# **Seeds**

# On this page

Seed issues | Seed texts | Seed numbers

Probably most of the time, doing things with the help of JWT for Jira Cloud is related to a **single object** - the **issue** currently being **transitioned**, e.g. by adding a comment, updating a field, etc.

Referring to the **current issue's information** can be done by using a simple field code like %{issue.description}.

However, since you can also use more complex functions in JWT that work with **multiple objects**, this simple notation is not sufficient for those use cases. To name a few examples

- Creating sub-tasks based on the (dynamic) selection of components set in the current issue
- Create a task for each issue returned by a JQL query (dynamic)
- Create a set of three (static) stories in an Epic with distinct pieces of information

Whenever JWT for Jira Cloud has to handle (or **iterates** over) multiple elements of Lists (or sources), those elements are referred to as **seeds**. Depending on the type of list, those are referred to as

- Seed issues for elements of issue lists
- Seed texts for elements of text lists (e.g. custom field options, components etc.)
- Seed numbers for the numbers from one to a given number



# Seed issues

Issue lists can be specified by issue list expressions like linkedIssues().

When dealing with issue lists, the notation for accessing values of each element is %{seed.issue.someField}, e.g. %{seed.issue.summary}

## **Workflow functions**

You might face seed issues when trying to create multiple issues with the Create issue post function by setting the "Mode" parameter:

| Number of issues to be created         | Description  |
|--|--|
| Multiple issues based on an issue list | An issue is created for every issue returned by the issue list expression. |

Example: You want to create multiple issue based on a parser function that returns three issues, e.g. linkedIssues():

- DEMO-1 Issue A
- DEMO-2 Issue B
- DEMO-3 Issue C

Creating issues based on this result, the post function will run three times, where the following values will be returned throughout those three runs.

| Run | %{seed.issue.key} | %{seed.issue.summary} |
|-----|-------------------|-----------------------|
| 1   | DEMO-1            | Issue A               |

| 2 | DEMO-2 | Issue B |
|---|--------|---------|
| 3 | DEMO-3 | Issue C |

In general, using the seed notation, the nth run returns the field values of the nth issue from this list.

# Seed texts

Text lists can either be

- static, e.g. ["firstElement", "secondElement", "thirdElement"], or
- composed dynamically by using the toStringList() expression parser functions, e.g. toStringList(%{issue.components}) or toStringList(%{issue.cf12345}}) (where the custom field with the ID 123456 is a multi option custom field) or
- calculated by using one of the functions that return a text list like findPattern(), e.g. %{findPattern(%{issue.versions}, "Release")}

When dealing with text lists, the notation for each element %{seed.text}.

#### **Workflow functions**

#### Create issue post function

| Number of issues to be created       | Description  |
|--------------------------------------|--|
| Multiple issues based on a text list | An issue is created for each element of a text list. |

Given the example of a static list above, the post function will run three times and the following values will be returned throughout those three runs

| Run | %{seed.text}  |
|-----|---------------|
| 1   | firstElement  |
| 2   | secondElement |
| 3   | thirdElement  |

Given a dynamic example, having selected the components Frontend and Backend on an issue with  $\{toStringList(\{issue.components\})\}$ , the post function will run two times returning the following values for each run

| Run | %{seed.text} |
|-----|--------------|
| 1   | Frontend     |
| 2   | Backend      |

After adding a third component Interface, the post function will run three times returning the following values for each run

| Run | %{seed.text} |
|-----|--------------|
| 1   | Frontend     |
| 2   | Backend      |
| 3   | Interface    |

According to this scenario, composing a summary with an expression like

```
Summary of %{seed.text} Issue
```

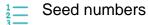
will result in three issues, named

Summary of Frontend Issue

- Summary of Backend Issue
- Summary of Interface Issue

# **Examples**

• Create a sub-task for each component



Numbers can either be

- **static**, e.g. 3, or
- composed dynamically by using the toNumber() expression parser function or
- calculated by using one of the functions that return a number like length(), e.g.  $\{length(%issue.versions))\}$

When dealing with number lists, the notation for each element is  $\{$ seed.number $\}$ .

### **Workflow functions**

Create issue post function

| Number of issues to be created    | Description  |  |
|-----------------------------------|--|--|
| Multiple issues based on a number | The number of issues provided by the numeric value is created. |  |

Given a static example with the numeric value of 3 in order to create three issues, the following values will be returned for each run

| Run | {seed.number} |
|-----|---------------|
| 1   | 1             |
| 2   | 2             |
| 3   | 3             |

According to this scenario, composing a summary with an expression like

```
Summary of Issue # {seed.number}
```

will result in three issues, named

- Summary of Issue # 1
- Summary of Issue # 2
- Summary of Issue # 3

If you still have questions, feel free to refer to our support team.