Issue lists

Overview

The **Issue list** **data type** is an ordered list of issues.

This data type is returned by functions returning selections of issues (**linked issues, sub-tasks, issues in a project**, or subsets).

Example

An issue list with 5 elements: `[HR-1,HR-2,HR-3]`

Issue list functions

Issue list functions either return **issue lists** (e.g. [issuekey-1,issuekey-2,issuekey3,...]) or **string lists** or **number lists** for retrieving **issue fields**

The following functions are intended to build expressions that reference **linked issues**, **sub-tasks**, or doing any kind of **issue selection**, and for retrieving their field values.

Function	Input	Returned value
subtasks()		Returns the ISSUE [] of sub-tasks of current issue.
subtasks(issue list **issues**)	ISSUE []	Returns the ISSUE [] of sub-tasks of issues in argument **issues**. Duplicated issues in argument **issues** are discarded. Example: **`subtasks(linkedIssues())`** returns the list of sub-tasks of linked issues.
subtasks(string **issue_keys**)	STRING	Returns the ISSUE [] of sub-tasks of issues whose keys are in **issue_keys**. Argument **issue_keys** is a comma separated list of issue keys. Duplicated issue keys in argument **issue_keys** are discarded. Example: **`subtasks(%{...parentIssuekey})`** returns the list of sub-tasks of parent issue, i.e., sibling sub-tasks plus current sub-task.
siblingSubtasks()		Returns the ISSUE [] of sibling sub-tasks of current issue, i.e., all sub-tasks with the same parent as current issue, except current issue. In case current issue is not a sub-task, an empty issue list will be returned. Note that **`siblingSubtasks()`** is equivalent to **`subtasks(%{...parentIssuekey}) EXCEPT issueKeysToIssueList(%{...Issuekey})`** , where **%{...parentIssuekey}** is Parent's issue key and **%{...Issuekey}** is Issue key.
siblingSubtasks(issue list **issues**)	ISSUE []	Returns the ISSUE [] of sibling sub-tasks of issues in argument **issues**, provided they are sub-tasks. Duplicated issues in argument **issues** are discarded.
siblingSubtasks(string **issue_keys**)	STRING	Returns the ISSUE [] of sibling sub-tasks of issues whose keys are in **issue_keys**, provided they are sub-tasks. Argument **issue_keys** is a comma separated list of issue keys. Duplicated issue keys in argument **issue_keys** are discarded.
linkedIssues()		Returns the ISSUE [] of issues linked to current issue, including **Epic-Task** links. An issue appears in the output as many times as is linked to current issue. Function **distinct**(issue list) can be used to remove duplicated issues. Example: **`distinct(linkedIssues() EXCEPT linkedIssues("has Epic, is Epic of"))`** returns all the issues linked to current issue, excluding **Epic-Task** issue links.

