Examples of Issue List expressions

On this page

Examples of Issue List expressions

This page presents a collection of issue selection expressions valid for the **Expression Parser**. All this expressions return an **Issue List** type.

Linked Issues

Epic Link is also a kind of issue link. It's represented by the following 2 issue link types **`has Epic`** and **`is Epic of`**, which are used like this:

- **Epic issue `is Epic of` Story issue**
- **Story issue `has Epic` Epic issue**

Expression	Issues returned	Notes
`linkedIssues()`	Issues linked to current issue through any issue link type, including **Epic Link**.	-
`linkedIssues("is blocked by")`	Issues linked to current one through **is blocked by** issue link type, i.e., **current issue `is blocked by` linked issue.**	-
`linkedIssues("is blocked by, is duplicated by, clones")`	Issues linked to current issue through **is blocked by**, *i***s duplicated by** and **blocks** issue link types.	-
`linkedIssues("has Epic")` Since version 2.3.0 we recommend to use the following simplified syntax: **`epic()`**	An issue list containing only the Epic of current issue.	The returned list will contain 0 or 1 element, depending on whether current issue has an epic issue. Function **`epic()`** has a somehow different behavior than **`linkedIssues("has Epic")`**, since in case current issue is an Epic, an issue list containing current issue will be returned. If current issue is a sub-task of an Epic issue, or a sub-task of an issue directly under an Epic, an issue list containing the Epic will be returned.
`linkedIssues("is Epic of")` Since version 2.3.0 we recommend to use the following simplified syntax: **`issuesUnderEpic()`**	In case current issue is an **Epic** it will return all the issues current issue is epic of.	Function **`issuesUnderEpic()`** has a somehow different behavior than **`linkedIssues("has Epic")`**, since in case current issue is not an Epic but is under an Epic issue, all the issues under current issue's Epic will be returned, including current issue itself. In case current issue is a sub-task of an issue under an Epic, all the issues directly under Epic of current issue's parent will be returned, including current issue's parent, but not their sub-tasks. In case we also want to include the sub-tasks of issues under epic we should use the following syntax: **`issuesUnderEpic() UNION subtasks(issuesUnderEpic())`**

`linkedIssues("is Epic of", linkedIssues("has Epic")))` Since version 2.3.0 we recommend to use the following simplified syntax: **`issuesUnderEpic()`**	Issues with the same epic as current issue.	Current issue is also included in the returned issue list.
`linkedIssues("is Epic of", linkedIssues("has Epic")) EXCEPT issueKeysToIssueList(%{00015})` Since version 2.3.0 we recommend to use the following simplified syntax: **`issuesUnderEpic() EXCEPT issueKeysToIssueList(%{00015})`**	Issues with the same epic as current issue, excluding current issue.	Current issue is not included in the issue list returned. **`{00015}` = Issue key**
`linkedIssues() EXCEPT linkedIssues("is Epic of, has Epic")`	All the issues linked to current issue, except those linked through **has Epic** or **is Epic of** issue link types.	-
`transitionLinkedIssues("")`	Issues that have been linked to current issue in transition screen.	-
`transitionLinkedIssues("blocks")`	Issues that have been linked to current issue in transition screen through **blocks** issue link type.	-
`transitivelyLinkedIssues("is blocked by")`	Issues which are directly or indirectly blocking current issue.	Indirect blocking occurs when an issue is blocking an issue that is directly blocking current issue. Example: **ISSUE-0 `blocks` ISSUE-1 `blocks` ISSUE-2 `blocks` ISSUE-3**, in this case **ISSUE-2** is directly blocking **ISSUE-3**, and **ISSUE-0** and **ISSUE-1** are indirectly blocking **ISSUE-3**.
`linkedIssues("", %{00041})`	Issues linked to parent of current issue.	This expression only makes sense when current issue is a sub-task. **`%{00041}` = Parent's issue key**
`linkedIssues("blocks", %{00041})`	Issues blocked by parent of current issue.	This expression only makes sense when current issue is a sub-task. **`%{00041}` = Parent's issue key**

Sub-tasks

All sub-tasks have one and only one parent issue, and may have sibling sub-tasks, i.e., those issues sharing the same parent issue. Relation between **Epic** and **Stories** is not implemented through parent-child relation, but using issue links "**is Epic of**" and "**has Epic**", as explained above.

