Examples of Parser expressions

On this page

Examples of Parser expressions

This page presents a collection of expressions valid for the **Expression Parser**.

Text Composition and Format

Expression	Example of Returned Value	Notes
`"Current issue was reported on " + %{00009} + " by " + %{00005} + "."`	Current issue was reported on 2014-09-03 19:28 by John Nash.	`%{00009}` **= Date and time of creation** `%{00005}` **= Reporter's full name**
`"Today is " + dayOfTheWeekToString({00057}, USER_LOCAL, USER_LANG) + "."`	Today is Monday.	Tells current day of the way in users local time zone and language. `{00057}` **= Current day and time**
`"Number of hours since issue creation: " + round(({00057} - {00009}) / {HOUR}) + " hours."`	Number of hours since issue creation: 75 hours.	`{00057}` **= Current day and time** `%{00009}` **= Date and time of creation**
`"Number of days to due date: " + floor(({00012} - {00057}) / {DAY}) + " days."`	Number of days to due date: 2 days.	`{00012}` **= Due Date** `{00057}` **= Current day and time**

Math Calculus

Expression	Returned Value	Notes
`max(count(subtasks(%{00041})) - 1, 0)` or since version 2.2.1: `count(siblingSubtasks())`	For a sub-task, the number of sibling sub-tasks.	Function `max(x, y)` is used to avoid returning `-1` when used with non-sub-task issues. `%{00041}` **= Parent's issue key**
`{10000} = null ? 1 : {10000} + 1`	Formula to increment a numeric custom field, setting it to `1` if it's initially unset.	`{10000}` is the field code for a supposed numeric custom field.
`{10000} + {10001} + {10003}`	Formula for summing 3 numeric custom fields when we are certain that **all 3 the fields are initialized**. In case any of these fields is not initialized, an error is raised and any of the following 2 expression examples should be used.	`{10000}`, `{10001}` and `{10003}` are three numeric custom field.
`({10000} = null ? 0 : {10000}) + ({10001} = null ? 0 : {10001}) + ({10003} = null ? 0 : {10003})`	Formula for summing 3 numeric custom fields when some of them **may be uninitialized**. When any of this fields is not initialized a zero value is assumed.	`{10000}`, `{10001}` and `{10003}` are three numeric custom field.

`sum([{10000}, {10001}, {10003}])`	A more compact syntax for summing 3 numeric custom fields when some of them **may be uninitialized**. Version **2.2.16** or higher is required.	**`{10000}`**, **`{10001}`** and **`{10003}`** are three numeric custom field. This syntax is available since version **2.2.16**.

Date-Time Calculus

Expression	Returned Value	Notes
`{00012} - 6 * {DAY}`	Calculates a date 6 natural days earlier than Due Date	**`{00012}` = Due Date**
`addTimeSkippingWeekends({00009}, 36*{HOUR} + 45*{MINUTE}, LOCAL)`	Returns a date-time value equivalent to adding 36 hour and 45 minutes to *date and time of issue creation*, skipping the periods of time which correspond to weekend.	**`{00009}` = Date and time of creation**
`addTimeSkippingWeekends({00009}, 36*{HOUR} + 45*{MINUTE}, LOCAL, {FRIDAY}, {SATURDAY})`	Same as previous expression, but using Israeli weekend.	Israeli weekend is on Friday and Saturday.
`addDaysSkippingWeekends({00012}, -6, LOCAL)`	Calculates a date 6 work days earlier than Due Date for Jira Server's local timezone.	**`{00012}` = Due Date** Work days depend on timezone, since certain time moment maybe Sunday in certain time zones, and Monday in another ones.
`subtractDatesSkippingWeekends({00012}, {00057}, LOCAL)/{DAY}`	Returns the number of working days from *Current Date and Time* to *Due Date*, i.e., skipping weekends in Jira server's timezone.	**`{00012}` = Due Date** **`{00057}` = Current day and time**
`round(({00057} - {00009}) / {HOUR})`	Number of hours since issue creation	Function **`round()`** approximates the number of hours to the nearer integer. **`{00057}` = Current day and time** **`%{00009}` = Date and time of creation**
`floor(({00012} - {00057}) / {DAY})`	Number of days to Due Date	Function **`floor()`** approximates the number of days by removing the fractional part. **`{00012}` = Due Date** **`{00057}` = Current day and time**
`datePart({00057}, LOCAL) + (dayOfTheWeek({00057}, LOCAL) = 7 ? 6 : 6 - dayOfTheWeek({00057}, LOCAL)) * {DAY}`	Returns a date value for **next Friday**, or for today if it's Friday	**`{00057}` = Current day and time** **Example**
`datePart({00057}, LOCAL) + (dayOfTheWeek({00057}, LOCAL) = 6 ? 7 : (dayOfTheWeek({00057}, LOCAL) = 7 ? 6 : 6 - dayOfTheWeek({00057}, LOCAL))) * {DAY}`	Returns a date value for **next Friday**, even if today is Friday.	**`{00057}` = Current day and time** **Example**
`floor(subtractDatesSkippingWeekends({00057}, {00009}, LOCAL) / {DAY}) + " days " + floor(modulus (subtractDatesSkippingWeekends({00057}, {00009}, LOCAL), {DAY}) / {HOUR}) + " hours " + round(modulus (subtractDatesSkippingWeekends({00057}, {00009}, LOCAL), {HOUR}) / {MINUTE}) + " minutes"`	Calculates the time since issue creation skipping weekends, and shows it as a text like this: **12 days 6 hours 34 minutes**.	**`{00057}` = Current day and time** **`%{00009}` = Date and time of creation**

