

Operators

The JWT expression parser accepts the **most common comparison operators** as well as **logical operators** .

The main purpose of these operators is to construct complex logical comparisons by **linking** individual expressions.

Comparison operators

The **operators**, their **meaning** and the applicable **data types** you can use them with are listed below.

A comparison always returns a BOOLEAN value.

Overview of all case-sensitive comparison operators

All operators respect the **case** of the **characters**.

Operator	Meaning	Examples (all examples return true)
=	equal to	<pre>1=1 true = true [1, 2, 3] = [1, 2, 3] ["blue", "red", "green"] = ["blue", "red", "green"]</pre>
		When working with Lists, each elements' existence and its order are being evaluated.
!=	not equal to	<pre>0 != 1 "HELLO" != "Hello" %{issue.description} != "Hello" true != false [1, 2, 3] != [1, 3, 2] ["blue", "red", "green"] != ["blue", "green", "red"]</pre>
		When working with Lists, each elements' existence and its order are being evaluated.
<	less than	<pre>1 < 2 "abc" < "bbc" "abc" < "abcd"</pre>

>	greater than	<pre>2 > 1 "bbc" > "abc" "abcd" > "abc"</pre>
<=	less than or equal to	<pre>3 <= 3</pre>
>=	greater than or equal to	<pre>"Hello world! Hello *" >= "Hello world"</pre>
~	contains	<pre>"Hello world!" ~ "world" #true. The text "world" is contained in the first text. %{issue.components.leads} ~ %{system.currentUser} #checks whether "Component leads" contains the "Current user". [1, 2, 3, 2, 2, 4] ~ [2, 1, 2] #true ["blue", "red", "green", "red", "white", "red"] ~ ["red", "green", "red"] #true ["green", "red"] ~ ["red", "green", "red"] #false</pre>
!~	does not contain	<pre>"Hello world!" !~ "world" #false. The text "world" is contained in the first text. %{issue.fixVersions} !~ %{issue.versions} #false if all "Affects version/s" are also selected as "Fix version/s". [1, 2, 3, 2, 2, 4] !~ [2, 1, 1, 4] #true ["blue", "red", "green", "red", "red"] !~ ["red", "green", "green", "red"] #true</pre>
in	is contained in	<pre>"world" in "Hello world!" #true. The text "world" is contained in the first text. %{system.currentUser} in %{issue.components.leads} #true if current user is a component lead of any of the issue's components [1, 1, 2] in [2, 1, 1, 1, 4] #true ["blue", "red", "red"] in ["red", "green", "blue", "red", "red"] #true 2 in [1, 2, 3] #true "blue" in ["red", "blue", "white"] #true</pre>
not in	is not contained in	<pre>"Hello world!" not in "world" #true %{issue.versions} not in %{issue.fixVersions} #false if all "Affects version/s" are also selected as "Fix version/s". [1, 1, 2, 2] not in [2, 1, 1, 1, 4] #true ["blue", "red", "red", "blue"] not in ["red", "blue", "red", "red"] #true 5 not in [1, 2, 3, 3, 4] #true "orange" not in ["blue", "red", "white"] #true</pre>
any in	any element is in	<pre>%{issue.versions} any in %{issue.fixVersions} # true if any selected "Affects version/s" has also been selected as "Fix version/s". [1, 3] any in [3, 4, 5] #true ["blue", "white"] any in ["black", "white", "green"] #true</pre>
none in	no single element is in	<pre>%{issue.versions} none in %{issue.fixVersions} #true if no selected "Affects version /s" has also been selected as "Fix version/s". [1, 2] none in [3, 4, 5] #true ["blue", "red"] none in ["black", "white", "green"] #true</pre>

When comparing lists, the **exact number** of occurrence (cardinality) per element must match.

Parser expression	Output	Description
<code>["blue", "red", "green", "red", "white", "red"] ~ ["red", "green", "red"]</code>	true	This expression returns true , since the element (text) red appears at least twice in the first list and the element (text) green occurs at least once in the first list.
<code>["green", "red"] ~ ["red", "green", "red"]</code>	false	This expression returns false , since the element (text) red does not appear twice in the first list.

Overview of all case ignoring comparison operators

The following comparison operators can be used with and [data types](#) .

All operators **ignore** the **case** of the **characters**.

