Create issues and sub-tasks

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

Find the new documentation at:

Create issue

On this page

Purpose

Post-function "**Create issues and subtasks**" allows you to automatically create **one** or **multiple** new **issues** and **sub-tasks** when executing a transition in your workflows. You can also **set the fields of the new issues** based on the values of fields in other issues, and **link the new issues** to the other issues in your Jira instance.

Example: Make Epic issues create automatically 3 Stories when executing a certain transition

In this example we show how to implement a post-function in the workflow of **Epic** issues in order to automatically create 3 Stories with the following summaries: '**GUI Design**', '**Business Logic**' and '**Data Model**'.

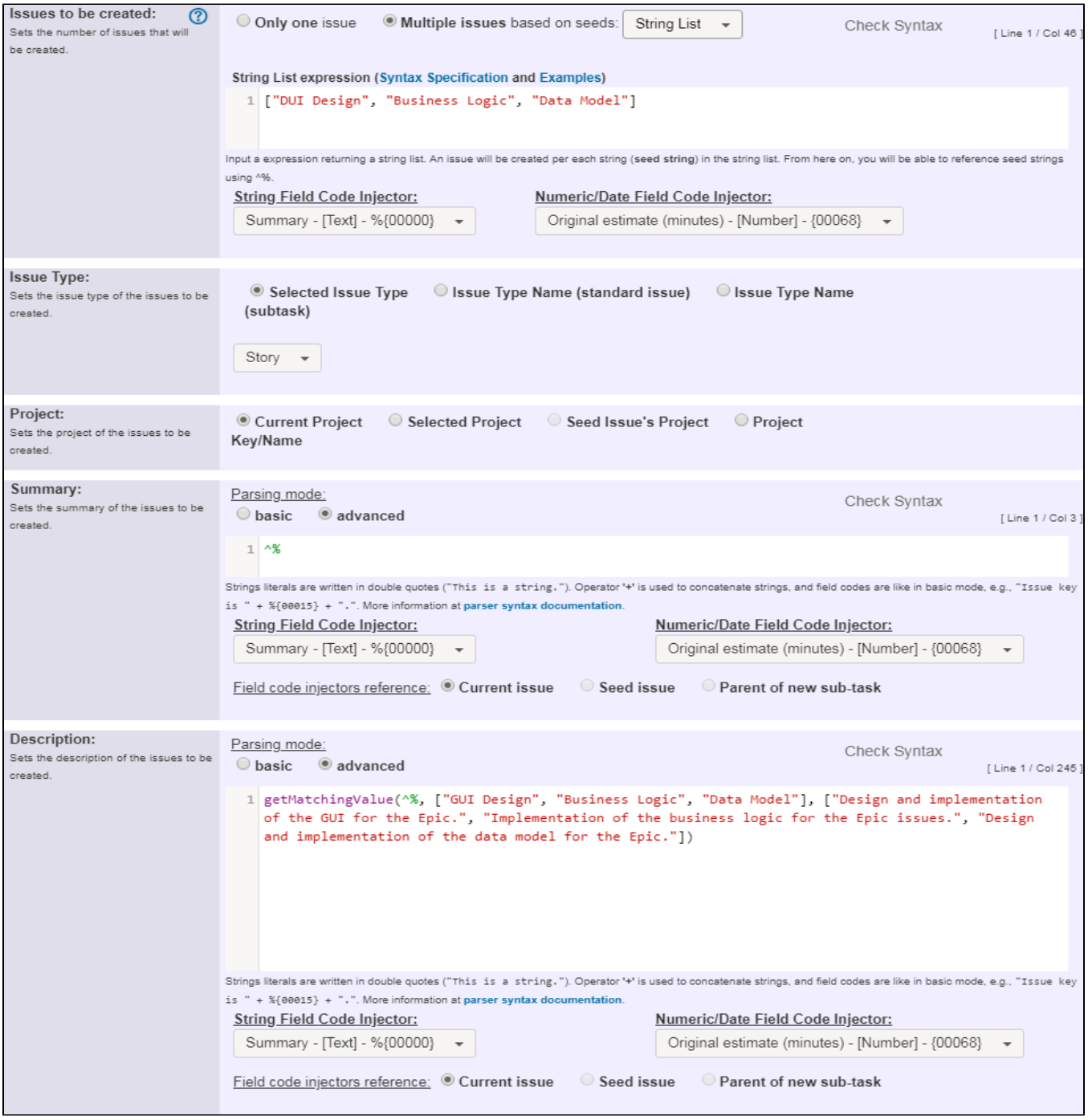
Description of the behavior configured

The 3 new **stories** that will be created will have the following characteristics:

