# JQL mode

The **JQL** parsing mode, just like the Issue list mode, lets you define a target **selection** of **issues**. It is being used in Post functions and Automation actions, that allow you to **create** or **update multiple issues, add** or **delete issue links**, etc.

The output has to be a **valid issue list** as defined in the Data types section. The input must be a **valid JQL query** where you can also use field codes.

Additionally, JWT expression parser functions can be used.

## Example expressions

| Parser expression | Description |
|---|---|
| `project = JWT and resolution = Unresolved` | This example returns all **unresolved issues** in the **JWT project**. |
| `type = Bug and project = %{issue.project.key}` | This example returns a **list** of all **Bugs** in the same project as the **current issue**. |
| `issuetype = "%{issue.issueType}" AND project = "%{issue.project.name}"` | This example returns a **list** of all issues in the **same project** and with the same **issue type** as the current issue. |

## JQL queries in JWT

JQL queries in Jira Workflow Toolbox are pretty much written like in the issue navigator. The main difference is, that Field codes can be used which will be **replaced** with **the corresponding values** at runtime.

If field values are expected to have **white spaces** or **JQL reserved words or characters**, you should enclose the field code **in quotes** (double or simple). Example: **`summary ~ "%{system.currentUser.displayName}"`** will return issues with the current user's full name in the summary. As the full name can contain spaces, the field code is enclosed in double quotes.

Anyways, there is an **exception** to this general rule: when the field contains a **comma-separated list of values**, and you want to use it with the JQL operator **IN**. In those cases the field code should not be enclosed in double quotes, since you want the content of the field to be processed as a **list of values**, not as a single text value.

**Example:**

You have temporarily stored the issue keys of all linked issues in a Temporary text field. The field now contains a comma-separated list of the following issue keys: **"CRM-1, HR-2, HR-3".**

| JQL query | JQL query after field code is replaced with values | Description |
|---|---|---|
| `key in ("%{issue.temporaryText1}")` | `key in ("CRM-1, HR-2, HR-3")` | This query will **not** return any results as it is **syntactically incorrect**. ✗ |
| `key in (%{issue.temporaryText1})` | `key in (CRM-1, HR-2, HR-3)` | This query will return **all issues** stored in the temporary field. ✓ |

# Disable JQL syntax pre-checks

When a JQL query is being entered, a syntax pre-check is performed in order to verify that the query is syntactically correct. When field codes are being used, however, the final query that will be executed is unknown, since it **depends** on the **actual values** of the fields in runtime.

In these cases the syntax pre-check is done with **speculative** values assigned to the fields, which might break the syntax check.

In order to **inhibit** the JQL syntax pre-check simply add `//` to the beginning of the query. Those characters will be **removed** once the actual JQL query will be executed.

```
// key in (%{issue.key})
```

Make sure to read all about working with Lists as they come with many extremely useful JWT expression parser functions.

List functions can also be used in the Advanced text mode. The difference is, that the output will be a **flat text** instead of an **issue list**.

The main advantage over the advanced text mode is, that you can use the returned **elements** of the issue list as Seeds**.**

JWT offers individual operators that can be used when working with Lists.

## Available operators

| Function | Short description | Output |
|---|---|---|
| APPEND | **Combines** the elements of two **lists**. | LIST |
| UNION | Returns **distinct elements** of two lists. | LIST |
| INTERSECT | Returns **common elements** of two lists. | LIST |
| EXCEPT | **Removes** certain elements from a list. | LIST |

## Order of operations

If you use multiple operators in a single expression, they will follow a certain order in which they are processed or a precedence.

| OPERATORS | PRECEDENCE | ASSOCIATIVITY |
|---|---|---|
| INTERSECT | 1 (highest) | Left-to-right |
| APPEND, EXCEPT, UNION | 2 (lowest) | Left-to-right |

- When using the list operators, you have to make sure that both lists that you compare are of the **same type**.
- All operators are **case insensitive**, i.e., they can also be written in lower case: **`append`, `union`, `intersect`** and **`except`**.
- There are **four equivalent functions** available for each type of list, and their behavior is **exactly equivalent** to that of its corresponding operator.
  - append()
  - except()
  - intersect()
  - union()
- This way, you can choose to use **operators** or **functions** according to your preference. Although operators yield shorter expressions and with fewer parentheses, the usage of functions produces a more functional consistent syntax.

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