Operator	Meaning	Examples (all examples return true)
<code>=~</code>	equal to	<pre>"HELLO" =~ "Hello" #true "up" =~ "UP" #true ["blue", "red", "green"] =~ ["Blue", "RED", "Green"] #true</pre>
<code>!=~</code>	not equal to	<pre>" HELLO" !=~ "Hello" #false, since there is a whitespace in the first text "up" !=~ "down" #true ("up" !=~ "UP") #false ["blue", "red"] !=~ ["Blue", "green"] #true ["blue", "red"] !=~ ["Red", "BLUE"] #true ["blue", "red", "green"] !=~ ["Blue", "RED", "Green"] #false</pre>
<code>~~</code>	contains	<pre>"Hello World!" ~~ "world" #true, checks whether a text contains a substring. "A small step for a man" ~~ "STEP" #true ["one", "two", "three"] ~~ ["TWO", "One"] #true, checks whether a text list contains all the elements of another text list.</pre>
<code>!~~</code>	does not contain	<pre>"Hello World!" !~~ "bye" #true, checks whether a text does not contain a substring. "A small step for a man" !~~ "big" #true ["one", "two", "three"] !~~ ["Four"] #true, checks whether a text list does not contain a single element of another text list. (["one", "two", "three"] !~~ ["TWO"]) = false</pre>

in~	is contained in	"world" in~ "Hello World!" #true, checks whether a substring is contained in another text. "STEP" in~ "A small step for a man" #true ["Two", "One"] in~ ["one", "two", "three"] #true, checks whether all the elements of a text list are contained in another text list.
not in~	is not contained in	"bye" not in~ "Hello World!" #true, checks whether a substring is not contained in another text. "big" not in~ "A small step for a man" #true ["Four"] not in~ ["one", "two", "three"] #true, checks whether any of the elements of a text list are not contained in another text list. ["Two"] not in~ ["one", "two", "three"] #false
any in~	any element is in	["blue", "violet"] any in~ ["Blue", "Red", "Green"] #true ["Five", "One"] any in~ ["FOUR", "FIVE", "SIX"] "bye" #true
none in~	no single element is in	["Orange"] none in~ ["red", "blue", "green"] #true, checks whether none of the elements of a text list are not contained in another text list. ["orange"] none in~ ["Red", "Orange"] #false

Applicable data types

Below you find a comprehensive matrix of all **operators** and applicable **data types** .

Comparison Operator	BOOLEAN	NUMBER	TEXT	NUMBER LIST	TEXT LIST	ISSUE
=	✓	✓	✓	✓	✓	✓
!=	✓	✓	✓	✓	✓	✓
<	-	✓	✓	-	-	-
>	-	✓	✓	-	-	-
<=	-	✓	✓	-	-	-
>=	-	✓	✓	-	-	-
~	-	-	✓	✓	✓	✓
!~	-	-	✓	✓	✓	✓
in	-	-	✓	✓	✓	✓
not in	-	-	✓	✓	✓	✓
any in	-	-	-	✓	✓	✓
none in	-	-	-	✓	✓	✓
=~	-	-	✓	-	✓	-
!=~	-	-	✓	-	✓	-
~~	-	-	✓	-	✓	-
!~~	-	-	✓	-	✓	-
in~	-	-	✓	-	✓	-

not in~	-	-	✓	-	✓	-
any in~	-	-	-	-	✓	-
none in~	-	-	-	-	✓	-

Please be aware the both operands of the respective comparison must have the **same data type**. The only exceptions are the following:

- **Automatic casting from** **to** : Whenever you write a numeric term at the right-hand side of a **comparison operator** like =, and the left-hand side is occupied by a text term, the parser will automatically transform the right-hand side term into a text (e.g. "30" = 30 will be interpreted the same way as "30" = "30")
- **Single values as operand in list operations:** Operators ~, !~, in and not in can be used for checking a single element (**or**) against a **or** a
- **Comparison with the null value:** A **field** which is not set or an empty text is interpreted as **null**. A field, which doesn't contain a number, is also interpreted as **null**.