`floor(({00057} - {00009}) / {DAY}) + " days " + floor(modulus(({00057} - {00009}), {DAY}) / {HOUR}) + " hours " + round(modulus(({00057} - {00009}), {HOUR}) / {MINUTE}) + " minutes"`	Calculates the time since issue creation, and shows it as a text like this: **12 days 6 hours 34 minutes**.	**`{00057}` = Current day and time `%{00009}` = Date and time of creation**

Issue Selection

Expression	Returned Value	Notes
`filterByFieldValue(subtasks(), %{00094}, ~ , "Component A")` or alternatively **`filterByPredicate(subtasks(), %{00094} ~ "Component A")`**	Returns an **issue list** with sub-tasks having "**Component A**" among its components.	**`%{00094}` = Components**
`except(subtasks(%{00041}), issueKeysToIssueList(%{00015}))` or alternatively **`filterByPredicate(subtasks(%{00041}), ^%{00015} != %{00015})`**	Returns an **issue list** with sibling sub-tasks, i.e., parent's sub-tasks except current issue.	**`%{00041}` = Parent's issue key** **`%{00015}` = Issue key**
`filterByFieldValue(filterByIssueType(getIssuesFromProjects(%{00018}), %{00014}), %{00000}, =, %{00000})` or alternatively **`filterByPredicate(getIssuesFromProjects(%{00018}), ^%{00014} = %{00014} AND ^%{00000} = %{00000})`**	Returns an **issue list** with all issues in the same project as current issue, with same issue type and same summary.	Might be used in combination with function **`count()`** for creating a validation to avoid issue creation when an issue with same summary already exists in the project and issue type. **`%{00018}` = Project key** **`%{00014}` = Issue type** **`%{00000}` = Summary**

Working with Fields in Linked Issues and Sub-tasks

Expression	Returned Value	Notes
`filterByCardinality(fieldValue(%{00094}, subtasks()), =, count(subtasks()))`	`["Component A", "Component B", "Component C"]`	Returns a **string list** with the **Components** present in all sub-tasks of current issue, i.e., those components common to all sub-tasks. **`%{00094}` = Components**
`{00012} > max(fieldValue({00012}, union(linkedIssues("is blocked by"), subtasks())))`	Validation to check that: **Due Date** is greater than latest **Due Date** among blocking issues and sub-tasks.	Function **max(number_list)** is available since version 2.1.22 **`{00012}` = Due Date**
`count(filterByFieldValue(subtasks(), %{00070}, =, "") UNION filterByFieldValue(subtasks(), %{00012}, =, "")) = 0` or alternatively **`count(filterByPredicate(subtasks(), ^%{00070} = null OR ^%{00012} = null)) = 0`**	Expression for checking whether all sub-tasks of current issue have fields **Due date** and **Environment** set.	**`%{00012}` = Due date** **`%{00070}` = Environment**
`count(filterByPredicate(linkedIssues("is Epic of"), ^%{00028} != null OR ^{00012} = null)) = 0`	This validation allows certain transition in **Epic's workflow** to be executed, only if all the **tasks are unresolved** and have **Due Date set**.	**`^%{00028}` = Resolution in foreign issues** **`^{00012}` = Due Date in foreign issues.** **Example**

Logical Constructions

Expression	Returned Value	Notes
`!(%{00017} = "Blocker" OR %{00017} = "Critical") OR {00012} != null`	Validation for checking that: If **Priority** is "**Blocker**" or "**Critical**" then **Due Date** must be initialized.	It is based on equivalent logical constructions: **A implies B = !A OR B** **`%{00017}` = Priority** **`{00012}` = Due Date**
`%{00017} = "Blocker" OR %{00017} = "Critical" IMPLIES {00012} != null`	Validation for checking that: If **Priority** is "**Blocker**" or "**Critical**" then **Due Date** must be initialized.	Same as former example but using logical connective **IMPLIES**, which is available since version **2.1.22**. **`%{00017}` = Priority** **`{00012}` = Due Date**
`{00012} = null OR {00012} >= ({00009} + 2 * {DAY})`	Validation for checking: If **Due Date** is set then it must be equal or grater than **Day and Time of Creation** plus 2 days.	`{00012}` = Due Date `{00009}` = Date and time of creation

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**.