linkedIssues (string **issue_link_types**)	STRING	Returns the ISSUE [] of issues linked to current one using issue link types in argument **issue_link_types**. Argument **issue_link_types** is a comma separated list of issue link type names, or an empty string ("") for representing all issue link types, i.e., **`linkedIssues("")`** is equivalent to **`linkedIssues()`** . Example: **`linkedIssues("blocks, clones")`** returns all issues linked with to current issue using issue link types blocks or clones.
linkedIssues (string **issue_link_types**, issue list **issues**)	STRING ISSUE []	Returns the ISSUE [] of issues linked to those ones in argument **issues** using issue link types in argument **issue_link_types**. Duplicated issues in argument **issues** are discarded. Example: **`linkedIssues("", subtasks())`** returns all issues linked to current issue's sub-tasks using any issue link type.
linkedIssues (string **issue_link_types**, string **issue_keys**)	STRING	Returns the ISSUE [] of issues linked to those ones whose keys are in argument **issue_keys**. Argument **issue_keys** is a comma separated list of issue keys. Duplicated issue keys in argument **issue_keys** are discarded. Example: **`linkedIssues("is blocked by", %{...parentIssuekey})`** returns all issues blocking parent issue.
transitionLinkedIssues(string **issue_link_types**)	STRING	Returns the ISSUE [] of issues linked to current one with links created in current transition screen using issue link types in argument **issue_link_types**. Argument **issue_link_types** is a comma separated list of issue link type names, or an empty string **`("")`** for representing all issue link types, i.e., **`transitionLinkedIssues("")`** is equivalent to **`transitionLinkedIssues()`** . This function is useful for validating only new issue links created by user in transition screen. Example: **`transitionLinkedIssues("blocks, clones")`** returns the list of issues linked in current transition's screen using issue link types **blocks** and **clones**.
transitivelyLinkedIssues (string **issue_link_types**)	STRING	Returns the ISSUE [] of issues directly or transitively linked to current issue using issue link types in argument **issue_link_types**. Argument **issue_link_types** is a comma separated list of issue link type names, or an empty string **`("")`** for representing all issue link types. Example of transitive link: if **ISSUE-1 blocks ISSUE-2 blocks ISSUE 3**, then **ISSUE-1** is blocking transitively **ISSUE-3**.
transitivelyLinkedIssues (string **issue_link_types**, issue list **issues**)	STRING ISSUE []	Returns the ISSUE [] of issues directly or transitively linked to those ones in argument **issues** using issue link types in argument **issue_link_types**. Argument **issue_link_types** is a comma separated list of issue link type names, or an empty string **`("")`** for representing all issue link types.
transitivelyLinkedIssues (string **issue_link_types**, string **issue_keys**)	STRING	Returns the ISSUE [] of issues directly or transitively linked to those ones in argument **issue_keys** using issue link types in argument **issue_link_types**. Argument **issue_link_types** is a comma separated list of issue link type names, or an empty string **`("")`** for representing all issue link types.
epic()		Returns an ISSUE [] containing current issue's epic, in case current issue is directly under an epic (e.g., a **Story**). If current issue is a sub-task, then the epic of its parent issue is returned. If current issue is an epic itself, then current issue is returned.
epic(issue list **issues**)	ISSUE []	Returns the ISSUE [] of epic issues under which those issues in argument **issues** are. If some of those issues are sub-tasks, then the epic of their parent is returned. Duplicated issues in argument issues are discarded. Output can contain duplicated issues. Example: **`epic(linkedIssues("is blocked by"))`** returns the list of epics of those issues which are blocking current issue.

epic(string **issue_keys**)	STRING	Returns the ISSUE [] of epic issues under which those issues with keys in **issue_keys** are. If some of those issues are sub-tasks, the epic of their parent is returned. Argument **issue_keys** is a comma separated list of issue keys. Duplicated issue keys in argument **issue_keys** are discarded. Output can contain duplicated issues. Example: **`epic("CRM-15, HD-21")`** returns the list of epics under which issues with keys **CRM-15** and **HD-21** are.
issuesUnderEpic()		Returns an ISSUE [] containing issues which are directly under current issue's epic (i.e., **Stories** are included in the output, but their **sub-tasks** are not). Current issue's epic is obtained using the logic of function epic() . Current issue is included in the output, except if current issue is an epic itself.
issuesUnderEpic(issue list **issues**)	ISSUE []	Returns an ISSUE [] containing issues which are directly under the epic of issues in argument **issues**. Duplicated issues are filtered from output. Example: **`issuesUnderEpic(linkedIssues("is blocked by"))`** returns the list of issues directly under epics of issues blocking current issue.
issuesUnderEpic(string **issue_keys**)	STRING	Returns an ISSUE [] containing issues which are directly under the epic of issues with keys in argument **issue_keys**. Argument **issue_keys** is a comma separated list of issue keys. Duplicated issues are filtered from output. Example: **`issuesUnderEpic("CRM-15, HD-21")`** returns the list of issues directly under epic of issues with keys **CRM-15** and **HD-21**.
siblingIssuesUnderEpic()		Returns an ISSUE [] containing issues which are directly under epic of current issue (i.e., **Stories** are included in the output, but their **sub-tasks** are not), excluding current issue. Current issue should be an issue directly under an epic, (i.e., it can't be a **sub-task** or an **epic**).
siblingIssuesUnderEpic(issue list **issues**)	ISSUE []	Returns an ISSUE [] containing issues which are directly under the epic of issues in argument **issues**, excluding issues in argument **issues** from the output. Duplicated issues are filtered from output. Example: **`siblingIssuesUnderEpic(linkedIssues("is blocked by"))`** returns the list of issues directly under epics of issues blocking current issue, excluding from the output issues blocking current issue.
siblingIssuesUnderEpic(string **issue_keys**)	STRING	Returns an ISSUE [] containing issues which are directly under the epic of issues with keys in argument **issue_keys**, excluding from the output issues with keys in argument **issue_keys**. Argument **issue_keys** is a comma separated list of issue keys. Duplicated issues are filtered from output. Example: **`siblingIssuesUnderEpic("CRM-15, HD-21")`** returns the list of issues directly under epic of issues with keys **CRM-15** and **HD-21**, excluding from the output issues with keys **CRM-15** and **HD-21**.
issuesFromJQL(string **jql_query**)	STRING	Returns the ISSUE [] resulting of the execution of a JQL query represented by string argument **jql_query**. Visibility permissions applied are those of **current user**. We advice to use this function for performance reasons when the number of issues to be retrieved or filtered is very high (all issues in a project or various projects). Typically you will want to use this function for replacing any current expression using **getIssuesFromProjects()** function.
issuesFromJQL(string **jql_query**, string **user_name**)	STRING	Returns the ISSUE [] resulting of the execution of a JQL query represented by string argument **jql_query**. Visibility permissions applied are those of user in argument **user_name**. We advice to use this function for performance reasons when the number of issues to be retrieved or filtered is very high (all issues in a project or various projects). Typically you will want to use this function for replacing any current expression using **getIssuesFromProjects()** function.

