Search Linked Issues-JQL Functions

linkedIssuesFromFilter()

Finds all issues which are linked by specified relation with those found by saved filter.
When **relationDirection** is provided then **relation** should also be provided.
When **recursionNumber** is specified then issues linked to found linked issues are also considered as a result.

Since version 2.10.0 we added **relationDescription** handling in queries.

Parameters:

- **filter** - could be given by name or its id.
- **relation** - is optional and given by name (about relation parameter more information in FAQ section), name of relation (or relation description) must match names that are set in configuration, by default they start with capital letter see more
- **relationDescription** avilable since version 2.10.0 (about relation description parameter more information in FAQ section), must match descriptions that are set in configuration, see more
- **relationDirection** - optional, have values **inward** or **outward** (do not provide if you choose direction by relation description)
- **recursionNumber** - optional, have positive integer values (not supported with relation description parameter)

Syntax:

```
linkedIssuesFromFilter(filter)
linkedIssuesFromFilter(filter, relation)
linkedIssuesFromFilter(filter, relation, relationDirection)
linkedIssuesFromFilter(filter, relation, relationDirection, recursionNumber)
linkedIssuesFromFilter(filter, relationDescription)
```

Examples:
Find all issues linked by any relation to all issues returned by filter myFilter

```
issue in linkedIssuesFromFilter("myFilter")
```

Find all issues linked by relation Duplicate to all issues returned by filter myFilter

```
issue in linkedIssuesFromFilter("myFilter", "Duplicate")
```

Finds all issues linked by relation Duplicate in direction inward to all issues returned by filter myFilter

```
issue in linkedIssuesFromFilter("myFilter", "Duplicate", "inward")
issue in linkedIssuesFromFilter("myFilter", "is duplicated by")
```

Find all issues linked by relation Duplicate in direction outward to all issues returned by filter myFilter

```
issue in linkedIssuesFromFilter("myFilter", "Duplicate", "outward")
issue in linkedIssuesFromFilter("myFilter", "duplicates")
```

Find all issues linked by any relation to all issues returned by filter with id 102010

```
issue in linkedIssuesFromFilter("102010")
```

linkedIssuesFromQuery()

Finds all issues which are linked by specified relation with those found by JQL. When **relation Direction** is provided then **relation** should also be provided.
When **recursionNumber** is specified then issues linked to found linked issues are also considered as a result.

Since version 2.10.0 we added **relationDescription** handling in queries.

List of JQL Functions:

Parameters:

- **JQL**- could be given by JQL query
- **relation** -is optional and given by name (about relation parameter more information in FAQ section), name of relation (or relation description) must match names that are set in configuration, by default they start with capital letter see more
- **relationDescription** avilable since version 2.10.0 (about relation description parameter more information in FAQ section), must match descriptions that are set in configuration, see more
- **relationDirection**- optional, have values **inward** or **outward** (do not provide if you choose direction by relation description)
- **recursionNumber**- optional, have positive integer values (not supported with relation description parameter)

Syntax:

```
linkedIssuesFromQuery(JQL)
linkedIssuesFromQuery(JQL, relation)
linkedIssuesFromQuery(JQL, relation, relationDirection)
linkedIssuesFromQuery(JQL, relation, relationDirection, recursionNumber)
linkedIssuesFromQuery(JQL, relationDescription)
```

Examples:
Find all issues linked by any relation to all issues returned by JQL

```
issue in linkedIssuesFromQuery("project = DEMO AND issuetype = BUG")
```

Find all issues linked by relation Blocks to all issues returned by query issuekey >= X

```
issue in linkedIssuesFromQuery("issuekey >= X", "Blocks")
```

Find all issues linked by relation Blocks in direction inward to all issues returned by JQL

```
issue in linkedIssuesFromQuery("project = DEMO AND issuetype = BUG", "Blocks"
, "inward")
issue in linkedIssuesFromQuery("project = DEMO AND issuetype = BUG", "is
blocked by")
```

Find all issues linked by relation Blocks in direction outward to all issues returned by JQL

```
issue in linkedIssuesFromQuery("project = DEMO AND issuetype = BUG", "Blocks"
, "outward")
issue in linkedIssuesFromQuery("project = DEMO AND issuetype = BUG", "blocks"
)
```

Find all Y and Z, when X is blocked by Y and Y is blocked by Z? (Recursive lookup)

```
issue in linkedIssuesFromQuery("issuekey = X", "Blocks", "inward", 2)
```

You have issues DEMO-1 cloned by DEMO-2, DEMO-2 cloned by DEMO-3, DEMO-3 cloned by DEMO-4. Finds issues clones DEMO-1 for only 2 recursive lookup, then JIRA will return DEMO-2, and DEMO-3

```
issue in  linkedIssuesFromQuery("issuekey = DEMO-1", "Cloners", inward, 2)
```

You have issues DEMO-1 duplicated by DEMO-2, DEMO-2 duplicated by DEMO-3, DEMO-3 duplicated by DEMO-4. Finds issues duplicats DEMO-1 for only 3 recursive lookup, then JIRA will return DEMO-2, DEMO-3 and DEMO-4.

```
issue in  linkedIssuesFromQuery("issuekey = DEMO-1", "Duplicate", inward, 3)
```

parentIssuesFromFilter()

Finds all issues which are parents of subtasks issues selected by specified saved filter.

Parameters:

- **filter**- could be given by name or its id.

Syntax:

```
parentIssuesFromFilter(filter)
```

Examples:

Finds all parent issues for all issues returned by filter myFilter

```
issue in parentIssuesFromFilter("myFilter")
```

Finds all parent issues for all issues returned by filter with id 102010

```
issue in parentIssuesFromFilter("102010")
```

parentIssuesFromQuery()

Finds all issues which are parents of subtasks issues selected by JQL.

