**Jira**

**1.** cascadeOption() - Search for issues that match the selected values of a 'cascading select' custom field.

* Find issues where a custom field ("Location") has the value "USA" for the first tier and "New York" for the second tier:

location in cascadeOption("USA","New York")

* Find issues where a custom field ("Location") has the value "USA" for the first tier and any value (or no value) for the second tier:

location in cascadeOption( "USA" )

* Find issues where a custom field ("Location") has the value "USA" for the first tier and no value for the second tier:

location in cascadeOption( "USA" ,none)

* Find issues where a custom field ("Location") has no value for the first tier and no value for the second tier:

location in cascadeOption(none)

* Find issues where a custom field ("Referrer") has the value "none" for the first tier and "none" for the second tier:

referrer in cascadeOption( "\"none\"" , "\"none\"" )

* Find issues where a custom field ("Referrer") has the value "none" for the first tier and no value for the second tier:

referrer in cascadeOption( "\"none\"" ,none)

**2.** componentsLeadByUser() - Find issues in components that are led by a specific user. You can optionally specify a user, or if the user is omitted, the current user (i.e. you) will be used.

* Find open issues in components that are led by you:

component in componentsLeadByUser() AND status = Open

* Find open issues in components that are led by Bill:

component in componentsLeadByUser(bill) AND status = Open

**3.** currentLogin() - Perform searches based on the time at which the current user's session began. See also lastLogin.

* Find issues that have been created during my current session:

created > currentLogin()

**4.** currentUser() - Perform searches based on the currently logged-in user. Note, this function can only be used by logged-in users.

* Find issues that are assigned to me:

assignee = currentUser()

* Find issues that were reported to me but are not assigned to me:

reporter = currentUser() AND (assignee != currentUser() OR assignee is EMPTY)

**5.** earliestUnreleasedVersion() - Perform searches based on the earliest unreleased version (i.e. next version that is due to be released) of a specified project.

* Find issues whose FixVersion is the earliest unreleased version of the ABC project:

fixVersion = earliestUnreleasedVersion(ABC)

**6**. issueHistory() - Find issues that you have recently viewed, i.e. issues that are in the 'Recent Issues' section of the 'Issues' drop-down menu.

* Find issues which I have recently viewed, that are assigned to me:

issue in issueHistory() AND assignee = currentUser()

**7.** issuesWithRemoteLinksByGlobalId() - Perform searches based on issues that are associated with remote links that have any of the specified global ids.

* Find issues that are linked to remote links that have globalId "abc":

issue in issuesWithRemoteLinksByGlobalId(abc)

* Find issues that are linked to remote links that have either globalId "abc" or "def":

issue in issuesWithRemoteLinksByGlobalId(abc, def)

**8.** lastLogin() - Perform searches based on the time at which the current user's previous session began.

* Find issues that have been created during my last session:

created > lastLogin()

**9.** latestReleasedVersion() - Perform searches based on the latest released version (i.e. the most recent version that has been released) of a specified project.

* Find issues whose FixVersion is the latest released version of the ABC project:

fixVersion = latestReleasedVersion(ABC)

* Find issues that relate to the latest released version of the ABC project:

affectedVersion = latestReleasedVersion(ABC) or fixVersion = latestReleasedVersion(ABC)

**10.** linkedIssues() - Searches for issues that are linked to an issue.

* Find issues that are linked to a particular issue:

issue in linkedIssues(ABC-123)

* Find issues that are linked to a particular issue via a particular type of link:

issue in linkedIssues(ABC-123,"is duplicated by")

**11.** membersOf() - Perform searches based on the members of a particular group.

* Find issues where the Assignee is a member of the group "jira-administrators":

assignee in membersOf("jira-administrators")

* Search through multiple groups and a specific user:

reporter in membersOf("jira-administators") or reporter in membersOf("jira-core-users") or reporter=jsmith

* Search for a particular group, but exclude a particular member or members:

assignee in membersOf(QA) and assignee not in ("John Smith","Jill Jones")

* Exclude members of a particular group:

assignee not in membersOf(QA)

**12.** projectsLeadByUser() - Find issues in projects that are led by a specific user. You can optionally specify a user, or if the user is omitted, the current user will be used.

