Boolean Expression examples

On this page

Boolean Expression Examples

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?	Notes
`%{00017} = "Critical"`	Validates that **Priority** has value **Critical**.	`%{00017}` = **Priority**
`%{00017} != "Blocker"`	Validates that **Priority** has a value different from **Blocker**.	`%{00017}` = **Priority**

`%{00017} in ["Blocker", "Critical"]`
or alternatively
`%{00017} = "Blocker" OR %{00017} = "Critical"`
	Validates that **Priority** has value **Blocker** or **Critical**.	`%{00017}` = **Priority**

`%{00017} != null`	Validates that **Priority** is set.	`%{00017}` = **Priority**
`{00068} > 60`	Validates that **Original estimate (minutes)** is greater than 60.	`{00068}` = numeric value for **Original estimate (minutes)**
`%{00014} = "Bug" IMPLIES %{00077} != null`	Validates that **Affects version/s** is set whenever issue type is **Bug**.	`%{00077}` = **Affects version/s** `%{00014}` = **Issue type**
`%{00017} in ["Blocker", "Critical", "Major"] IMPLIES (%{00003} != null AND {00012} != null)`	Validates that if **Priority** is **Blocker**, **Critical** or **Major** then issue must be assigned and **Due date** must be set.	`%{00003}` = **Assignee** `{00012` = **Due date**
`length(%{00001}) >= 10`	Validates that **Description** contains at least 10 characters.	`%{00001}` = **Description**
`{13700} 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.	`{13700}` is code for numeric value of custom field **"Story Points"**. This code depends on each particular Jira instance. **Example**
`{10000} + {10001} + {10003} > 10`	Validates that 3 numerical custom fields sum a value higher than 10. This boolean expression assumes that all 3 fields are initialized.	`{10000}`, `{10001}` and `{10003}` 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([{10000}, {10001}, {10003}]) > 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.	`{10000}`, `{10001}` and `{10003}` are field codes of 3 supposed numerical custom fields. Custom field codes depend on each particular Jira instance. This boolean expression also works when some of the fields are not initialized.

Time Related

Boolean Expression	What is it for?	Notes
`{00012} >= datePart({00057}, LOCAL) + 5 * {DAY}`	Validates that it's at least 5 days left for *Due date*.	**`{00057}` = Current date and time `{00012}` = Due Date**
`datePart({00012}, LOCAL) >= datePart ({00057}, LOCAL)`	Validates that due date has not yet expired.	**`{00012}` = Due Date `{00057}` = Current day and time**
`timePart({00057}, LOCAL) >= 8:00 AND timePart({00057}, LOCAL) <= 20:30`	Checks whether **Current Time** is between 8:00 and 20:30 in server's timezone.	**`{00057}` = Current day and time**
`dayOfTheWeek({00012}, USER_LOCAL) >= {MONDAY} AND dayOfTheWeek({00012}, USER_LOCAL) <= {FRIDAY}`	Checks if **Due Date** is between Monday and Friday in currently logged user's timezone.	**`{00012}` = Due Date**
`{00057} <= {00009} + 10 * {DAY} + 5 * {HOUR} + 45 * {MINUTE}`	Checks whether we are still within 10 days, 5 hours and 45 minutes since **da te and time of issue creation**.	**`{00057}` = Current date and time `{00009}` = Date and time of creation**
`{00057} <= addTimeSkippingWeekends ({00009}, 10 * {DAY} + 5 * {HOUR} + 45 * {MINUTE}, LOCAL)`	Checks whether we are still within 10 days, 5 hours and 45 minutes since **da te and time of issue creation**, skipping the periods of time which correspond to weekend in server's local timezone.	**`{00057}` = Current date and time `{00009}` = Date and time of creation**
`timeDifference({00057}, {00009}, "my_schedule", LOCAL) <= 10 * {DAY} + 5 * {HOUR} + 45 * {MINUTE}`	Checks whether we are still within 10 days, 5 hours and 45 minutes since **da te and time of issue creation** considering **custom schedule** called **`my_schedule`**.	**`{00057}` = Current date and time `{00009}` = Date and time of creation**
`timeDifference({00012}, {00057}, "my_schedule", LOCAL) > 48 * {HOUR}`	Validates that due date is at least 48 hours in the future considering **custom schedule** called **`my_schedule`**.	**`{00057}` = Current date and time `{00012}` = Due date**
`{00012} != null IMPLIES {00012} >= ({00009} + 2 * {DAY})`	Validates that if **Due Date** is set, then it must be at least 2 days later than **Da y and Time of Creation**.	**`{00012}` = Due Date `{00009}` = Date and time of creation**

