Jira expression mode

The **Jira expression** parsing mode, unlike the General mode and the Logical mode, is based on a domain-specific language **designed and provided by Atlassian**, not JWT.

It can be used to evaluate custom Jira expressions. Jira expressions follow **JavaScript syntax** and can be thought as a JavaScript dialect. See the introduction section from the official documentation here .

JWT for Jira Cloud only uses the features provided by Atlassian which cannot be modified by JWT for Jira Cloud!

Where are Jira expressions used in JWT for Jira Cloud?

Jira expressions can be used in every context. The Jira expression mode is available in all post functions provided by Jira Workflow Toolbox for Jira Cloud as well as in the Jira expression condition and Jira expression validator.

In order to illustrate with a simple example, the following Jira expression would ensure that an **issue is currently assigned** to a Jira user.

```
issue.assignee != null
```

What is the difference to JWT expressions?

While they might look quite same, it is important to know that **Jira expressions** and the **JWT expressions** have nothing in common in the background.

The equivalent to our former example, using JWT expression field codes would read:

```
%{issue.assignee} != null
```

Apparently, one of the differences to Jira expressions is the way the field codes are referenced.

Rule of thumb!

If you spot **`%{...}`** the syntax is being used in a JWT expression (either the General mode or the Logical mode). If the **{** sign is missing you are likely looking at Jira expressions written in the Jira expression mode.

Another major difference is that you can write your own functions (so-called Arrow functions) in Jira expressions and use complex objects.

Example expressions

Jira expression	Description
`issue.issueType.name == "Story"`	This example makes sure that the current issue type is a "**Story**".
`issue.subtasks.map(s =>  s.key)`	This example returns the keys of all sub-tasks of the current issue as a list: **PU-87,PU-98**

`issue.subtasks.reduce((result, issue) => result.set( issue.status.name, (result[issue.status.name] || 0) + 1), new Map())`	This example counts sub-tasks of the current issue by their status name and returns the result as an object: **{** **"To Do": 1,** **"In Progress": 3** **}**

Additional examples

Field codes

Jira expression	Description
`issue.priority.name == "Medium"`	Issue's **priority** must be **"Medium"**.
`issue.issueType.name.match('^(Bug|Task)$') != null`	Issue **type** must be **"Bug"** or **"Task**" (using a regular expression).
`issue.labels.includes("Support")`	Check if one of the issue's labels is **"Support"**

Numbers and dates

Counting elements

You can count elements by adding **`.length`**

Jira expression	Description
`issue.description.plainText.length >= 100`	Issue's description must be at least **100 characters** in length.
`issue.comments.length >= 5`	Issue must have at least **5 comments.**
`new Date().getDay() != 1`	The current day **must not be Monday.** ⚠️ (see JavaScript reference for getDay())

Lists

Jira expression	Description

`issue.subtasks[0].key`	Accessing a specific element of a list: Getting the key of the first sub-task
`issue.subtasks.length`	Returns the number of the sub-tasks.
`issue.subtasks.map(s => s.status.name)`	Getting a list of the status names of the issue's sub-tasks using the function **map**
`issue.comments.some(com => com.body.plainText.match ('([A-Z][A-Z0-9]+)-\d+') != null)`	Issue must have at least one **comment** containing an **issue key** (using a regular expression).
`issue.subtasks.some(sub => sub.components.some (comp => (comp.name == "QA")))`	Issue must have at least one **sub-task** with the **component "QA"** set.
`issue.subtasks .every(sub => sub.comments .some(com => com.body.plainText.match('([A-Z][A-Z0-9]+)-\d+') != null))`	**Every sub-task** of the issue must have at least one **comment** containing an **issue key** (using a regular expression).
`issue.comments .map(c => c.body.plainText) .filter(text => text.length > 99) .length > 0`	Issue must have at least **one comment** with at least **100 characters**.
`issue.links .filter(link => link.type.name == "Blocks" ) .length == 0`	The issue must not have a **link of type Blocks**.
`issue.links .filter(link => link.linkedIssue.status.name == "Done") .length == issue.links.length`	**All linked issues** must be in the **status** DONE
`issue.links .filter(link => link.linkedIssue.issueType.name == "Bug") .every(link => link.linkedIssue.resolution != null)`	All linked **bugs** must be **resolved**.

Why do I need both?

Right now, Jira expressions are the only officially supported way to formulate custom **conditions** or **validators** in Jira Cloud.

Being the "brain" of **Jira Workflow Toolbox for Server and Data Center**, its JWT expression editor and the underlying **expression parser** have evolved from a small set of handy functions to a comprehensive list and a fundamental part of this app over the years.

Therefore it was only obvious to make this core functionality available for Jira Cloud as well.

JWT for Jira Cloud thus provides two different, powerful means to work with Jira content. In many cases, the functionality is overlapping, e.g. when it comes to certain field codes and functions. Furthermore they complement each other: For instance, the JWT expression parser provides a function for getting issues from a JQL query (issuesFromJQL), which is not provided out of the box by Jira expressions. On the other hand, with Jira expressions you are able to define own functions, which requires basic scripting knowledge, and work with more complex data types.

Choose the appropriate parsing mode depending on your needs and programming skills!

> You cannot mix the different modes in a single expression!

Where do I start?

Familiarize yourself with the Jira expression mode. Once you have a general overview make sure to check out the various **use cases** we have prepared for you.

To deep-dive into Jira expressions we suggest reading up on the additional information we have prepared for you:

- Field codes
- Data types (Jira expressions)
- Operators (Jira expressions)

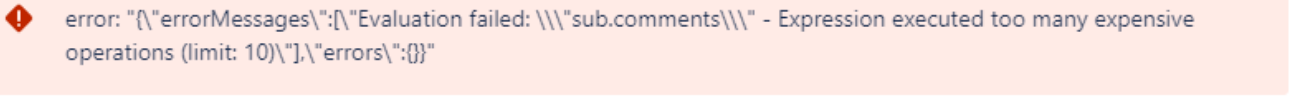
What else should I know?

Jira expressions follow certain constraints with regard to the evaluation of those expressions (see the official documentation).

While the limits should be high enough not to interfere with any intended usage, it's important to realize that they do exist:

- The Expression length is **limited to 1,000 characters** or **100 syntactic elements**.
- Expressions **do not support logging** or **custom error messages**. Any non-boolean value will be considered as false.
- Expressions can execute a maximum of **10 so-called "expensive" operations**, i.e. those that load additional data, such as entity properties, comments, or custom fields, e.g.
  Given an expression to check whether every sub-task has at least one comment containing an issue key, will fail if this issue has more than 10 sub-tasks.

```
issue.subtasks.every(sub => sub.comments.some(com => com.body.plainText.match('([A-Z][A-Z0-9]+)-\d+') != null))
```



Need additional resources?

The best way would be to start with the official documentation:

- Official documentation for Jira expressions: https://developer.atlassian.com/cloud/jira/platform/jira-expressions
- Official documentation for Jira expression types: https://developer.atlassian.com/cloud/jira/platform/jira-expressions-type-reference

---

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