for base level).

☐ Advanced

Advanced mode: Strings literals are written in double quotes (`"This is a string."`). Operator `'+'` is used to concatenate strings, and field codes are like in basic mode, e.g., `"Issue key is " + %{00015} + ". "`. More information at [parser syntax documentation](#).

Text to be parsed and then copied to target field:

[Line 1 / Col 11]

Syntax Specification

Check Syntax

1

- %{00061}

Summary - [Text] - %{00000}

Insert String Value

Original estimate (minutes) - [Number] - {00068}

Insert Numeric Value

- **Compose dynamic text** by inserting field codes (`%{nnnnn}`) that will be replaced with corresponding field values prior to be copied to target field.

- You can reference parent and child values of **cascading select fields** writing `%{nnnnn.0}` for parent value, and `%{nnnnn.1}` for child value. Use greater indexes to read **multi-level cascading select** custom fields.

- You can change **reporter**, **assignee**, **due date**, **issue status**, **priority**, **resolution**, **labels**, **components**, **fixed versions**, **affected versions**, **original estimate**, **estimated**, **time spent**, and **security level** by choosing the suitable target field and value to be assigned.

- To assign **cascading selects**, **multi selects**, **multi checkboxes**, **components**, **labels**, **fixed versions** and **affected versions** you should use comma or semicolon separated values.

- Additionally, fields of type **Select list**, **Radio button**, **Multi select**, **Multicheck box**, **Multi user**, **Multi groups**, **Components** and **Versions** can be set through regular expressions: options that matches a regular expression can be set by writing `//(regular_expression)/`, and options that doesn't match a regular expression can be set by writing `!(regular_expression)/`.

- **Setting and unsetting individual values** in multi-valued fields, leaving the rest untouched, can be achieved simply by inserting a character `'&'` or `'&'` preceding the value or the list of values. You can insert more than one `'&'` character in a sole setting operation.

- Fields **Attachments (only new attachments will be added)** and **Attachments (all current attachments will be replaced)** expect one or more issue keys whose attachments will be copied to current issue.

- You can also use this post-funcion to **cast a string into a number**.

Conditional execution:

Optional boolean expression that should be satisfied in order to actually execute the post-function.

(Syntax Specification)

1

Leave the field empty for executing the post-function unconditionally.

Collection of Examples

[Line 1 / Col 1]

Check Syntax

Logical connectives:

and, or and not. Alternatively you can also use `&`, `|` and `!`.

Comparison operators:

`=`, `!`, `>`, `>=`, `<` and `<=`. Operators `in`, `not in`, `any in`, `none in`, `~` and `!~` can be used with *strings*, *multi-valued fields* and *lists*.

Logical literals:

`true` and `false`. Literal `null` is used with `=` and `!` to check whether a field is initialized, e.g. `{00012} != null` checks whether *Due Date* is initialized.

String Field Code Injector:

Summary - [Text] - %{00000}

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068}

Run as:

Select the user that will be used to execute this feature. JIRA will apply restrictions according to the permissions, project roles and groups of the selected user.

Current user

User defined by a field.

Input a specific user.

Text to be parsed is: - %{00061}

Note that:

- **%{00061}** is field code for "Ephemeral string 1"

Once configured, the transition looks like this:

The following will be processed after the transition occurs

Add post function

1. Fields from the following issues will be read:

Inward issue link types: **is cloned by** and **is blocked by**.

Outward issue link types: **none**

Subtasks won't be read.

Sibling subtasks won't be read.

Issue types: **any**

Statuses: **any**

Linked issues or subtasks may belong to **any** project.

Target fields and Source values:

Target Field	Type of Value	Source Value	Calculated Value	Don't Overwrite
Ephemeral string 1	Field in selected issues	Reporter		

This feature will be run as user in field **Current user**.

by JWT

2. The following text parsed in **basic** mode will be copied to **Team**:

- **%{Ephemeral string 1}**

This feature will be run as user in field **Current user**.

by JWT

Alternative implementation

Using one single post-function, use post-function **Copy parsed text to a field** with the following configuration:

Target field:

Team - [User Picker (multiple users)]

Field to be written with the resulting parsed text.

☐ Don't overwrite target field if it's already set.