filterByIssueType(issue list **issues**, string **issue_types**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those issue types appearing in argument **issue_types**. Argument **issue_types** is a comma separated list of issue type names. Example: **`filterByIssueType(subtasks(), "Bug, Improvement, New Feature")`** returns the list of sub-tasks with issue types **Bug**, **Improvement** or **New Feature**.
filterByStatus(issue list **issues**, string **statuses**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those ones in statuses appearing in argument **statuses**. Argument **statuses** is a comma separated list of status names. Example: **filterByStatus(linkedIssues("is blocked by"), "Open, Reopened, In Progress")** returns the list of blocking issues in statuses **Open**, **Reopened** or **In Progress**.
filterByStatusCategory(issue list **issues**, string **status_categories**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those ones in statuses with categories in **status_categories**. Argument **status_categories** is a comma separated list of status category names. Example: **`filterByStatusCategory(linkedIssues("is blocked by"), "New, In Progress")`** returns the list of blocking issues in statuses with categories **New** or **In Progress**.
filterByResolution(issue list **issues**, string **resolutions**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those ones with resolutions appearing in argument **resolutions**. Argument **resolutions** is a comma separated list of resolution names. If this argument receives an empty string **`("")`**, the function will return issues with unset field Resolution. Example: **`filterByResolution(subtasks(), "Won't Fix, Cancelled")`** returns the list of sub-tasks with resolutions **Won't Fix** or **Cancelled**.
filterByProject(issue list **issues**, string **projects**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those ones in projects present at argument **projects**. Argument **projects** is a comma separated list of project keys. Example: **`filterByProject(linkedIssues(), "CRM, HR")`** returns the list of linked issues belonging to projects with keys **CRM** or **HR**.
filterByProjectCategory (issue list **issues**, string **project_categories**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those ones in projects with category in **project_categories**. Argument **project_categories** is a comma separated list of project category names. Example: **`filterByProjectCategory(linkedIssues(), "Development, Production")`** returns the list of linked issues belonging to projects in categories keys **Development** or **Production**.
filterByFieldValue(issue list **issues**, numeric field **field**, comparison operator **operator**, number **n**)	ISSUE [] NUMBER	Filters ISSUE [] in argument **issues**, leaving only those issues where logical predicate formed by arguments **field operator n** is evaluated as true. Available comparison operators are **`=, !=, <, <=, >`** and **`>=`** . Argument **field** has format **{...somefield}**. Example: **`filterByFieldValue(subtasks(), {00079}, >, 1)`** returns sub-tasks with more than one **Affects Version/s**.
filterByFieldValue(issue list **issues**, string field **field**, comparison operator **operator**, string **s**)	ISSUE [] STRING	Filters ISSUE [] in argument **issues**, leaving only those issues where logical predicate formed by arguments **field operator s** is evaluated as true. Available comparison operators are **`=, !=, <, <=, >, >=, ~, !~, in`** and **`not in`**. Case ignoring operators are also available: **`=~, !=~, ~~, !~~, in~`** and **`not in~`** . Argument **field** has format **%{...somefield}** for string fields, or **%{...somefield.i}** for cascading select fields. Example: **`filterByFieldValue(linkedIssues(), %{...components}, ~, "Web")`** returns linked issues with component "**Web**".

