

Expression parser 101

Introduction

The **Automation Toolbox for Jira** expression parser provides [over 200 functions](#) and [operators](#) to read, manipulate and filter data from Jira issues, users, groups, projects and more.

Among the long list of functionalities, the parser functions support **setting and/or updating field values**, issue **filtering**, date and time **calculations**, string **manipulation** and the execution of **mathematical operations**.

The expression parser has been in constant development since 2009 when it was first introduced in [Jira Workflow Toolbox](#). Since that time, the expression parser has seen constant development, improvement, and extended functionality.



All **full list of supported functions** can be found here: [Expression parser 201 - All functions](#)

Main Features

The expression parser has two major core functionalities:

- **Extend accessibility** to
 - issue
 - system
 - project
 - version
 - component
 - and user **data** through the use of **Virtual Fields**
- Provide a set of **operators** and **functions** to
 - read
 - filter
 - extract
 - manipulate
 - write
 - update related **data**

Field codes and usage

Overview

One of the most important features of **Automation Toolbox for Jira** is the easy accessibility to Jira data stored in system fields, custom fields and a significant number of other, virtual fields that are made available by the **Automation Toolbox for Jira** implementation. You can access, validate, do mathematical calculations and manipulate the values found in these fields through the use of **field codes**.

These codes are unique identifiers (keys) to all available fields.

Automation Toolbox for Jira uses **field codes** in [triggers](#), [conditions](#), [selectors](#), and [actions](#):

- normal custom fields
- system fields
- parent fields available to all sub-tasks
- issue, project and user properties

Field codes are not only used as unique field identifiers, but they are also an important safety feature for the Jira instance. Custom fields, for instance, can be renamed and the names do not have to be unique, but using **Automation Toolbox for Jira** field codes make the fields you use in your rules immune to renaming.

You can choose the appropriate field codes by using the drop-down lists that **Automation Toolbox for Jira** makes available wherever **expressions** can be used.

On this page

- [Introduction](#)
- [Main Features](#)
- [Field codes and usage](#)
 - [Overview](#)
- [Field code notation](#)
 - [Field codes: STRING vs. NUMBER](#)
 - [Field codes in the documentation](#)
- [Virtual fields](#)
- [Parsing modes](#)
 - [General Information](#)
 - [Parsing modes overview](#)
- [Functions and operators](#)
 - [Functions](#)
 - [Operators](#)

Field code notation

Depending on the **context** in which they are being used, field codes will contain a prefix following this **notation**: **{origin.field/data}**.

Available **contexts** (or **origins**) in Automation Toolbox for Jira are:

Context	Description
Trigger	The issue , user , version , component or project event that triggers the execution of the rule .
Selector	The issue currently being processed by the selector . (e.g. an issue returned by a JQL query).  Selectors usually hold multiple issues. They will be processed iteratively one by one.
System	Some data does not have an issue context (e.g. the currently logged in user or the system date and time)

The **prefix**, denoting the **origin** (where the data should be **read from / written to**), is a referential part of the field code and **will be inserted into the expression** whenever you select a field from a dropdown list (as shown below).

Your browser does not support the HTML5 video element

Here are some examples:

- **%{trigger.issue.description}**
- **%{trigger.parent.summary}**
- **%{trigger.project.lead}**
- **%{selector.issue.cf10021}**
- **%{system.currentUser}**

Field codes for Jira standard or system fields will display the attribute in a legible form like **%{trigger.issue.summary}**.



All selected **custom fields** will be notated like this **%{trigger.issue.cfnnnnn}** where **nnnnn** contains the Jira **custom field id**.
Once an expression has been saved, the real name will be displayed in the configuration element.

The purpose of using the cfnnnnn notation is quite simple - **custom fields can be renamed** .

Field codes: **STRING** vs. **NUMBER**

Field codes must always be enclosed by **curly brackets** (or braces) but if they are used for **text-strings**, the brackets must be preceded by a **percent sign %**.

NUMBER

or **Date-Time fields** can be referenced as numbers using the following notation: **{so menumberfield}**. ( **no preceding % sign**)

STRING

Any field type or data type is susceptible of being transformed to text, so **any field can be referenced as a text-string value** using the following notation: **%{somefield}**.

