Create specific sub-tasks for each selected component

On this page

Features used to implement the example

- **Create issues and sub-tasks**

Example: Create specific sub-tasks for each selected component

This is an example of creation of **multiple issues** based on **seed strings** using **Create issues and sub-tasks** post-function.

We want to automatically create specific set of sub-tasks for each specific component selected in current issue. In this particular example we want to create the following sub-tasks for each component:

Component	Sub-tasks
Character	Model, Texture, Animation
Landscape	Illumination, Music
Portrait	Photo, Post-processing

We want that each new sub-task remains unassigned, and that the reporter is the user, who executes the transition in parent issue. The rest of the fields in the new sub-tasks will inherit from parent issue.

We use **Create issues and sub-tasks** post-function in a transition or the workflow of parent issue with the following configuration:

Issues to be created:
Sets the number of issues that will be created.

○ **Only one** issue ◉ **Multiple issues** based on seeds: String List ▾ Check Syntax [Line 3 / Col 61]

String List expression (Syntax Specification and Examples)

```
1 (%{00094} ~ "Character" ? ["Model", "Texture", "Animation"] : []) UNION
2 (%{00094} ~ "Landscape" ? ["Illumination", "Music"] : []) UNION
3 (%{00094} ~ "Portrait" ? ["Photo", "Post-processing"] : [])
```

Input a expression returning a string list. An issue will be created per each string (**seed string**) in the string list. From here on, you will be able to reference seed strings using ^%.

String Field Code Injector: Components - [Components] - %{00094} ▾

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

Issue Type:
Sets the issue type of the issues to be created.

Sub-task ▾

Parent Issue:
Sets the parent of the sub-tasks to be created.

Current Issue ▾

Summary:
Sets the summary of the issues to be created.

Parsing mode: ○ **basic** ◉ **advanced** Check Syntax [Line 1 / Col 22]

```
1 "Sub-task for " + ^%
```

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**.

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

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

Field code injectors reference: ◉ **Current issue** ○ **Seed issue** ○ **Parent of new sub-task**

Description:
Sets the description of the issues to be created.

Parsing mode: ○ **basic** ◉ **advanced** Check Syntax [Line 1 / Col 107]

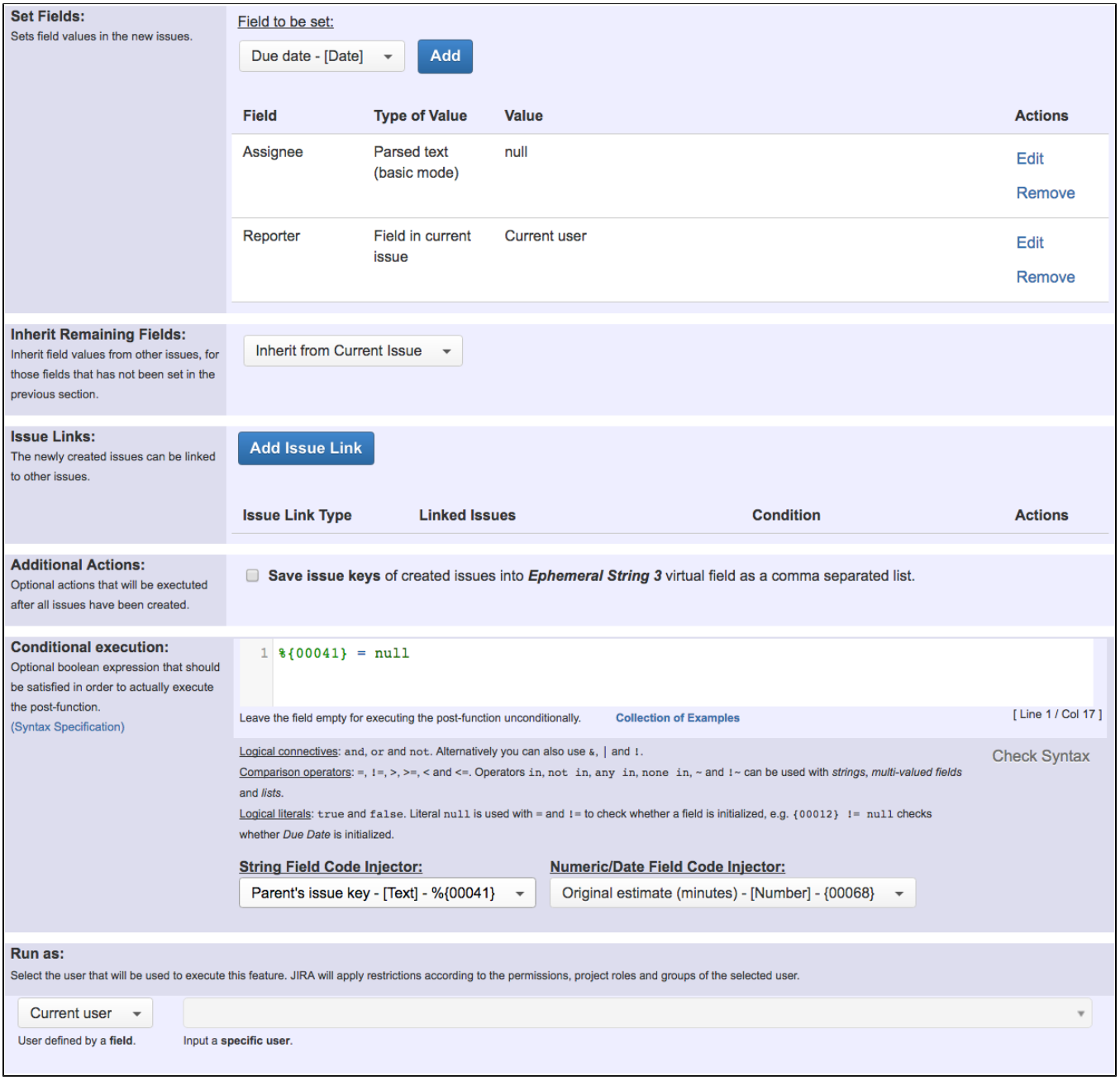
```
1 getMatchingValue(^%, ["Model", "Texture", "Animation", "Illumination", "Music", "Photo",
  "Post-processing"],
2 ["Create the model for the new character.",
3 "Create the texture for the new character.",
4 "Create the animation for the new character.",
5 "Create the illumination for the new landscape.",
6 "Create the music for the new landscape",
7 "Do the photos for the portrait.",
8 "Do the post-processing of the portrait."])
```

