Examples of Boolean expressions

Boolean expressions are **logical constructions** that return **true** or **false**, and are used for implementing **conditions**, **validations**, and **conditional executed post-functions**.

Simple Field Value Checking

Boolean Expression	What is it for?
`%{...priority} = "Critical"`	Validates that **Priority** has value **Critical**.
`%{...priority} != "Blocker"`	Validates that **Priority** has a value different from **Blocker**.

`%{...priority} in ["Blocker", "Critical"]`
`or alternatively`
`%{...priority} = "Blocker" OR %{...priority} = "Critical"`
	Validates that **Priority** has value **Blocker** or **Critical**.

`%{...priority} != null`	Validates that **Priority** is set.
`{...timeoriginalestimate} > 60`	Validates that **Original estimate (minutes)** is greater than 60.
`%{...issuetype} = "Bug" IMPLIES %{...versions} != null`	Validates that **Affects version/s** is set whenever issue type is **Bug**.
`%{...priority} in ["Blocker", "Critical", "Major"] IMPLIES (%{...assignee} != null AND {...duedate} != null)`	Validates that if **Priority** is **Blocker**, **Critical** or **Major** then issue must be assigned and **Due date** must be set.
`length(%{...description}) >= 10`	Validates that **Description** contains at least 10 characters.
`{...cf13700} in [0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]`	Validates that a numeric custom field called "**Story Points**" has been given a value in Fibonacci sequence. {...cf13700} is code for numeric value of custom field "**Story Points**". This code depends on each particular Jira instance.
`{...cf10000} + {...cf10001} + {...cf10003} > 10`	Validates that 3 numerical custom fields sum a value higher than 10. This boolean expression assumes that all 3 fields are initialized. {...cf10000}, {...cf10001} and {...cf10003} are field codes of 3 supposed numerical custom fields. Custom field codes depend on each particular Jira instance. **Warning**: This boolean expression fails when any of the fields is not initialized.
`sum([{...cf10000}, {...cf10001}, {...cf10003}]) > 10`	Validates that 3 numerical custom fields sum a value higher than 10. If a field is not initialized it's assumed a zero value.

Time Related

Boolean Expression	What is it for?
`{...duedate} >= datePart({...currentDateTime}, LOCAL) + 5 * {DAY}`	Validates that it's at least 5 days left for **Due date**.

On this page

`datePart({...duedate}, LOCAL) >= datePart({...currentDateTime}, LOCAL)`	Validates that due date has not yet expired.
`timePart({...currentDateTime}, LOCAL) >= 8:00 AND timePart({...currentDateTime}, LOCAL) <= 20:30`	Checks whether **Current Time** is between 8:00 and 20:30 in server's timezone.
`dayOfTheWeek({...duedate}, USER_LOCAL) >= {MONDAY} AND dayOfTheWeek({...duedate}, USER_LOCAL) <= {FRIDAY}`	Checks if **Due Date** is between Monday and Friday in currently logged user's timezone.
`{...currentDateTime} <= {...created} + 10 * {DAY} + 5 * {HOUR} + 45 * {MINUTE}`	Checks whether we are still within 10 days, 5 hours and 45 minutes since **date and time of issue creation**.
`{...currentDateTime} <= addTimeSkippingWeekends({...created}, 10 * {DAY} + 5 * {HOUR} + 45 * {MINUTE}, LOCAL)`	Checks whether we are still within 10 days, 5 hours and 45 minutes since **date and time of issue creation**, skipping the periods of time which correspond to weekend in server's local timezone.
`timeDifference({...currentDateTime}, {...created}, "my_schedule", LOCAL) <= 10 * {DAY} + 5 * {HOUR} + 45 * {MINUTE}`	Checks whether we are still within 10 days, 5 hours and 45 minutes since **date and time of issue creation** considering **custom schedule** called `my_schedule`.
`timeDifference({...duedate}, {...currentDateTime}, "my_schedule", LOCAL) > 48 * {HOUR}`	Validates that due date is at least 48 hours in the future considering **custom schedule** called my_schedule.
`{...duedate} != null IMPLIES {...duedate} >= ({...created} + 2 * {DAY})`	Validates that if **Due Date** is set, then it must be at least 2 days later than **Day and Time of Creation**.

On this page

Multi-Valued Fields (Versions, Components, Labels, Multi-Select List, Checkboxes, etc.)

