

Create a story for each component in an epic

On this page

- [Features used to implement the example](#)
- [Example: Create a story for each component in an epic](#)
- [Other examples of that function](#)
- [Related Usage Examples](#)

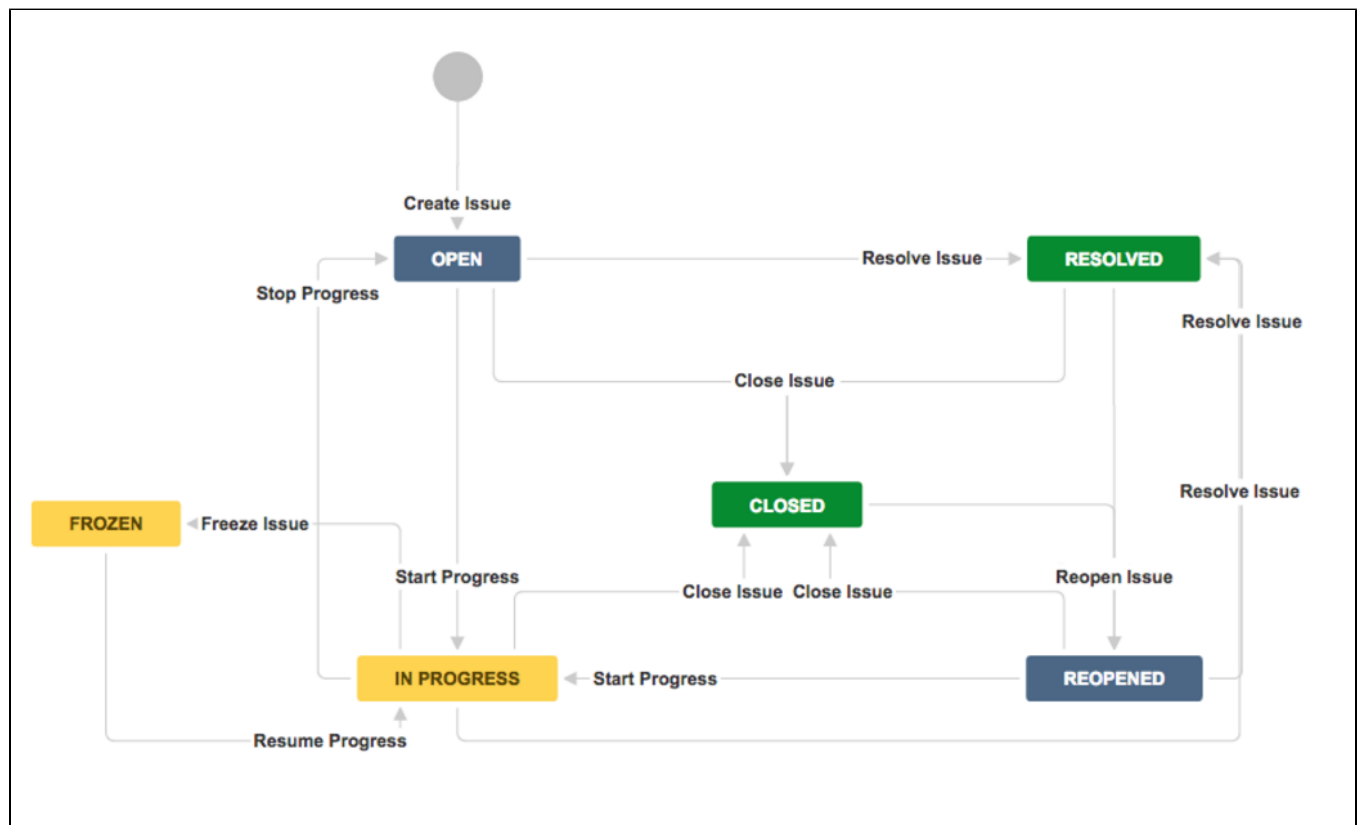
Features used to implement the example

- [Create issues and sub-tasks](#)

Example: Create a story for each component in an epic

In this example we will show how to use [Create issues and sub-tasks](#) for creating a Story for each Component selected in Epic issue.

We have 3 components in our project ("Component A", "Component B" and "Component C"). **Epic** issues and **Stories** share the same workflow:



We want to implement a post-function in "**Start Progress**" transition, that will be executed only by **Epic** issues. This post-function will create a **Story** for each **Component** selected in **Epic** story. The newly created **stories** will have the following characteristics:

- **Summary** will be "Story for component component_name."
- **Description** will be "This is a Story automatically created for component component_name."
- **Assignee** will be the leader of the component corresponding to the new story
- **Reporter**: current user, i.e., the user who executes transition "**Start Progress**" in **Epic** issue
- **Components**: we select in each new Story only the component that motivated its creation
- **Priority**: Stories for "Component A" and "Component C" will have priority **Blocker**, the rest of stories will inherit priority from **Epic** issue
- **Security Level**: all the **Stories** created will inherit the **Security Level** from **Epic** issue