* Find open issues in projects that are led by you:

project in projectsLeadByUser() AND status = Open

* Find open issues in projects that are led by Bill:

project in projectsLeadByUser(bill) AND status = Open

**13.** projectsWhereUserHasPermission() - Find issues in projects where you have a specific permission.

* Find open issues in projects where you have the "Resolve Issues" permission:

project in projectsWhereUserHasPermission("Resolve Issues") AND status = Open

**14.** projectsWhereUserHasRole() - Find issues in projects where you have a specific role. Note, this function is only available if you are logged in to Jira.

* Find open issues in projects where you have the "Developers" role:

project in projectsWhereUserHasRole("Developers") AND status = Open

**15.** releasedVersions() - Perform searches based on the released versions (i.e. versions that your Jira administrator has released) of a specified project. You can also search on the released versions of all projects, by omitting the project parameter.

* Find issues whose FixVersion is a released version of the ABC project:
  + fixVersion in releasedVersions(ABC)
* Find issues that relate to released versions of the ABC project:
  + (affectedVersion in releasedVersions(ABC)) or (fixVersion in releasedVersions(ABC))

**16.** standardIssueTypes() - Perform searches based on "standard" Issue Types, that is, search for issues that are not sub-tasks.

* Find issues that are not subtasks (i.e. issues whose Issue Type is a standard issue type, not a subtask issue type):

issuetype in standardIssueTypes()

**17.** subtaskIssueTypes() - Perform searches based on issues that are sub-tasks.

* Find issues that are subtasks (i.e. issues whose Issue Type is a subtask issue type):

issuetype in subtaskIssueTypes()

**18.** unreleasedVersions() - Perform searches based on the unreleased versions (i.e. versions that your Jira administrator has not yet released) of a specified project. You can also search on the unreleased versions of all projects, by omitting the project parameter.

* Find issues whose FixVersion is an unreleased version of the ABC project:

fixVersion in unreleasedVersions(ABC)

* Find issues that relate to unreleased versions of the ABC project:

affectedVersion in unreleasedVersions(ABC)

**20**. votedIssues() - Perform searches based on issues for which you have voted.

* Find issues that you have voted for:

issue in votedIssues()

**21.** watchedIssues() - Perform searches based on issues that you are watching.

Find issues that you are watching:

issue in watchedIssues()

**22.** endOfDay()

* Find issues due by the end of today:

due < endOfDay()

* Find issues due by the end of tomorrow:

due < endOfDay("+1")

**23.** endOfMonth()

* Find issues due by the end of this month:
  + due < endOfMonth()
* Find issues due by the end of next month:
  + due < endOfMonth("+1")
* Find issues due by the 15th of next month:
  + due < endOfMonth("+15d")

**24.** endOfWeek()

* Find issues due by the end of this week:
  + due < endOfWeek()
* Find issues due by the end of next week:
  + due < endOfWeek("+1")

**25.** endOfYear()

* Find issues due by the end of this year:

due < endOfYear()

* Find issues due by the end of March next year:

due < endOfYear("+3M")

**26**. now() - Perform searches based on the current time.

* Find issues that are overdue:

duedate < now() and status not in (closed, resolved)

**27**. startOfDay() - Perform searches based on the start of the current day.

* Find new issues created since the start of today:

created > startOfDay()

* Find new issues created since the start of yesterday:

created > startOfDay("-1")

* Find new issues created in the last three days:

created > startOfDay("-3d")

**28**. startOfMonth() - Perform searches based on the start of the current month.

* Find new issues created since the start of this month:

created > startOfMonth()

* Find new issues created since the start of last month:

created > startOfMonth("-1")

* Find new issues created since the 15th of this month:

created > startOfMonth("+14d")

**29**. startOfWeek() - Perform searches based on the start of the current week.

* Find new issues since the start of this week:

created > startOfWeek()

* Find new issues since the start of last week:

created > startOfWeek("-1")

**30.** startOfYear() - Perform searches based on the start of the current year.

* Find new issues since the start of this year:

created > startOfYear()

* Find new issues since the start of last year:

created > startOfYear("-1")

**31.** votedIssues() - Perform searches based on issues for which you have voted.

* Find issues that you have voted for:

issue in votedIssues()

**32.** watchedIssues() - Perform searches based on issues that you are watching.

* Find issues that you are watching:

issue in watchedIssues()

**Jira Software**

**1.** openSprints() - Search for issues that are assigned to a sprint that was started, but has not yet been completed. NOTE: It is possible for an issue to belong to both a completed sprint(s) and an incomplete sprint(s).