Cascading Select or Multi-Cascading Select fields, where **i** is the index that represents the level to be accessed. (**i = 0** is used for base level) are notated as **%{somefield.i}** .



A full list of available data types [can be found here](#).

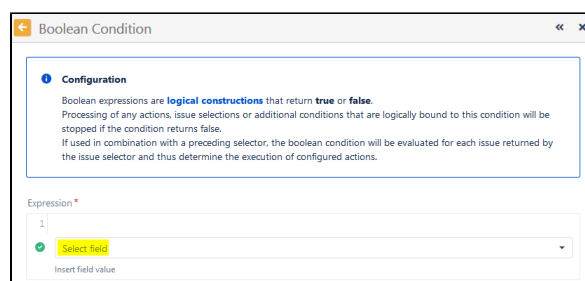
Field codes in the documentation

Wherever field codes are used in the documentation they will be notated with **three periods (...)** instead of prefixes.

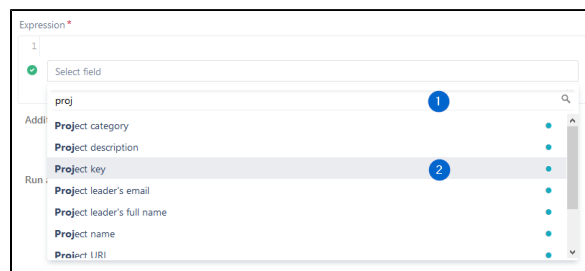
- `%{...summary}`
- `%{...cf10021}`
- `{...duedate}`

Example of using field codes

The example below shows and expression usage in a Boolean Condition.



Click on Select field and then



- 1 - start typing the name of the field you wish to insert
- 2 - click on one of the fields provided in the drop-down list



The chosen field code will then be inserted into your parser expression where you can then enhance the expression with more fields or any other methods the expression parser allows.

The expression syntax will be evaluated as you create the expression. If the syntax is correct, a green check circle will appear.



In the above examples, we've chosen to add a Boolean Condition validating that the **Project key** of the issue being processed is not (!=) TEST.

After you **save** the function, the real field names will be displayed in the rule element.

BOOLEAN CONDITION

Only if the following expression returns true

```
!(Project key (Trigger)) != "TEST"
```

This condition is evaluated as user in field **CURRENT USER**.

In contrast to system or ATJ special virtual fields (which cannot be renamed), **custom fields** will be inserted into an expression with a different notation as seen below:

Expression *

```
1 [event.issue.cf10900] > 1000
```

Y **BOOLEAN CONDITION**

Only if the following expression returns true

`{Sale Amount (Trigger)} > 1000`

This condition is evaluated as user in field `ASSIGNEE`.

In this example, we've chosen the custom field "Sale Amount" to evaluate. In the expression, it is notated as **cf10900**. **10900** is the unique field id in the Jira configuration.

Once the element is saved, it appears with its real name.

BOOLEAN CONDITION

Only if the following expression returns **true**

(Sale Amount Net (Trigger)) > 1000

This condition is evaluated as user in field **ASSIGNEE**.

If at some point the field should be renamed i.e. Sale Amount Net, the expression will stay the same, but the element will now display the new name and the rule does not need to be updated.

For more detailed information on field availability and parser usage, please see the section on [Virtual fields](#).

Virtual fields

Automation Toolbox for Jira provides a set of special fields called **virtual fields**, making almost all **properties of issues, projects and users** accessible to every feature in the app.

Virtual fields may be **read** and **written** by Automation Toolbox for Jira in the same way ordinary custom fields are.

Virtual fields and their associated **field codes**, were created to

- provide data accessibility beyond the scope of normal Jira workflow processing
- insure **data integrity** throughout their use.

In Automation Toolbox for Jira you can use virtual fields by searching for and picking their **associated field codes** in the dropdown menus provided wherever a **parser expression** can be inserted.

Your browser does not support the HTML5 video element

✔ All **comprehensive overview** of all available **virtual fields** can **be found here**.