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**.

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

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

Field code injectors reference: ◉ **Current issue** ○ **Seed issue** ○ **Parent of new sub-task**



We are using the following expressions:

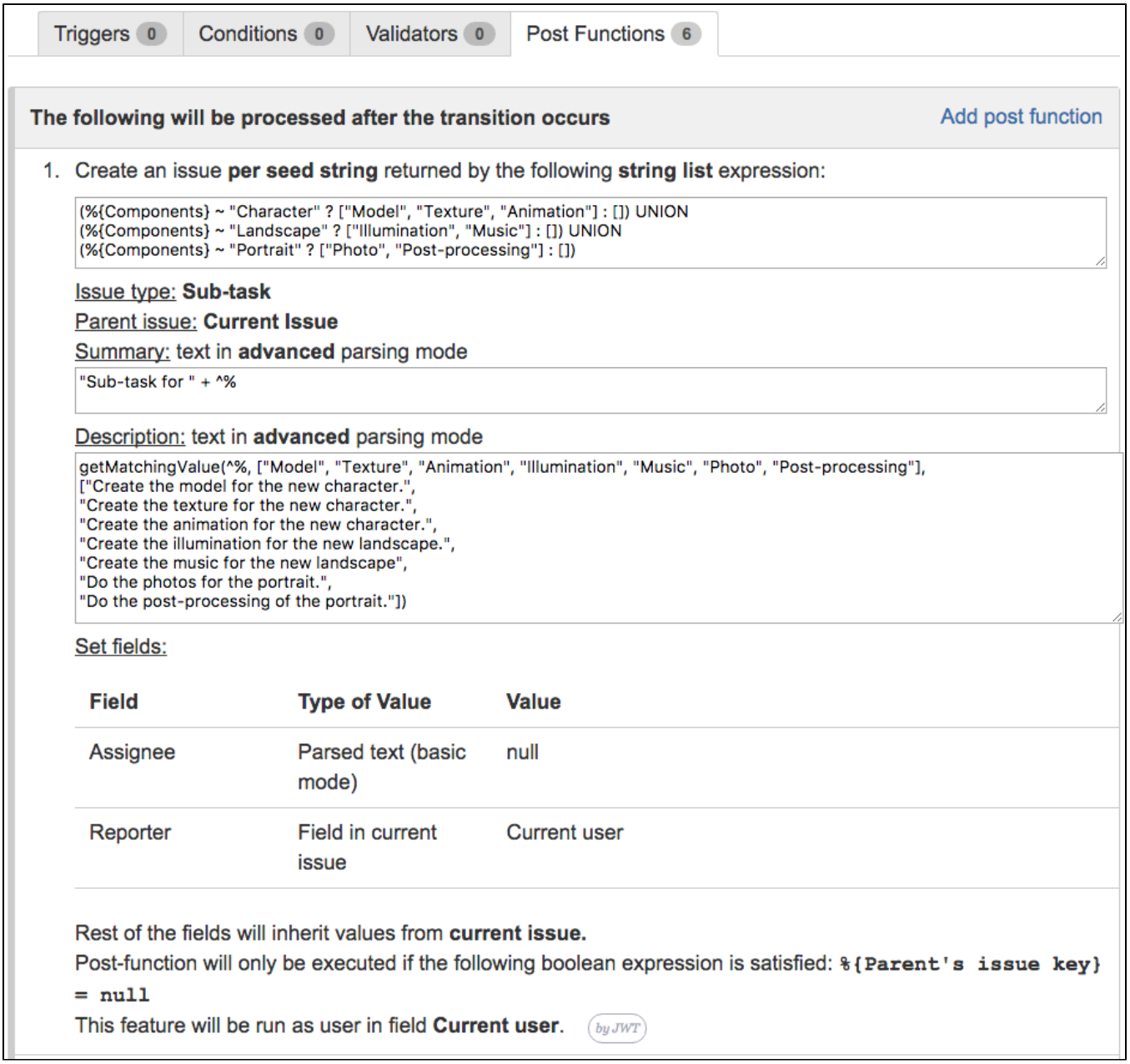
- **Seed strings**
  ```
  (%{00094} ~ "Character" ? ["Model", "Texture", "Animation"] : []) UNION
  (%{00094} ~ "Landscape" ? ["Illumination", "Music"] : []) UNION
  (%{00094} ~ "Portrait" ? ["Photo", "Post-processing"] : [])
  ```
  where **%{00094}** is field code for **Components**