- **Labels:** we will automatically add some labels depending on the **Component** that motivated the creation of the **Story**:
Component A: web-resource.
Component B: sales-strategy and marketing.
Component C: customer-service and user-experience.
- **Automatic transition** execution after Story creation: Just after a **story** is created, it will be moved through the workflow to status "**Frozen**". To do it we execute the following 2 transitions: "**Start Progress**" and "**Freeze Issue**".
- **Comment:** a comment with the following text will be added: "**Issue frozen just after creation on current_date_time.**"

In order to achieve the described behavior we will use the following configuration:

Issues to be created: Sets the number of issues that will be created.	<div> <input type="radio"/> Only one issue <input checked="" type="radio"/> Multiple issues based on seeds: String List </div> <div> String List expression (Syntax Specification and Examples) </div> <div> <pre>1 toStringList(%{00094})</pre> </div> <div> 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 ^%. </div> <div> <div> String Field Code Injector: Summary - [Text] - %{00000} </div> <div> Numeric/Date Field Code Injector: Original estimate (minutes) - [Number] - {00068} </div> </div>
Issue Type: Sets the issue type of the issues to be created.	<div> Story </div>
Project: Sets the project of the issues to be created.	<div> <input checked="" type="radio"/> Current Project <input type="radio"/> Selected Project <input type="radio"/> Seed Issue's Project <input type="radio"/> Project Key </div>
Summary: Sets the summary of the issues to be created.	<div> Parsing mode: <input type="radio"/> basic <input checked="" type="radio"/> advanced </div> <div> Check Syntax </div> <div> <pre>1 "Story for component " + ^%</pre> </div> <div> 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. </div> <div> <div> String Field Code Injector: Summary - [Text] - %{00000} </div> <div> Numeric/Date Field Code Injector: Original estimate (minutes) - [Number] - {00068} </div> </div> <div> Field code injectors reference: <input checked="" type="radio"/> Current issue <input type="radio"/> Seed issue <input type="radio"/> Parent of new sub-task </div>
Description: Sets the description of the issues to be created.	<div> Parsing mode: <input type="radio"/> basic <input checked="" type="radio"/> advanced </div> <div> Check Syntax </div> <div> <pre>1 "This is a Story automatically created for component " + ^% + "."</pre> </div> <div> 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. </div> <div> <div> String Field Code Injector: Summary - [Text] - %{00000} </div> <div> Numeric/Date Field Code Injector: Original estimate (minutes) - [Number] - {00068} </div> </div> <div> Field code injectors reference: <input checked="" type="radio"/> Current issue <input type="radio"/> Seed issue <input type="radio"/> Parent of new sub-task </div>

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)	<code>nthElement(toStringList({Components leaders}), indexOf(^%, toStringList({Components})))</code>	Edit Remove
Reporter	Field in current issue	Current user	Edit Remove
Epic Link	Field in current issue	Epic Name	Edit Remove
Components	Parsed text (advanced mode)	<code>^%</code>	Edit Remove
Priority	Parsed text (advanced mode)	<code>getMatchingValue(^%, ["Component A", "Component C"], ["Blocker", "Blocker", {Priority}])</code>	Edit Remove
Security level	Field in current issue	Security level	Edit Remove
New labels	Parsed text (advanced mode)	<code>getMatchingValue(^%, ["Component A", "Component B", "Component C"], ["web-resource", "sales-strategy marketing", "customer-service user-experience"])</code>	Edit Remove
Execute transition	Parsed text (basic mode)	Start Progress	Edit Remove
Execute transition	Parsed text (basic mode)	Freeze Issue	Edit Remove
New comment	Parsed text (basic mode)	Issue frozen just after creation on %{Current date and time}.	Edit Remove

Inherit Remaining Fields:

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

Don't inherit field values

Issue Links:

The newly created issues can be linked to other issues.

Add Issue Link

Issue Link Type	Linked Issues	Condition	Actions
-----------------	---------------	-----------	---------

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

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

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

1
%{00014} = "Epic"

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.

Once configured, transition **"Start Progress"** will look like this:

OPEN

REOPENED

Start Progress

IN PROGRESS

Screen: None - it will happen instantly

Triggers 0

Conditions 1

Validators 0

Post Functions 8

The following will be processed after the transition occurs

Add post function

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

toStringList(%{Components})

Issue type: Story

Project: Current Project

Summary: text in **advanced** parsing mode

"Story for component " + ^%

Description: text in **advanced** parsing mode

"This is a Story automatically created for component " + ^% + "."

Set fields:

Field	Type of Value	Value
Assignee	Parsed text (advanced mode)	nthElement(toStringList(%{Components leaders}), indexOf(^%, toStringList(%{Components})))
Reporter	Field in current issue	Current user
Epic Link	Field in current issue	Epic Name
Components	Parsed text (advanced mode)	^%
Priority	Parsed text (advanced mode)	getMatchingValue(^%, ["Component A", "Component C"], ["Blocker", "Blocker", %{Priority}])
Security level	Field in current issue	Security level
New labels	Parsed text (advanced mode)	getMatchingValue(^%, ["Component A", "Component B", "Component C"], ["web-resource", "sales-strategy marketing", "customer-service user-experience"])
Execute transition	Parsed text (basic mode)	Start Progress
Execute transition	Parsed text (basic mode)	Freeze Issue
New comment	Parsed text (basic mode)	Issue frozen just after creation on %{Current date and time}.