Parsing Mode:

☐ Basic

Basic mode: Insert field codes anywhere in the text, and they will be replaced with corresponding field values. Field code formats are `{nnnnn}`, and `{nnnnn.i}` for Cascading Select fields (`i = 0` for base level).

☒ Advanced

Advanced mode: Strings literals are written in double quotes (`"This is a string."`). Operator `+` is used to concatenate strings, and field codes are like in basic mode, e.g., `"Issue key is " + {00015} + "."`. More information at [parser syntax documentation](#).

Text to be parsed and then copied to target field:

[Line 1 / Col 84]

Syntax Specification

Check Syntax

1

```
"- " + toString(fieldValue({00006}, linkedIssues("is blocked by, is cloned by")))
```

Reporter - [User] - {00006}

Insert String Value

Original estimate (minutes) - [Number] - {00068}

Insert Numeric Value

- **Compose dynamic text** by inserting field codes (`{nnnnn}`) that will be replaced with corresponding field values prior to be copied to target field.
- You can reference parent and child values of **cascading select fields** writing `{nnnnn.0}` for parent value, and `{nnnnn.1}` for child value. Use greater indexes to read **multi-level cascading select** custom fields.
- You can change **reporter**, **assignee**, **due date**, **issue status**, **priority**, **resolution**, **labels**, **components**, **fixed versions**, **affected versions**, **original estimate**, **estimated**, **time spent**, and **security level** by choosing the suitable target field and value to be assigned.
- To assign **cascading selects**, **multi selects**, **multi checkboxes**, **components**, **labels**, **fixed versions** and **affected versions** you should use comma or semicolon separated values.
- Additionally, fields of type **Select list**, **Radio button**, **Multi select**, **Multicheck box**, **Multi user**, **Multi groups**, **Components** and **Versions** can be set through regular expressions: options that matches a regular expression can be set by writing `/(regular_expression)/`, and options that doesn't match a regular expression can be set by writing `!(regular_expression)/`.
- **Setting and unsetting individual values** in multi-valued fields, leaving the rest untouched, can be achieved simply by inserting a character `'` or `!` preceding the value or the list of values. You can insert more than one `'` / `!` character in a sole setting operation.
- Fields **Attachments (only new attachments will be added)** and **Attachments (all current attachments will be replaced)** expect one or more issue keys whose attachments will be copied to current issue.
- You can also use this post-funcion to **cast a string into a number**.

Conditional execution:

Optional boolean expression that should be satisfied in order to actually execute the post-function.

(Syntax Specification)

1

Leave the field empty for executing the post-function unconditionally.

Collection of Examples

[Line 1 / Col 1]

Logical connectives:

and, or and not. Alternatively you can also use `&`, `|` and `!`.

Comparison operators:

`=`, `!`, `>`, `>=`, `<` and `<=`. Operators `in`, `not in`, `any in`, `none in`, `~` and `!~` can be used with *strings*, *multi-valued fields* and *lists*.

Logical literals:

`true` and `false`. Literal `null` is used with `=` and `!` to check whether a field is initialized, e.g. `{00012} != null` checks whether *Due Date* is initialized.

String Field Code Injector:

Summary - [Text] - {00000}

Numeric/Date Field Code Injector:

Original estimate (minutes) - [Number] - {00068}

Run as:

Select the user that will be used to execute this feature. JIRA will apply restrictions according to the permissions, project roles and groups of the selected user.

Current user

User defined by a field.

Input a specific user.

Text to be parsed: `"- " + toString(fieldValue({00006}, linkedIssues("is blocked by, is cloned by")))`

Once configured, the transition looks like this:

The following will be processed after the transition occurs

Add post function

1. The following text parsed in **advanced** mode will be copied to **Team**:

```
"- " + toString(fieldValue({Reporter}, linkedIssues("is blocked by, is cloned by")))
```

This feature will be run as user in field **Current user**.

by JWT

Other examples of that functions

Read fields from linked issues or sub-tasks

Page: Add all assignees of certain sub-task types to a "Multi-User Picker" custom field

Page: Add and remove a single or a set of items from multi valued fields

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: Copy attachments from one issue to another

Page: Make an issue inherit highest priority among those of linked issues

Page: Propagate highest priority from blocked issues to blocking issues

Page: Sum sub-task's "Time Spent" (work logs) and add it to a certain linked issue

Copy parsed text to a field

Page: Add all assignees of certain sub-task types to a "Multi-User Picker" custom field

Page: Add and remove a single or a set of items from multi valued fields

Page: Add current user to comment

Page: Add or remove request participants

Page: Add watchers from a part of the issue summary: "Summary_text - watcher1, watcher2, watcher3, ..."