Expression	Issues returned	Notes
`subtasks()`	Sub-tasks of current issue.	-
`subtasks(%{00041})`	Sub-tasks of current sub-task's parent, including current sub-task.	**`%{00041}` = Parent's issue key**
`subtasks(linkedIssues("is blocked by"))`	Sub-tasks of all the issues linked to current issue using **is blocked by** issue link type.	-

siblingSubtasks()	Sub-tasks of current sub-task's parent, excluding current sub-task.	-
subtasks(epic())	Sub-tasks of current issue's Epic.	-
subtasks(issuesUnderEpic())	Sub-tasks of issues under current issue's Epic.	-
subtasks(issuesUnderEpic()) EXCEPT subtasks()	Sub-tasks of issues under current issue's Epic excluding current issue's sub-tasks.	-

Filtering Issues Lists

Once we have an issue list, we can filter it by **issue type, status, status category, resolution, project, field values, cardinality** (i,e., number of appearances in the list), or using a **boolean predicate**, which is the most powerful method of issue filtering.

Expression	Issues returned	Notes
filterByIssueType (linkedIssues(), "Improvement, New Feature")	Issue types **"Improvement"** and **"New Feature"** linked to current issue.	-
filterByStatus (filterByIssueType (linkedIssues(), "Improvement, New Feature"), "Open, In Progress")	Issue types **"Improvement"** and **"New Feature"** linked to current issue, which are in statuses **"Open"** or **"In Progress"**.	In this example we are applying 2 filters, one after another, using function composition.
filterByResolution (subtasks(), "Cannot Reproduce, Incomplete")	Sub-tasks with resolutions **"Cannot Reproduce"** or **"Incomplete"**.	-
filterByResolution (subtasks(), "")	Unresolved subtasks.	-
filterByProject (linkedIssues(), "CRM, HR")	Issue that belong to projects with keys **"CRM"** or **"HR"**.	-
filterByPredicate (linkedIssues(), ^%{00016} not in ["Closed", "Resolved"])	**%{00016} = Issue status** Linked issues in statuses different from **"Closed"** and **"Resolved"**.	**^%{00016} not in ["Closed", "Resolved"]** is a boolean expression which should be satisfied in order to pass the filter. We add suffix **^** to field codes in order to reference the values of issues being filtered (i.e., linked issues), instead of current issues values.
filterByPredicate (linkedIssues(), ^%{00016} = %{00016} AND toUpperCase (^%{00000}) ~ toUpperCase ("important")) Since version 2.2.42 case ignoring operator **~~** can be used: **filterByPredicate (linkedIssues(), ^%{00016} = %{00016} AND ^%{00000} ~~ "important")**	Linked issues with the same status as current issue, which also contain the word **"important"** in their summary.	**%{00016} = Issue status** We use function **toUpperCase()** in order to ignore the case when looking for the word **"important"** in issue summaries.

Obtaining Issue Lists using JQL Queries

Issue lists with big numbers of issues are temporarily stored in server's memory. For this reason it's recommended not to build up big lists in your expressions, like retrieving all the issues in a project using function **getIssuesFromProjects("PKEY")**. Instead, it's better to use function **issuesFromJQL("JQL_Query")** using a **JQL_Query** that returns a small number of issues to work with.

Parameter **JQL_Query** is a **string** that represents a valid JQL Query. We typically build dynamic JQL queries inserting field values that we concatenate to string literals using + operator.

Expression	Issues returned	Notes

issuesFromJQL("project = " + %{00018} + " AND issuetype in (Bug, Incident)")	Issues with types **"Bug"** and **"Incident"** in the same project of current issue.	**%{00018} = Project key**
issuesFromJQL("project = " + %{00018} + " AND issuetype = '" + %{00014} + "'")	**%{00018} = Project key** **%{00014} = Issue type** Issues with same issue type and project as current issue.	Note that we have written issue type in simple quotation marks. The reason is that issue type name may contain spaces.