Parameters:

- **JQL**- could be given by JQL query

Syntax:

```
parentIssuesFromQuery(JQL)
```

Examples:
Find all parent issues for all issues returned by JQL

```
issue in parentIssuesFromQuery("project = DEMO AND issuetype = Sub-Task")
```

subtaskIssuesFromFilter()

Finds all issues which are subtasks of issues selected by specified saved filter

Parameters:

- **filter**- could be given by name or its id.

Syntax:

```
subtaskIssuesFromFilter(filter)
```

Examples:
Find all subtask issues for all issues returned by filter myFilter

```
 issue in subtaskIssuesFromFilter("myFilter")
```

Find all subtask issues for all issues returned by filter with id 102010

```
 issue in subtaskIssuesFromFilter("102010")
```

subtaskIssuesFromQuery()

Finds all issues which are subtasks of issues selected by JQL

Parameters:

- **JQL**- could be given by JQL query

Syntax:

```
subtaskIssuesFromQuery(JQL)
```

Examples:
Find all subtask issues for all issues returned by JQL

```
issue in subtaskIssuesFromQuery("project = DEMO AND issuetype = Task")
```

numberOfLinkedIssuesFromQuery()

This function is available in 2.5.0 plugin version and later.

Finds all issues which number of linked issue meet a condition. Relation type and direction is not analyse by function so all linked issues are proceeded to check condition. Query has three required parameters.

Parameters:

- **JQL**- JQL query to analyse.
- **mathematicalSymbol**- available symbols ==, <=, !=, >=, >, <
- **numberOfLinkedIssues**- number of linked issues

Syntax:

```
numberOfLinkedIssuesFromQuery(JQL, mathematicalSymbol, numberOfLinkedIssues)
```

Examples:
Finds all linked issues from DEMO project which contain more than two linked issues.

```
issue in linkedIssuesFromQuery("project = DEMO",">","2")
```

numberOfLinkedIssuesFromFilter()

This function is available in 2.5.0 plugin version and later.

Finds all issues which number of linked issue meet a condition. Relation type and direction is not analyse by function so all linked issues are proceeded to check condition. Query has three required parameters.

Parameters:

- **filter**- could be given by name or its id.
- **mathematicalSymbol**- available symbols ==, <=, !=, >=, >, <
- **numberOfLinkedIssues**- number of linked issues

Syntax:

```
numberOfLinkedIssuesFromQuery(filter, mathematicalSymbol,
numberOfLinkedIssues)
```

Examples:
Finds all linked issues returned by myFilter which contain less than three linked issues.

```
issue in linkedIssuesFromQuery("myFilter","<","3")
```

Finds all linked issues returned by 102011 filter id which contain exacly one linked issue.

```
issue in linkedIssuesFromQuery("102011","==","1")
```

having...()

This function is available in 2.4.0 plugin version and later.

Since version 2.10.0 we added **relationDescription** handling in queries.
All the above functions have their having... counterparts:

havingLinkedIssuesFromQuery(...),

havingLinkedIssuesFromFilter(...),

havingSubtaskIssuesFromQuery(...),

havingSubtaskIssuesFromFilter(...),

havingParentIssuesFromQuery(...),

havingParentIssuesFromFilter(...).

Those functions do the same task as the previously-existing functions, but return not the link 'targets', but **link 'sources' that generated the targets**. havingXXXX("subquery")is equivalent to:reversedXXX("") and subquery, (where""is a subquery which matches all the issues).

For example:

```
issue in havingSubtaskIssuesFromQuery("text ~ linux")
```

returns those issues which are parents of some subtask issues, and containlinux text somewhere. This is equivalent to (but faster than) this query:

```
issue in parentIssuesFromQuery("") and text ~ linux
```

Note that this is not the same as query:

```
issue in parentIssuesFromQuery("text ~ linux")
```

which returns parent issues of subtask issues containinglinux(returned parents themselves don't have to containlinux).

Similarly otherhaving...queries work.

epicsFromFilter()

This function is available in 2.9.0 version plugin version and later.

This function is available only with JIRA Software.

Function will return all *Epics* that where assigned to issues returned by given **filter** paramet er.

Function allows you to discover all *Epics* that where used in your issue set. In this case issue set is defined as **filter**.

Parameters:

- **filter**- could be given by name or its id.

Syntax:

```
epicsFromFilter(filter)
```

Examples:

```
issue in epicsFromFilter("testowy")
```

epicsFromQuery()

This function is available in 2.9.0 version plugin version and later.

This function is available only with JIRA Software.

Function will return all *Epics* that where assigned to issues returned by given **query** parameter.

Function allows you to discover all *Epics* that where used in your issue set. In this case issue set is defined as **query**.

Parameters:

- **JQL** - could be given by JQL query

Syntax:

```
epicsFromQuery(JQL)
```

Examples:

```
issue in epicsFromQuery("project = TEST")
```

epicsWithIssue()

This function is available in 2.9.0 version plugin version and later.

This function is available only with JIRA Software.

Function will returned all *Epics* that have at least one issue assigned. Function will search for all *Epics* in your JIRA instance.

Parameters: no

Syntax:

```
epicsWithIssue()
```

Examples:

```
issue in epicsWithIssue()
```

epicsWithoutIssue()

This function is available in 2.9.0 version plugin version and later.

This function is available only with JIRA Software.

Function will returned all *Epics* that have no issues assigned. Function will check all *Epics* in your JIRA instance.

Parameters: no

Syntax:

```
epicsWithoutIssue()
```

Example:

```
issue in epicsWithoutIssue()
```