Page: Assign issue based on the value of a Cascading Select custom field

Page: Assign issue to last user who executed a certain transition in the workflow

Page: Automatically close resolved sub-tasks when parent issue is closed

Page: Automatically reopen parent issue when one of its sub-tasks is reopened

Page: Calculate the time elapsed between 2 transition executions

Page: Close parent issue when all sub-tasks are closed

Page: Combine the values of several Multi-User picker fields

Page: Compose a parsed text including the "full name" or a user

selected in a User Picker custom field

Page: Compose dynamic text by inserting field values in a text template

Page: Copy issue labels to a custom field

Page: Copy the value of a user property into a user picker

Page: Create a comment in sub-tasks when parent transitions

Page: Execute transition in epic

Page: Getting the number of selected values in a custom field of type Multi Select

Page: Limit the number of hours a user can log per day

Page: Make a sub-task's status match parent issue's current status on creation

Page: Make parent issue progress through its workflow

Page: Moving story to "In Progress" when one of its sub-tasks is moved to "In Progress"

Page: Moving story to "Ready for QA" once all its sub-tasks are in "Ready for QA" status

Page: Parse Email addresses to watchers list

Page: Parsing text from last comment and appending it to issue's summary

Page: Remove versions selected in a version picker custom field

Page: Replace certain issue link types with different ones

Page: Restrict parent issue from closing if it has sub-tasks that were created during a given parent issue status

Page: Set a Select or Multi-Select field using regular expression to express the values to be assigned

Page: Set assignee depending on issue type

Page: Set field depending on time passed since issue creation

Related Usage Examples

- Add and remove a single or a set of items from multi valued fields
 - example
 - post-function
 - custom-field
 - issue-links
 - sub-task
- Prevent transitioning when there is a blocking issue
 - example
 - validator
 - issue-links
 - sub-task
 - transition
- Make linked issues, sub-tasks and JQL selected issues progress through its workflows
 - example
 - condition
 - validator
 - post-function
 - issue-links
 - sub-task
 - transition
- Sum sub-task's "Time Spent" (work logs) and add it to a certain linked issue
 - example
 - post-function
 - issue-links
 - sub-task
 - work-log
- Sum "Time Spent" in all sub-tasks of issues linked with issue link types "LinkA", "LinkB", "LinkC"
 - example
 - post-function
 - issue-links
 - sub-task
 - work-log
- Copy "Due date" into a date type custom field in a linked issue if it's greater than current issue's "Due date"
 - example
 - post-function
 - custom-field
 - issue-links
- Create a dynamic set of sub-tasks based on checkbox selection with unique summaries
 - example
 - post-function
 - custom-field
 - sub-task
- Add all assignees of certain sub-task types to a "Multi-User Picker" custom field
 - example
 - post-function
 - custom-field
 - sub-task
- Create a sub-task for each user selected in a Multi-User Picker
 - example
 - post-function
 - custom-field
 - sub-task
- Update Cascading Select custom field with a value of the field in parent issue
 - example
 - post-function

Page: Set priority for issues that have been in a certain status for longer than 24 hours

Page: Set security level based on groups and project roles the reporter or creator are in

Page: Transition linked issues in currently active sprint

Page: Transition only a sub-task among several ones

Page: Transition parent issue only when certain issue sub-task types are done

Page: Update Cascading Select custom field with a value of the field in parent issue

Page: Update checkboxes custom field if a file has been attached during a transition

Page: Validation on issue attachments

Page: Validation on MIME types of issue attachments

Page: Writing a comment to blocked issues when blocking issues are resolved

- custom-field
 - sub-task
- Validate only issue links created in transition screen
 - example
 - validator
 - issue-links
- Require issue link when resolving as duplicate
 - example
 - validator
 - issue-links
- Ensure that all issues linked with a certain issue link type have "Due Date" field set
 - example
 - validator
 - issue-links
- Block an epic's transition depending on linked issues status and due date
 - example
 - validator
 - issue-links
 - transition
- Writing a comment to blocked issues when blocking issues are resolved
 - example
 - post-function
 - issue-links