filterByCardinality(issue list **l**, comparison operator **operator**, number **n**)	ISSUE [] NUMBER	Returns ISSUE [] in **l** whose cardinality (i.e., the number of times it appears in list **l**) satisfies the comparison cardinality **operator n**. Available comparison operators: **`=, !=, <, <=, >`** and **`>=`** . Example: **`filterByCardinality(linkedIssues(), >, 1)`** returns a list with all issues linked to current issue with 2 or more issue links.
append(issue list **l**, issue list **m**)	ISSUE []	Returns ISSUE [] with all issues in arguments **l** and **m** . Duplicated issues may appear in output. Use function **union(l, m)** instead, if you want to avoid repetitions. Example: **`append(linkedIssues("is blocked by"), subtasks())`** returns the list blocking issues plus sub-tasks. If a sub-task is also linked with issue link type "**is blocked by**", it will appear twice in the output list.
union(issue list **l**, issue list **m**)	ISSUE []	Returns ISSUE [] with all issues in argument **l** or in argument **m** without duplicated issues. Example: **`union(linkedIssues(), subtasks())`** returns the list of linked issues and sub-tasks of current issue, without issue repetitions.
except(issue list **l**, issue list **m**)	ISSUE []	Returns ISSUE [] with all issues in argument **l** which are not in argument **m**. Duplicated issues in **l** may appear in output. Use function **distinct()** to remove them if you need to. Example: **`except(linkedIssues(), subtasks())`** returns the list of linked issues removing those which are also sub-tasks of current issue.
intersect(issue list **l**, issue list **m**)	ISSUE []	Returns ISSUE [] with all issues in argument **l** and **m** simultaneously. Example: **`intersect(linkedIssues(), subtasks())`** returns the list of linked issues which are also sub-tasks of current issue.
distinct(issue list **l**)	ISSUE []	Returns ISSUE [] with all issues in list **l** without any duplication. Example: **`distinct(linkedIssues())`** returns the list of linked issues, with only one occurrence per issue, although an issue may be linked with more than one issue link type.
fieldValue(string field **field**, issue list **issues**)	STRING ISSUE []	Returns the STRING [] of string values stored in argument **field** in those issues in argument **issues**. Argument field has format **%{...somefield}**, or **%{...somefield.i}** for cascading select fields. The number of values in output is the number of issues in argument **issues** with **field** set, except for multi-valued fields, for which a value is returned for each selected value in the field. Multi-valued fields are fields of types **Multi Select, Checkboxes, Components, Versions, Multi User Picker, Multi Group Picker, Issue Pickers, Attachments** and **Labels**. Example: **`fieldValue(%{...reporter}, subtasks())`** returns the list of **reporter users** of sub-tasks.
fieldValue(numeric field **field**, issue list **issues**)	NUMBER ISSUE []	Returns the NUMBER [] of numeric values stored in argument **field** in those issues in argument **issues**. Argument **field** has format **{...somefield}**. The number of values in output is the number of issues in argument **issues** with **field** set. Example: **`fieldValue({...duedate}, subtasks())`** returns the list of **Due Dates** of sub-tasks.
textOnIssueList(issue list **issues**, string **text_expression**)	ISSUE [] STRING	Returns a STRING [] resulting of evaluating **text_expression** against each of the issues in argument **issues**. Argument **text_expression** is an expression that returns a **string**, where references to field values of issues in argument issues are done with prefix **^** before field code, e.g., **^%{...summary}** is field code for Summary in each of the issues in argument **issues**. Example: **`textOnIssueList(subtasks(), ^%{...assignee} = ^%{...reporter} ? ^%{...Issuekey} : null)`** returns the issue keys of sub-tasks with same user as reporter and as assignee.