* Find all issues that are assigned to a sprint that has not yet been completed:

sprint in openSprints()

**2.** closedSprints() - Search for issues that are assigned to a completed Sprint. Note, it is possible for an issue to belong to both a completed Sprint(s) and an incomplete Sprint(s).

* Find all issues that are assigned to a completed sprint:

sprint in closedSprints()

**3.** futureSprints() - Search for issues that are assigned to a sprint that hasn't been started yet. NOTE: It is possible for an issue to belong to both a completed sprint(s) and an incomplete sprint(s).

* Find all issues that are assigned to a sprint that hasn't been started yet:

sprint in futureSprints()

**4.** parentEpic() - Search for issues and sub-tasks that are linked to an epic.

* Find issues and sub-tasks in the epic DEMO-123:

parentEpic = DEMO 123

* Find issues and sub-tasks the epic DEMO-1 or SAMPLE-4:

parentEpic in (DEMO-1, SAMPLE-4)

**Jira Service Desk**

**1.** approved() - Search for requests that required approval and have a final decision of approved.

* Find all requests that have been approved:

approval = approved()

**2.** approver() - Search for requests that require or required approval by a user.

* Find requests that require or required approval by John Smith:

approval = approver(jsmith)

* Find requests that require or required approval by John Smith or Sarah Khan:

approval = approver(jsmith,skhan)

**3**. breached() - Returns issues that whose most recent SLA has missed its goal.

* Find issues where Time to First Response was breached:

"Time to First Response" = breached()

**4.** completed() - Returns issues that have an SLA that has completed at least one cycle.

* Find issues where Time to First Response has completed at least one cycle:

"Time to First Response" = completed()

**5.** elapsed() - Returns issues whose SLA clock is at a certain point relative to a cycle's start event.

* Find issues that have been waiting for a first response for more than 1 hour:

"Time to First Response" > elapsed("1h")

**6**. everbreached() - Returns issues that have missed one of their SLA goals.

* Find issues have missed their goal for Time to First Response:

"Time to First Response" = everbreached()

**7**. myApproval() - Search for requests that require approval or have required approval by the current user.

* Find all requests that require or have required my approval

approval = myApproval()

**8.** myPending() - Search for requests that require approval by the current user.

* Find all requests that require my approval

approval = myPending()

**9.** organizationMembers() - Search for all requests sent by the members of an organization.

* Find all requests sent by members of the organization Atlassian:

reporter in organizationMembers("Atlassian")

* Find requests sent by people who are not in the organizations Atlassian or ACME:

reporter not in organizationMembers("Atlassian","ACME")

**10.** paused() - Returns issues that have an SLA that is paused due to a condition.

* Find issues where Time to First Response is paused:

"Time to First Response" = paused()

**11**. pending() - Search for requests that require approval.

* Find all requests that require approval:

approval = pending()

**12.** pendingBy() - Search for requests that require approval by a certain user. This uses an OR operator, and you must specify a username.

* Find requests that require approval by John Smith:

approval = pendingBy(jsmith)

* Find requests that require by John Smith or Sarah Khan:

approval = pendingBy(jsmith,skhan)

**13**.remaining() - Returns issues whose SLA clock is at a certain point relative to the goal.

* Find issues that will breach Time to Resolution in the next two hours:

"Time to Resolution" < remaining("2h")

**14.** running() - Returns issues that haxve an SLA that is running, regardless of the calendar.

* Find issues where Time to First Response is running:

"Time to First Response" = running()

**15.** withinCalendarHours() - Returns issues that have an SLA that is running according to the SLA calendar.

* Find issues where Time to First Response is within calendar hours:

"Time to First Response" = withinCalendarHours()