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

Boolean Expression	What is it for?	Notes

`"Component A" in %{00094}`	Checks that component called **Component A** is contained in current issues **Components**.	**%{00094} = Components**
`"Component A" not in %{00094}`	Checks that component called **Component A** is not contained in current issues **Components.**	**%{00094} = Components**
`"web" in %{00080} AND "mobile" in %{00080}`	Validates that current issue contains both labels **"web"** and **"mobile"**	**%{00080} = Labels**
`numberOfSelectedItems(%{00094}) = 1`	Validation to check that **Components** has only 1 element selected.	**%{00094} = Components**
`numberOfSelectedItems(%{00094}) < numberOfAvailableItems(%{00094})`	Validates that not all the components are selected.	**%{00094} = Components**
`%{00077} none in %{00074}`	Checks that there isn't any version in common between **Affect version/s** and **Fix version/s**	**%{00077} = Affected versions** **%{00074} = Fixed versions**
`%{00077} any in %{00074}`	Checks that there is at least one version in common between **Affect version/s** and **Fix version/s**	**%{00077} = Affected versions** **%{00074} = Fixed versions**

Linked Issues, Sub-tasks and Epic-Story relations

Expression	What is it for?	Notes
`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(), %{00094} not in ^%{00094})) = 0`	Validates that all the components selected in current issue (parent issue) are also selected in each of its sub-tasks.	**%{00094} = Components**
`%{00077} in fieldValue(%{00077}, subtasks())`	Validates that each affected version in current issue (parent issue) is selected at least in one of its sub-tasks.	**%{00094} = Affected versions**
`{00012} > max(fieldValue ({00012}, linkedIssues("is blocked by") UNION subtasks()))`	Validates tha **Due Date** is greater than latest **Due Date** among blocking issues and sub-tasks	**{00012} = Due date**
`count(filterByPredicate (filterByIssueType(linkedIssues ("is Epic of"), "Story, Bug"), ^%{00028} != null OR ^{00012} = null)) = 0`	Validates that all the stories and bugs of an Epic have fields **Due date** set and **Resolution** unset.	**^%{00028} = Resolution in foreign issues** **^{00012} = Due Date in foreign issues**
`count(filterByPredicate (linkedIssues("is Epic of") UNION subtasks(), ^%{00028} = null)) = 0`	Validates that all the tasks and sub-tasks of an Epic are resolved.	**%{00028} = Resolution**
`(%{00014} in ["Sub-task"] IMPLIES %{00040} in ["Task", "New Feature", "Enhancement"]) AND (%{00014} in ["Agile Sub-task"] IMPLIES %{00040} in ["Story", "Epic"])`	Validation for **limiting the type of subtask** 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"**.	This validator should be added to transition "**Create Issue**" of sub-task's workflow. **%{00014} = Issue type** **%{00040} = Parent's issue type**
`%{00041} != null IMPLIES %{00042} != "Closed"`	Validates that parent issue is not closed during the creation of a sub-task.	This validation must be added in transition "**Create Issue**" of sub-task's workflow. **%{00041} = Parent's issue key** **%{00042} = Parent's issue status**