mathOnIssueList(issue list **issues**, number **math_time_expression**)	ISSUE [] NUMBER	Returns a NUMBER [] resulting of evaluating **math_time_expression** against each of the issues in argument **issues**. Argument **math_time_expression** is a math/time expression, where references to field values of issues in argument issues are done with prefix **^** before field code, e.g., **^{...duedate}** is field code for **Due date** in each of the issues in argument **issues**. Example: **`mathOnIssueList(linkedIssues("is blocked by"), (^{...duedate} != null ? ^{...duedate} - ^{...created} : 0) / {HOUR})`** returns a list of numbers with the number of days from issue creation to due date for all issues linked using "**is blocked by**" issue link type.
numberOfRemoteIssueLinks(string **issue_link_types**)	STRING	Returns the NUMBER of issue links to other Jira instances using any of the issue link types in argument **issue_link_types**. Argument **issue_link_types** is a comma separated list of issue link type names, or empty string **`("")`** for representing all issue link types.
count(issue list **l**)	ISSUE []	Returns the NUMBER of issues in **l**. Example: **`count(filterByResolution(linkedIssues("is blocked by"), ""))`** returns the number of non-resolved blocking issues.
getIssuesFromProjects(string **projects**)	STRING	Returns an ISSUE [] with all issues of projects in argument **projects**. Argument **projects** is a string containing a comma separated list of project keys or project names. Example: **`getIssuesFromProjects("CRM, HT")`** returns all issues in project **CRM** and **HT**. This function can make your expression run slowly due to the high number of issues retrieved and needing to be filtered. Using **issuesFromJQL()** for retrieving and filtering issues will make your expression run much faster.
first(issue list **l**)	ISSUE []	Returns an ISSUE [] with the first element in issue list **l**, or an empty list if **l** is an empty list.
last(issue list **l**)	ISSUE []	Returns an ISSUE [] with the last element in issue list **l**, or an empty list if **l** is an empty list.
nthElement(issue list **l**, number **n**)	ISSUE [] NUMBER	Returns an ISSUE [] with the element at position **n** in issue list **l**, where **n >= 1** and **n <= count(l)**. Returns an **empty list** if **n** is greater than the number of elements in **l**.
sublist(issue list **l**, number **indexFrom**, number **indexTo**)	NUMBER	Returns an ISSUE [] with elements in **l** from **indexFrom** index to **indexTo** index. Having **indexFrom >= 1** and **indexFrom <= count(l)** and **indexTo >= 1** and **indexTo <= count(l)** and **indexFrom <= indexTo**.
indexOf(string **issue_key**, issue list **l**)	STRING	Returns the index NUMBER in issue list **l** of issue with key **issue_key**. **Zero** is returned when issue is not found in **l**.
indexOf(issue list **element**, issue list **l**)	ISSUE []	Returns the index NUMBER in issue list **l** of first issue in **element**. **Zero** is returned when first issue in **element** is not found in **l**.
sort(issue list **l**, field **field**, **order**)	ISSUE []	Returns an ISSUE [] with elements in **l** ordered according to values of **field**. Argument **field** has format **{...somefield}** for numeric and date-time fields, **%{...somefield}** for string fields, or **%{...somefield.i}** for cascading select fields. Available orders are **ASC** (for ascending order) and **DESC** (for descending order). Example: **`sort(linkedIssues("is blocked by"), {...duedate}, ASC)`** returns the list of issues blocking current issue, sorted in ascending order by **Due date**.

Examples

Input	Output
`subtasks()`	Returns the list of sub-tasks of the current issue.
`linkedIssues("is blocked by, is caused by")`	Returns the list of issues linked to current one through issue link types "**is blocked by**" and "**is caused by**".
`filterByIssueType (linkedIssues(), "Bug, Incident")`	Returns the list of linked issues with issue type "**Bug**" or "**Incident**".
`filterByPredicate (siblingSubtasks(), %{... resolution} != null)`	Returns the list of sibling sub-tasks (i.e., sub-tasks of same parent as current sub-task) which are not resolved.