jiraExpression()

This function returns the result of a given Jira expression as a text. In case a number, a list or a logical value is returned by the Jira expression, the returned text of the function jiraExpression() can be converted accordingly, e.g. with toStringList().

It unfolds its power when combined with JWT functionality which is not available in Jira expressions and vice versa.

Syntax
`jiraExpression(expression) #Output: Text`

Examples

Parser expression	Description
`%{sort(distinct(toStringList(jiraExpression("issue.subtasks.map(s=>s.components.map(c=>c.name))"))),ASC)}`	This example returns an ordered list of the unique component names of all sub-tasks of an issue, e.g. **Backend, Frontend, UI** To convert to text result, the following function is used: • toStringList()
`%{Hi [~accountid:%{issue.project.lead}],` `the user %{system.currentUser.displayName} has written %{jiraExpression("issue.comments.filter(c=>c.author.accountId==user.accountId).length")} comments to this issue and created % {jiraExpression("issue.subtasks.filter(s=>s.reporter.accountId==user.accountId).length")} sub-tasks.}`	This example returns: **Hi [~accountid:557058:145e0473-5707-439c-80e4-1160dd57f114],** **the user John Doe has written 3 comments to this issue and created 2 sub-tasks.**

`issue.project.lead = system.currentUser ?`
`  (toLogicalValue(jiraExpression("issue.votes == 0")) ? "This issue does not seem to be important for anyone" :`
`  "There are voters for this issue!" ):`
`  "Not interesting for you!"`
	The function can also easily be used in a logical expression by converting the text result to a logical value (by using the function toLogicalValue()):
	
	If this expression runs for a user which is the project lead, this example returns :
	
	There are voters for this issue! in case there voters for this issue
	
	This topic does not seem to be important for anyone if there are no voters for this issue.
	
	In case the current user is not the project lead, only the message **Not interesting for you!** is returned.
	
	To achieve this the following functions is used:
	
	• toLogicalValue()

Additional information

Parameters used in this function

Parameter	Input (data type)	Description
`expression`	TEXT	Any given text which can be evaluated as Jira expression. ⚠️ The expression is not checked for syntactical correctness when defining it in the parser expression editor.

Output

This function returns a TEXT

Variant where you can additionally define a list of issues which can be used within the Jira expression g iven as the first parameter. You can access these issues through an injected context called **issues**.

Syntax

```
jiraExpression(expression, issueList) #Output: Text
```

Examples

Parser expression	Description

`%{sort(`
`  distinct(`
`  toStringList(`
`  jiraExpression("issues.flatMap(i=>i.`
`subtasks).flatMap(s=>s.components).map(c=>c.name)",`
`  linkedIssues())),`
`  ASC)}`
	This example returns an ordered list of the unique component names of all sub-tasks of the linked issues, e.g.
	
	Backend, Frontend, UI
	
	To achieve this the following function is used:
	
	• linkedIssues()
	
	You have to use **flatMap** because sub-tasks of a list of issues are returned as a multi list in a Jira expression.
	
	Since the function returns a text, it is converted to a text list using toStringList().

`toLogicalValue(jiraExpression("issues.every(c=>c.` `name=='UI')",subtasks()))`	This example returns `true` in a logical expression if all subtasks have a component **UI** To achieve this the following functions are used: • toLogicalValue() • subtasks()

`toStringList(issue.components.leads) ~ system.`
`currentUser AND`
`  toLogicalValue(jiraExpression("issues.map(i=>i.`
`issueType.name).includes('Task')",linkedIssues()))`
	This example returns `true` in a logical expression if the current user is lead of at least one componnetall subtasks have a component **UI**
	
	To achieve this the following functions are used:
	
	• toLogicalValue()
	• linkedIssues()

Additional information

Parameters used in this function

Parameter	Input (data type)	Description
`expression`	TEXT	Any given text which can be evaluated as Jira expression. ⚠️ The expression is not checked for syntactical correctness when defining it in the parser expression editor.
`issueList`	ISSUE LIST	Any given issue list. Usually this value is retrieved from a function (e.g. linkedIssues() or subtasks()).

Output

This function returns a TEXT

Variant where you can additionally define a string of Issue keys which can be used within the Jira expression given as the first parameter. You can access these issues through an injected context called **issues**.

Syntax
`jiraExpression(expression, issueKeys) #Output: Text`

Examples

Parser expression	Description

`%{sort(`
`  distinct(`
`  toStringList(`
`  jiraExpression("issues.map(i=>i.subtasks).`
`flatten().map(s=>s.components.map(c=>c.name)).`
`flatten()",`
`  "PU-2, PU-7, PU-120"))),`
`  ASC)}`
	This example returns an ordered list of the unique component names of all sub-tasks of the issues listed in the second parameter, e.g.
	
	Backend, Frontend, UI
	
	You have to use **flatten** because sub-tasks of a list of issues are returned as a multi list in a Jira expression.
	
	Since the function returns a text, it is converted to a text list using toStringList().

Additional information

Parameters used in this function

Parameter	Input (data type)	Description
`expression`	TEXT	Any given text which can be evaluated as Jira expression. ⚠️ The expression is not checked for syntactical correctness when defining it in the parser expression editor.

issueKeys	TEXT	A text with a comma-separated list of **issue keys**.

Output

This function returns a TEXT

Use cases and examples

Use case	JWT feature	Workflow function	Field type	Automated action	Parser functions
Add request participants		Execute remote action			jiraExpression()
Create a sub-task for each user selected in a User Picker field		Create issue			jiraExpression()
Inform the project manager about an added attachment		Send email			jiraExpression()