- **Project**: the project where the stories are going to be created is the same as current issue, i.e., the **Epic**.
- **Summary**: will take the value of the **seed string**, i.e., "**GUI Design**", "**Business Logic**" and "**Data Model**". To do it we use string expression **^%**, which is simply the value of the seed string.
- **Description**: we have a different description per each new story:
  - Story created by seed "**GUI Design**": "Design and implementation of the GUI for the Epic."
  - Story created by seed "**Business Logic**": "Implementation of the business logic for the Epic issue."
  - Story created by seed "**Data Model**": "Design and implementation of the data model for the Epic."
    We use the following string expression:
    `getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["Design and implementation of the GUI for the Epic.", "Implementation of the business logic for the Epic issue.", "Design and implementation of the data model for the Epic."])`
- **Set Fields**: we set the following fields in the new story:
  - **Assignee**: we assign each issue to a different user. To do it we write the **name of a project role**, or a **user name** (not to be confused with user's full name). In case we use a project role, we previously should have **set the default user for each project role in each project**.
    We use the following string expression: **`getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["GUI Designer", "Logic Analyst", "Data Architect"])`, where "GUI Designer"**, **"Logic Analyst"** and **" Data Architect"** are project role names.
  - **Reporter**: we use **current user** as value for setting the reporter. Current user is the user who is triggering the transition where the post-function is being executed.
  - **Components**: we set field **Components** with different values per each new story.
    We use the following string expression: **`getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["GUI", "Business Core, Auxiliary Elements", "Data Base, Data Persistency"])`**
  - **Epic Link**: we set **Epic Link** to the value of **Epic Name** in current issue, which is an **Epic**. This way we are linking each new story with its epic. If we were executing the post-function from a **story**, we would have to use field **Epic Link** as value, this way we would be making a **story** to create a new **sibling story**.

- **Execute Transition**: we write the name of transition "**Start Progress**", this way we are moving the new stories to status **In Progress** just after issue creation. We can write into this field more than once, making the new issue to sequentially execute several transitions in its workflow.
- **New comment**: we create a comment in each new issue with a dynamic text using basic parsing mode, where field codes are replaced with their values at runtime.
  We use the following text: **`This issue was created automatically from epic "%{12511}" on %{00057}.`**.
- **New watchers**: we use the following string expression for obtaining the list of users who are watching issues blocked by the **Epic**: **`(toString(fieldValue(%{00133}, linkedIssues("blocks"))))`**, and this way all those users become watchers of the new stories. Note that **%{00133}** is field codes for **Watchers**.

- **Inherit Remaining Fields**: fields not set in section **Set Fields** will inherit the values of the same fields in the **Epic** issue.
- **Issue Links**: each new story will be linked using "**blocks**" issue link type, to all the issues blocked by its **Epic**. This way we are creating a direct blocking relation where there was only an indirect one (through the epic).
- **Conditional execution**: we set a condition in order to ensure that the post-function is only executed when current issue is an **Epic**, this way we can use the post-function in workflows shared with other issue types.



Set Fields:
Sets field values in the new issues.

Field to be set:

Due date - [Date] **Add**

Field	Type of Value	Value	Actions
Assignee	Parsed text (advanced mode)	`getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["GUI Designer", "Logic Analyst", "Data Architect"])`	Edit Remove
Reporter	Field in current issue	Current user	Edit Remove
Components	Parsed text (advanced mode)	`getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["GUI", "Business Core, Auxiliary Elements", "Data Base, Data Persistency"])`	Edit Remove
Epic Link	Field in current issue	Epic Name	Edit Remove
Execute transition	Parsed text (basic mode)	Start Progress	Edit Remove
New comment	Parsed text (basic mode)	This issue was created automatically from epic "%{Epic Name}" on %{Current date and time}.	Edit Remove
New watchers	Parsed text (advanced mode)	`toString(fieldValue(%{Watchers}, linkedIssues("blocks")))`	Edit Remove

Inherit Remaining Fields:
Inherit field values from other issues, for those fields that has not been set in the previous section.