Things to remember

Remember	Examples
Operators ~, !~, in and not in can be used for checking a single element (NUMBER or TEXT) against a NUMBER LIST or a TEXT LIST	<pre>1 in [1, 2, 3] ["blue", "red"] ~ "blue"</pre>
Operators ~, !~, in and not in when used with a text are useful to look for substrings in another string.	<pre>"I love coding" ~ "love" "I don't like Mondays" !~ "Fridays" "love" in "I love coding" "Fridays" not in "I don't like Mondays"</pre>
Operators ~, !~, in and not in respect cardinality, i.e., container list must have at least the same number of elements as contained list.	<pre>[1, 1] in [1, 1, 1] [1, 1] not in [1, 2, 3]</pre>
Operators = and !=, when used for comparing lists, require to have the same elements , with the same cardinality and the same order .	<pre>[1, 2, 3] = [1, 2, 3] [4, 5, 6] != [4, 6, 5]</pre>
Operators <, >, <= and >= work according to lexicographical order when comparing text.	<pre>1 < 2 "abc" < "bbc" "abcd" > "abc"</pre>



Logical operators

The table below lists all logical operators that can be used for **linking logical terms** in an expression.

Logical operators take logical terms (which return values) as operands and can thus be built using:

- a boolean value
- a [JWT expression parser function](#) returning a boolean value
- a comparison
- a logical term enclosed by brackets ()
- two logical terms connected with a logical operator, where boolean literals and comparisons themselves are logical terms.

Logical operators can only be used in **logical expressions** in the [Logical mode](#) or in combination with the conditional operator.

Overview of all logical operators

Operator	Meaning	Precedence
NOT or !	logical negation	1 (highest)
AND or &	logical conjunction	2
OR or 	logical disjunction	3
XOR	exclusive or, i.e., a XOR b is equivalent to a AND !b OR !a AND b	3
IMPLIES or IMP	logical implication, i.e., a IMPLIES b is equivalent to !a OR b	4
XNOR or EQV	logical equivalence, i.e., a EQV b is equivalent to a IMPLIES b AND b IMPLIES a	4 (lowest)

A single logical term can be enclosed by **brackets ()** in order to increase the readability of the expressions or to define a **precedence** which differs from the given one.

Logical operators can also be written in lower case (e.g. **and** , **or**)



Conditional operator

The conditional operator **?** : is a powerful operator to construct conditional expressions.

It basically allows you to construct the following expression: **IF** logical_expression **true THEN** term_1 **ELSE** term_2.

```
<logical_expression> ? <term_1> : <term_2>
```

CONDITION



CONDITION MET



CONDITION NOT MET

Weather = 



It is sunny



It is cloudy

count(subtasks()) > 0



„There are sub-tasks!“



„No sub-tasks.“

The conditional operator is extremely helpful when being used in [calculated fields](#).

Examples of using the conditional operator

Expression	Description
<code>%{issue.priority} = "Highest" ? "Please have a look at this issue immediately" : "No stress, come back later"</code>	IF the priority of an issue is Blocker, THEN this function will return "Please have a look at this issue immediately" ELSE it will return "No stress, come back later".
<code>{issue.duedate} != null ? ({...duedate} - {...currentDateTime}) / {HOUR} : 0</code>	IF an issue does have a due date set (due date is not null), THEN this function will return the number of hours from the current date-time to the due date ELSE it will return 0 .
<code>%{issue.somefield} = "Red" ? "Color" : "No color"</code>	IF a custom field (e.g. a select list) has a value of Red, THEN this function will return "Color", ELSE it will return "No color".
<code>timePart({...currentDateTime}, LOCAL) > 21:00 AND timePart({...currentDateTime}, LOCAL) < 7:00 ? "Night" : "Day"</code>	IF the current time is between 21:00 and 7:00 THEN this function will return "Night" , ELSE it will return "Day".



List operators

Function	Short description	Output
APPEND	Combines the elements of two lists .	LIST
UNION	Returns distinct elements of two lists.	LIST
INTERSECT	Returns common elements of two lists.	LIST
EXCEPT	Removes certain elements from a list.	LIST

Order of operations

If you use multiple operators in a single expression, they will follow a certain order in which they are processed or a precedence.

OPERATORS	PRECEDENCE	ASSOCIATIVITY
INTERSECT	1 (highest)	Left-to-right
APPEND, EXCEPT, UNION	2 (lowest)	Left-to-right

- When using the list operators, you have to make sure that both lists that you compare are of the **same type**.
- All operators are **case insensitive**, i.e., they can also be written in lower case: **append**, **union**, **intersect** and **except** .
- There are **four equivalent functions** available for each type of list, and their behavior is **exactly equivalent** to that of its corresponding operator.
 - `append()`
 - `except()`
 - `intersect()`
 - `union()`
- This way, you can choose to use **operators** or **functions** according to your preference. Although operators yield shorter expressions and with fewer parentheses, the usage of functions produces a more functional consistent syntax.

If you still have questions, feel free to refer to our [support](#) team.