Field Name	Field code	Value
Summary	%{issue.summary}	Issue Summary as STRING

Description	<code>%{issue.description}</code>	Issue Description as STRING
Assignee	<code>%{issue.assignee}</code>	User name of the Assignee as STRING
Parent's assignee	<code>%{parent.assignee}</code>	User name of the parent's Assignee as STRING
Number of votes received	<code>%{issue.votes}</code>	NUMBER of votes received by the issue.



Remember

Numeric field codes are **only available for number fields, date/time fields and countable virtual fields**.

Parsing modes



The available **Selectors**, **Conditions** and **Actions** depend on the selected **Trigger**.

The syntax depends on the **parsing mode** and the **context** of the rule. Therefore the following table explains the different parsing modes.

General Information

There **are multiple parsing modes** available in the **expression parser**. The **two most commonly** used parsing modes are:

- **Basic**: with this simple parsing mode you can write free text and insert field codes with format `%{...somefield}` or `%{...somefield.i}` anywhere in your text. These **field codes will be replaced at runtime with the corresponding field values** of the issue currently being processed.
- **Advanced**: with this parsing mode we can do much more complex text composition thanks to the usage of functions for replacing substrings, changing case, reading fields in linked issues, sub-tasks, JQL selected issues, and much more. It requires the text to be parsed to be written as **string expression** respecting the **parser syntax**.

You can **easily switch between** parsing modes. The available modes **depend on the context**!

Your browser does not support the HTML5 video element



Most functions will accept **string** values so casting values to string is a very **powerful function**. Details can be found below in the **converting data types** section!
Additionally you can directly transform a field value to text using the following syntax: `%{...somefield}`

Parsing modes overview

Mode	Supported features	Return type	Example
------	--------------------	-------------	---------

Basic	Field codes	STRING	The basic parsing mode supports the usage of field codes. Field codes can be used to access issue field values. simple text using a field code to read the summary <pre>This is the issue summary: %{trigger.issue.summary}</pre>
Advanced	Field codes Parser Functions	STRING	The advanced parsing mode has a defined syntax that allows you to write functions to read and manipulate data from any issue in Jira. Field codes are supported as well as clear text, written in quotation marks. Advanced expression to read the issue summary and use a function to get the assignee mail address <pre>"This is the issue summary:" + % {trigger.issue.summary} + " and the assignee mail is: " + userEmail(% {trigger.issue.assignee})</pre>
Math /date time	Field codes Parser Functions	NUMBER DATE DATE_TIME	The mathematical and date time parsing mode works like the advanced mode but expect a number as result instead of a string. The resulting number is used to update numeric or date time fields. In case of date or date time fields the number will be cast to a date. Time to resolve the issue <pre>{trigger.issue.resolutionDate} - {trigger.issue.createdDate}</pre>
Logical	Field codes Parser Functions	BOOLEAN	The logical parsing mode works like the advanced parsing mode but expression result must return true or false. Check if the assignee is equals the reporter <pre>{trigger.issue.assignee} = {trigger. issue.reporter}</pre>
Issue List	Field codes Parser Functions	ISSUE []	
String List	Field codes Parser Functions	STRING []	
Mixed	Field codes Parser Functions written in three curly braces	STRING	



Automatic parsing mode converter: You can write your text in **basic mode**, and then switch to **advanced mode**. The text to be parsed will be automatically rewritten as a string expression. Now, you can simply make the modifications you require, making use of text formatting functions, or inserting math or time expressions where needed.

To update issue fields the parsing result will be cast to the expected value e.g. a user name will be cast to a user to update a user field like the assignee field.

Functions and operators

To use the full power of the expression parser you can use **various functions** and combine them with **operators**.

Functions

- [Boolean expressions](#)
- [Numbers, Dates and Times](#)
- [Strings](#)
- [Selectable fields](#)
- [Users, groups and roles](#)
- [Versions](#)
- [Historical field values](#)
- [Miscellaneous](#)
- [Functions to temporarily store and retrieve values](#)
- [Working with lists](#)

A **comprehensive overview** can be found [here](#).

Operators

Different operators can be used in the expression parser. All operators are listed on these pages:

- [Operators](#)
- [List operators](#)