Inherit from Current Issue

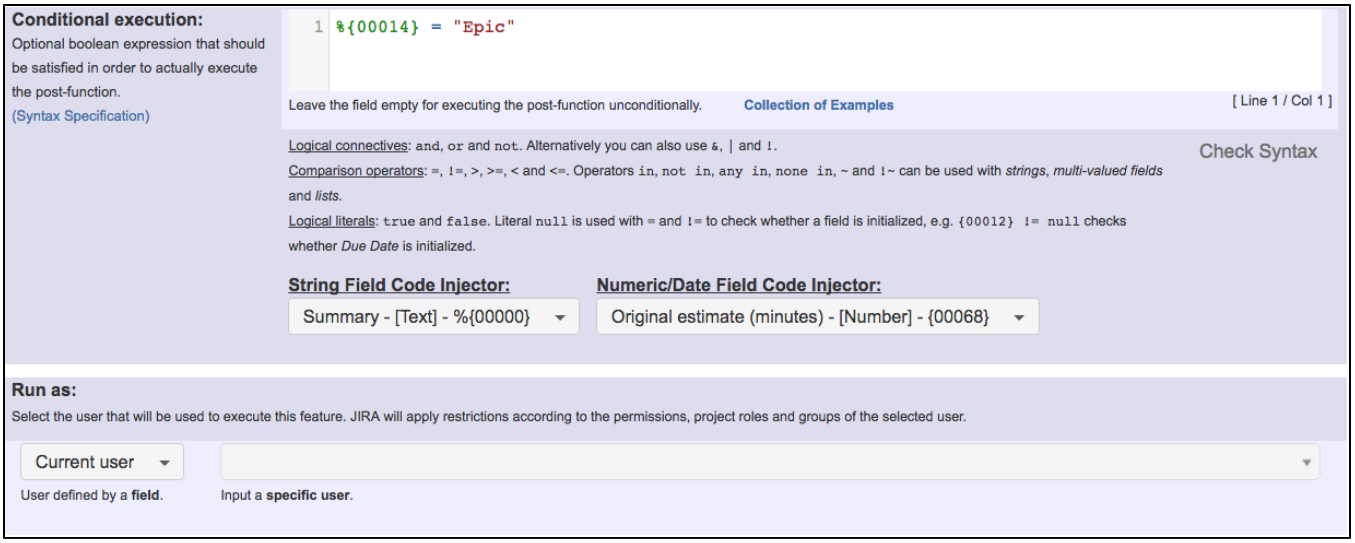
Issue Links:
The newly created issues can be linked to other issues.

Add Issue Link

Issue Link Type	Linked Issues	Condition	Actions
blocks	**Issue List Expression** `linkedIssues("blocks")`		Edit Remove

Additional Actions:
Optional actions that will be exectuted after all issues have been created.

☐ **Save issue keys** of created issues into ***Ephemeral String 3*** virtual field as a comma separated list.



Once configured, transition will look like this:

Triggers 0 | Conditions 1 | Validators 1 | Post Functions 7

The following will be processed after the transition occurs Add post function

1. Create an issue **per seed string** returned by the following **string list** expression:

```
["GUI Design", "Business Logic", "Data Model"]
```

Issue type: **Story**
Project: **Current Project**
Summary: text in **advanced** parsing mode

```
^%
```

Description: text in **advanced** parsing mode

```
getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["Design and implementation of the GUI for the Epic.", "Implementation of the business logic for the Epic issue.", "Design and implementation of the data model for the Epic."])
```

Set fields:

Field	Type of Value	Value
Assignee	Parsed text (advanced mode)	`getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["GUI Designer", "Logic Analyst", "Data Architect"])`
Reporter	Field in current issue	Current user
Components	Parsed text (advanced mode)	`getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["GUI", "Business Core, Auxiliary Elements", "Data Base, Data Persistency"])`
Epic Link	Field in current issue	Epic Name
Execute transition	Parsed text (basic mode)	Start Progress
New comment	Parsed text (basic mode)	This issue was created automatically from epic "%{**Epic Name**}" on %{**Current date and time**}.
New watchers	Parsed text (advanced mode)	`toString(fieldValue(%{Watchers}, linkedIssues("blocks")))`