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**.

Result screenshots post-function "Create issues and subtasks" - Create a Story for each Component in Epic

Issues to be created

We use **multiple issue** creation based on **seed strings**. We need to generate a **string list** with each component selected in the **Epic**. To do it we use the following expression: `toStringList(%{00094})`, where `%{00094}` is field code for **Components**.

From now on, we can reference each **seed string** in expressions using `^%`. We won't be able to reference **seed strings** when we use **basic parsing mode**.

Summary

We use a parsed text in **advanced** parsing mode with the following string expression: `"Story for component " + ^%`.

Description

We use a parsed text in **advanced** parsing mode with the following string expression: `"This is a Story automatically created for component " + ^% + ". "`.

Assignee

We obtain the component leader for the component in the seed string by using **advanced** parsing mode with the following string expression: `nthElement(toStringList(%{00125}), indexOf(^%, toStringList(%{00094})))`, where `%{00125}` returns the **Component leaders** for selected components, and `%{00094}` returns selected **Components** in the same order as the previous field.

Reporter

We use the **Current user**, i.e., the user who is executing the **"Start Progress"** transition in **Epic** issue.

Epic Link

We link newly created **Stories** with **Epic** issue by setting field **Epic Link** in newly created **Stories** with the **Epic Name** in **Epic** issue.

Components

We set field **Components** in newly created **Stories** with the **Component** that motivated the creation of the **Story**, which is in seed string, i.e., `^%`.

Priority

We set field **Priority** using the following text in **advanced** parsing mode: `getMatchingValue(^%, ["Component A", "Component C"], ["Blocker", "Blocker", %{00017}])`.

With this expression we are using priority **Blocker** for issues with **Component A** and **Component C**, and priority in **Epic** issue for issues with the rest of components. Priority in **Epic** is represented by field code `%{00017}`.

Security Level

We use the **Security Level** of the **Epic** issue.

New Labels

We create different labels for each component using the following text expression: `getMatchingValue(^%, ["Component A", "Component B", "Component C"], ["web-resource", "sales-strategy marketing", "customer-service user-experience"])`.

Automatic Transition Execution

We write twice into virtual field **"Execute transition"** with the **names of the transitions** we want to be executed after each **Story** is created. The order of execution of transitions is the same as the order of writing into virtual field **"Execute transition"**. As we are simply writing string literals (transition names), we can use **basic parsing mode**.

New Comment

We write into virtual field **"New comment"** for creating a comment in newly created stories. We use the following text in **basic** parsing mode: `Issue frozen just after creation on %{00057}.`, where `%{00057}` is field code for **Current date and time**.

We can use **basic parsing mode** because we are only using field codes to be replaced by their corresponding field values. If we needed to use advanced parsing function, or seed string references (i.e., `^%`), we would need to use **advanced parsing mode**.

Conditional Execution

Only when boolean expression `%{00014} = "Epic"` is satisfied, post-function will only be executed. This way we can use the post-function in workflows shared with other issue types.

Note that:

- `%{00014}` is field code for **issue type**
-

Other examples of that function

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 Usage Examples

- [Creating a Jira Service Desk internal comment](#)
 - [example](#)
 - [post-function](#)
- [Limit the number of hours a user can log per day](#)
 - [example](#)
 - [validator](#)
 - [post-function](#)
 - [work-log](#)
- [Using project properties to calculate custom sequence numbers](#)
 - [example](#)
 - [post-function](#)
 - [calculated-field](#)
 - [project-properties](#)
- [Set a date based on current date](#)
 - [example](#)
 - [post-function](#)
- [Setting the priority depending on the multiplication of custom fields](#)
 - [example](#)
 - [calculated-field](#)
 - [post-function](#)
- [Parse Email addresses to watchers list](#)
 - [example](#)
 - [post-function](#)
- [Set the assignee based on a condition](#)
 - [example](#)
 - [post-function](#)
- [Create a dynamic set of sub-tasks based on checkbox selection with unique summaries](#)
 - [example](#)
 - [post-function](#)
 - [custom-field](#)
 - [sub-task](#)
- [Create a static set of sub-tasks with unique summaries](#)
 - [example](#)
 - [post-function](#)
- [Triage Jira Service Desk email requests \(Move issues\)](#)
 - [example](#)
 - [post-function](#)
 - [move](#)
 - [transition-issue](#)
- [Moving story to "In Progress" when one of its sub-tasks is moved to "In Progress" \(Transition issues\)](#)
 - [example](#)
 - [post-function](#)
 - [transition](#)
- [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