`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", %{00041})) - 1 <= count (filterByStatus(linkedIssues ("is Epic of", %{00041}), "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**". **%{00041} = Parent's issue key**
`%{00028} = "Duplicate" IMPLIES count(linkedIssues ("duplicates")) > 0`	Validates that issues being resolved as **"Duplicate"** has at least a **"duplicates"** issue link.	**%{00028 = Resolution**
`count(filterByPredicate (subtasks(), ^%{00016} != "Done" AND ^%{00017} in ["Critical", "Major"])) = 0`	Validates that sub-tasks with priority **Critical** or **Major** are in status **Done**.	**%{00016} = Status** **%{00017} = Priority**
`%{00028} = "Duplicate" IMPLIES count(filterByPredicate (linkedIssues("duplicates"), ^%{00028} = "Duplicate" OR ^%{00018} != %{00018})) = 0`	Validates that issues resolved as **"Duplicate"** can contain **"duplicates"** issue links only to issues in the **same project** which are not also resolved as **"Duplicate"**.	**^%{00028} = Resolution in foreign issues** **^%{00018} = Project key in foreign issues** **%{00018} = Project key in current issue**

Versions

Expression	What is it for?	Notes
`toStringList(%{00074}) in releasedVersions()`	Validates that all fixed versions are released.	Requires version 2.1.32 or higher. **%{00074} = Fix version/s**
`toStringList(%{00077}) any in unreleasedVersions()`	Validates that at least one affected version is unreleased.	Requires version 2.1.32 or higher. **%{00077} = Affect version/s**
`latestReleasedVersion() in archivedVersions()`	Validates that the latest released version in current issue's project is archived.	Requires version 2.1.32 or higher.
`earliestUnreleasedVersion() not in archivedVersions()`	Validates that earliest unreleased version in current issue's project is not archived.	Requires version 2.1.32 or higher.
`toStringList(^%{00074}) in releasedVersions (^%{00018})`	Validates that all fixed versions of a foreign issue are released.	**%{00074} = Fixed versions** **%{00018} = Project key** Foreign issues appear in "**Filtering by field values**" parameter in **Block or hide a transition for an issue depending on its issue links**, **Condition and validation on sub-tasks**, **Write field on linked issues or sub-tasks**, etc., and also in **"filterByPredicate()"** parser function.

Attachments

Expression	What is it for?	Notes
`{00135} > 0`	Validates that at least a file has been attached in current transition.	**{00135} = Number of transition attachments**

`(%{00161} ~ "image/png" OR %{00161} ~ "image/jpeg") AND %{00161} ~ "application/pdf"`	Requires that the user attaches in current transition **at least one image** in **format jpg or png**, and **at least a pdf** file.	**`%{00161}` = Transition's attachments with details**
`matches(%{00072}, "(.*(image/jpeg|image/png).*){3,}") AND matches(%{00072}, "(.*(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**.	**`%{00072}` = Attachments with details**
`count(toStringList(%{00071})) = count(distinct(toStringList(%{00071})))`	Validation for rejecting **repeated file names** in attachments.	**`%{00071}` = Attachments** Example
`toLowerCase(%{00001}) ~ "please, attach" IMPLIES %{00160} != null`	Requires at least one attachment in transition screen if issue Description contains sub-string **"please, attach"**.	**`%{00160}` = Transition's attachments**
`count(findPatternIgnoreCase(%{00001}, "please,?\\s*attach")) > 0 IMPLIES %{00160} != null`	Same as previous validation, but making the comma character optional, and also allowing any number of whitespaces.	**`%{00160}` = Transition's attachments**

Comments

Expression	What is it for?	Notes
`%{00127} != null`	Validates that a comment has been entered in transition screen.	**`%{00127}` = Transition's comment**
`%{00127} != null OR isBulkTriggeredTransition()`	Validates that a comment has been entered in transition screen, except when issue is transitioned by a **bulk update operation**.	Requires version **2.2.12** or higher. **`%{00127}` = Transition's comment**
`matches(%{00127}, "((?s).*\\w\\b){3}")`	Requires a **comment with at least 3 words** to be introduced in transition screen	**`%{00127}` = Transition's comment** You can modify the number of required words simply by editing the number in the regular expression.