Rest of the fields will inherit values from **current issue.**

Issue Links:

Issue Link Type	Linked Issues	Condition
blocks	**Issue List expression** `linkedIssues("blocks")`	

Post-function will only be executed if the following boolean expression is satisfied: `%{Issue type} = "Epic"`

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

Screenshots showing an example of execution of the post-function

Configuration Parameters

Issues to be created

Specifies whether we want to create **only one** issue, or **multiple issues**. In case of multiple issues we use **one seed for each new issue**. We should use one of 3 different kinds of seeds:

- **Seed issues**: we use existing issues as seeds for the new issues. We have 2 different methods to select out seeds issues:
  - **JQL Query**: a new issue will be created per each issue returned by a **JQL Query**.
  - **Issue List**: a new issue will be created per each issue returned by an **Issue List expressions**. Fields in seed issues can be referenced using prefix **^** in field codes. Example: **Create a Subtask in each Story of an Epic**.
- **Seed strings**: a new issue will be created per each string returned by a **String List expressions**. This is the method used in the previous example. Seed string can be referenced using **^%** when setting fields or issue links for the new issue. The usage example described above, and these others 3 ones use seed strings: **Create a Story for each Component in Epic**, **Create a sub-task for each user selected in a Multi-User Picker** and **Create specific sub-tasks for each selected component**.
- **Seed numbers**: we use a **Math-Time expressions** (can be a literal number) for specifying the number of issues to be created. Seed number (i.e., number of creation order starting by 1) can be referenced using **^** when setting fields or issue links for the new issue. Example: **Create 3 issues in 3 different projects**.

Issue Type

This parameter specifies the issue type of the new issues or subtasks. When selecting **Story** you will have to set field **Epic Link** in order to set the relationship with **Epic**, like shown in the previous usage example.

Project

This parameter is unavailable for sub-tasks, since they have to belong to the same project as their parent issue. There are 4 methods of specifying the project:

- **Current Project**: the project of the current issue, i.e., the issue that is executing the post-function.
- **Selected Project**: we use a dropdown list for selecting a project among those present in the JIRA instance.
- **Seed Issue's Project**: the project of the seed issue which is causing the issue creation. This option is only available when creating multiple issues based on seed issues.
- **Project Key**: a **string expression** that returns a **project key**. This method is typically used for specifying different a project for each new issue. In the previous example we could have used something like: `getMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["CRM", "HKV", "PKT"])`, where **CRM**, **HKV** and **PKT** are project keys.

Parent Issue

This parameter is available only for creation of sub-tasks, i.e., when the issue type selected in parameter **Issue Type** is a sub-task. There are 6 methods for selecting the parent issue:

- **Current Issue**: the issue that is executing the post-function. It only makes sense when current issue is not a sub-task itself.
- **Parent of Current Issue**: the parent of the issue that is executing the post-function, which obviously should be a sub-task. With this option we are creating a **sibling sub-task**.
- **Seed Issue**: seed issue which is causing the issue creation. It makes sense when seed issue is not a sub-task itself.
- **JQL Query**: we use a JQL query for returning an issue that will be the parent of the sub-task that will be created. If the JQL query returns more than one issue, the first one which isn't a sub-task will be used.
- **Issue List**: we use an **Issue List expressions** for selecting the parent issue. If more than one issue is returned, then the first non-sub-task will be used.
- **Issue Key**: a **string expression** will be used for returning the issue key of the parent. We could use something like `"CRM-23"` for using a fixed issue. In the previous example, we could have used the following string expression for selecting different parents for each new issue: **ge** `tMatchingValue(^%, ["GUI Design", "Business Logic", "Data Model"], ["CRM-23", "CRM-24", "CRM-25"])`.

Summary

We set the summary of the new issues. We can use two different parsing modes:

- **Basic**: field codes with format **%{nnnnn}** can be inserted among the text. The field codes will be replaced with their corresponding field values at run time.