- **Summary** `"Sub-task for " + ^%`

- **Description**
  ```
  getMatchingValue(^%, ["Model", "Texture", "Animation", "Illumination", "Music", "Photo", "Post-
  processing"],
  ["Create the model for the new character.",
  "Create the texture for the new character.",
  "Create the animation for the new character.",
  "Create the illumination for the new landscape.",
  "Create the music for the new landscape",
  "Do the photos for the portrait.",
  "Do the post-processing of the portrait."])
  ```

- **Conditional execution** `%{00041} = null`
  where **%{00041}** is field code for **Parent's issue key**

Once configured, transition will look like this:



Result screenshots post-function "Create issues and sub-tasks" - Create sub-task for each component

The workflow is shared between parent issue and sub-task, thus we are using **Conditional execution** with boolean expression `%{00041} = null` to avoid the post-function to be executed by sub-tasks.

Note that:

- **%{00041}** is field code for **Parent's issue key**

Other examples of that function

Related Usage Examples

Page: Assign new issues to a different project role depending on field value in current issue
Page: Clone an issue and all its subtasks (with additional restrictions)
Page: Create 3 issues in 3 different projects
Page: Create a dynamic set of sub-tasks based on checkbox selection with unique summaries
Page: Create a static set of sub-tasks with unique summaries
Page: Create a story for each component in an epic
Page: Create a sub-task for each user selected in a Multi-User Picker
Page: Create a sub-task in each story of an epic
Page: Create specific sub-tasks for each selected component

- Validation on sibling sub-tasks depending on issue type and status
  - example
  - validator
  - sub-task
  - transition
- Restrict sub-task type creation depending on parent issue status
  - example
  - validator
  - sub-task
- Require at least one sub-task in status "Resolved" or "Closed" when "Testing required" is selected in Check-Box custom field
  - example
  - validator
  - sub-task
- Restrict sub-task type creation depending on parent issue type
  - example
  - validator
  - sub-task
- Block a transition until all sub-tasks have certains fields populated
  - example
  - condition
  - validator
  - sub-task
  - transition
- Create a dynamic set of sub-tasks based on checkbox selection with unique summaries
  - example
  - post-function
  - custom-field
  - sub-task
- Transition sub-tasks when parent is transitioned
  - example
  - post-function
  - sub-task
  - transition
  - outdated
- Transition only a sub-task among several ones
  - example
  - post-function
  - sub-task
  - transition
  - outdated
- Moving sub-tasks to "Open" status when parent issue moves to "In Progress"
  - example
  - post-function
  - sub-task
  - transition
  - outdated
- Moving story to "Ready for QA" once all its sub-tasks are in "Ready for QA" status
  - example
  - post-function
  - sub-task
  - transition
  - outdated
- Add and remove a single or a set of items from multi valued fields
  - example
  - post-function
  - custom-field
  - issue-links
  - sub-task
- Automatically close resolved sub-tasks when parent issue is closed
  - example
  - post-function
  - sub-task
  - transition
  - outdated
- Change parent's status depending on sub-task's summary
  - example
  - post-function
  - sub-task
  - transition
  - outdated

- Moving story to "In Progress" when one of its sub-tasks is moved to "In Progress"
  - example
  - post-function
  - sub-task
  - transition
  - outdated
- Close parent issue when all sub-tasks are closed
  - example
  - condition
  - validator
  - post-function
  - sub-task
  - transition