Boolean Expression	What is it for?
`"Component A" in %{...components}`	Checks that component called **Component A** is contained in current issues **Components**.
`"Component A" not in %{...components}`	Checks that component called **Component A** is not contained in current issues **Components** .
`"web" in %{...labels} AND "mobile" in %{...labels}`	Validates that current issue contains both labels "**web**" and "**mobile**"
`numberOfSelectedItems(%{...components}) = 1`	Validation to check that **Components** has only 1 element selected.
`numberOfSelectedItems(%{...components}) < numberOfAvailableItems(%{...components})`	Validates that not all the components are selected.
`%{...versions} none in %{...fixVersions}`	Checks that there isn't any version in common between **Affect version/s** and **Fix version/s**
`%{...versions} any in %{...fixVersions}`	Checks that there is at least one version in common between **Affect version/s** and **Fix version/s**

Linked Issues, Sub-tasks and Epic-Story relations

Expression	What is it for?
`count(linkedIssues("is Epic of")) >= 5 AND count(subtasks()) >= 3`	Validates that current issue (which is an Epic) has at least 5 tasks and 3 sub-tasks.

`count(filterByPredicate(subtasks(), %{...components} not in ^%{...components})) = 0`	Validates that all the components selected in current issue (parent issue) are also selected in each of its sub-tasks.
`%{...versions} in fieldValue(%{...versions}, subtasks())`	Validates that each affected version in current issue (parent issue) is selected at least in one of its sub-tasks.
`{...duedate} > max(fieldValue({...duedate}, linkedIssues("is blocked by") UNION subtasks()))`	Validates tha **Due Date** is greater than latest **Due Date** among blocking issues and sub-tasks
`count(filterByPredicate (filterByIssueType(linkedIssues("is Epic of"), "Story, Bug"), ^%{...resolution} != null OR ^{...duedate} = null)) = 0`	Validates that all the stories and bugs of an Epic have fields **Due date** set and **Resolution** unset.
`count(filterByPredicate(linkedIssues ("is Epic of") UNION subtasks(), ^%{...resolution} = null)) = 0`	Validates that all the tasks and sub-tasks of an Epic are resolved.
`(%{...issuetype} in ["Sub-task"] IMPLIES %{...parentIssueType} in ["Task", "New Feature", "Enhancement"]) AND (%{...issuetype} in ["Agile Sub-task"] IMPLIES %{...parentIssueType} in ["Story", "Epic"])`	Validation for **limiting the type of sub-task** that can be created **by certain parent issue types**. Current example limits creation of "**Sub-task**" sub-task type to parent issues "**Task**", "**New Feature**" and "**Enhancement**", and "**Agile Sub-task**" sub-task type to parent issues "**Story**" and "**Epic**".
`%{...parentIssueKey} != null IMPLIES %{...parentIssueStatus} != "Closed"`	Validates that parent issue is not closed during the creation of a sub-task.
`count(siblingSubtasks()) = count (filterByStatus(siblingSubtasks(), "Resolved, Closed"))`	Validates that all the sibling sub-tasks are in **Resolved** or **Closed** status. Sibling sub-tasks are those sub-tasks sharing the same parent issue as current sub-task.
`count(linkedIssues("is Epic of", %{...parentIssueKey})) - 1 <= count (filterByStatus(linkedIssues("is Epic of", %{...parentIssueKey}), "Resolved, Closed"))`	Validates that all the sibling stories are in **Resolved** or **Closed** status. Sibling stories are those issues sharing the same epic as current issue, regardless of their issue type. We are also considering that current issue may not yet be in **Closed** or **Resolved** status, since this validation is intended to be used in transitions like "**Resolve Issue**" and "**Close Issue**".
`%{...resolution} = "Duplicate" IMPLIES count(linkedIssues ("duplicates")) > 0`	Validates that issues being resolved as "**Duplicate**" has at least a "**duplicates**" issue link.
`count(filterByPredicate(subtasks(), ^%{...status} != "Done" AND ^%{...priority} in ["Critical", "Major"])) = 0`	Validates that sub-tasks with priority **Critical** or **Major** are in status **Done**.

Versions

Expression	What is it for?
`toStringList(%{...fixVersions}) in releasedVersions()`	Validates that all fixed versions are released.
`toStringList(%{...versions}) any in unreleasedVersions()`	Validates that at least one affected version is unreleased.
`latestReleasedVersion() in archivedVersions()`	Validates that the latest released version in current issue's project is archived.
`earliestUnreleasedVersion() not in archivedVersions()`	Validates that earliest unreleased version in current issue's project is not archived.

On this page

Attachments

Expression	What is it for?
`matches(%{...attachmentsWithDetails}, "(.*(image/jpeg|image/png).*){3,}") AND matches(%{...attachmentsWithDetails}, "(.*(text/plain|application/pdf).*){2}")`	Requires that the issue has attached at least **3 images in format jpg or png**, and exactly **2 documents in formats pdf or txt**.
`count(toStringList(%{...attachments})) = count(distinct(toStringList(%{...attachments})))`	Validation for rejecting **repeated file names** in attachments.

On this page