- **Advanced**: we use a **string expression** for setting the summary. In this mode we can insert references to seeds (seed string **^%**, or seed number **^**), or field values on seed issues (format **^%{nnnnn}** and **^{nnnnn}**). We use the **Expression Parser** implemented by the plugin.

Description

We set the description of the new issues. We can use two different parsing modes:

- **Basic**: field codes with format **%{nnnnn}** can be inserted among the text. The field codes will be replaced with their corresponding field values at runtime.
- **Advanced**: we use a **string expression** for setting the summary. In this mode we can insert references to seeds (seed string **^%**, or seed number **^**), or field values on seed issues (format **^%{nnnnn}** and **^{nnnnn}**). We use the **Expression Parser** implemented by the plugin.

Set Fields

This section is used for setting fields in the new issues, including **Assignee**, **Reporter**, **Epic Link**, etc.

Depending on the the field type we can use different methods for specifying the value of the fields.

Post-function **Create Issues and Sub-tasks** allows many different ways for assigning the newly created issues. These are some examples:

Assignment	Value Type	Value/Source	Explanation
Current user	Field in current issue	Current user	The new issue is assigned to the user who is executing the transition containing the post-function.
Parent issue's assignee	Field in current issue	Parent's assignee	If current issue is a sub-task, we are assigning the new issues to the user who currently is assigned parent issue.
Epic's reporter	Field in epic issue	Reporter	If current issue is linked to an epic, the new issues are assigned to the reporter of that epic.
Specific user	Parsed text (basic mode)	A **user name**, not to be confused with user's full name.	In the example we assign the newly created issues to **admin** user.
Load balancing: least busy user in a project role	Parsed text (advanced mode)	A string expression that uses function **leastBusyUserInRole()**.	In the example we assign the issue to the user in project role Developers for current project who has the least number of non-resolved issues in project belonging to category called **'New projects'**. We use the following expression: `leastBusyUserInRole("Developers", %{00018}, "category = 'New projects'")`

Inherit values for remaining fields

Optionally we can inherit field values for the fields whose values have not been set in section **Set Fields**. There are 5 different options for selecting the issue the values will be inherited from:

- **Current Issue:** the issue that is executing the post-function.
- **Seed Issue:** seed issue which is causing the issue creation. This method is only available when creating multiple issues based on seed issues.
- **Parent of Current Issue:** the parent of the issue that is executing the post-function. This method only makes sense when current issue is a sub-task.
- **Parent of New Sub-task:** the parent of the new sub-task. This method is only available when the issue type of the new issue is a sub-task.
- **Epic of Current Issue:** the epic of current issue. If current issue is a sub-task and hasn't **Epic Link** set, then the epic of its parent will be used.

Issue Links

This section is used for specifying issue links to be created between new issues and other issues in the Jira instance. We can set as many issue links as we need, specifying an **issue link type** and a set of **issues to be linked**. There are 7 different methods to define the issues to be linked:

- **Current Issue:** the issue that is executing the post-function.
- **Seed Issue:** seed issue which is causing the issue creation. This method is only available when creating multiple issues based on seed issues.
- **Parent of Current Issue:** the parent of the issue that is executing the post-function. This method only makes sense when current issue is a sub-task.
- **Parent of New Sub-task:** the parent of the new sub-task. This method is only available when the issue type of the new issue is a sub-task.
- **JQL Query**: we use a JQL query for selecting a set of issues that will be the linked to each new issue, using the issue link type previously selected.
- **Issue List**: we use an **Issue List expressions** for selecting a set of issues that will be the linked to each new issue, using the issue link type previously selected.

Additional actions

Optional actions to be carried out once all the new issues have been created. Currently there is only one available action:

- **Save issue keys** of created issues into **Ephemeral String 3** virtual field as a comma separated list:
  Adds a comma separated list of issue keys at the end of the current value of **Ephemeral String 3**, this way we can accumulate in this field all the issues created in subsequent executions of post-function Create issues and sub-tasks within a same post-function. Then we can use this value in another post-function, e.g. sending an email, or creating a comment.

Conditional execution

The post-function will be executed only when the boolean expression entered in this parameter is true, otherwise nothing will happen. You can make your boolean expression depend on the values of one or more fields, issue links, sub-tasks, etc. Use the syntax defined by the **Expression Parser**.

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

Related Features