S-JQL Cookbook

Here are the most common examples of using S-JQL.

1. Find issues added to a structure

Goal: Suppose that you are using a structure named "My todo list" as a collection of issues, and you want to see in the Issue Navigator all issues added to this structure.

How to achieve: In the Issue Navigator, switch to Advanced Searching and run the following query:

```
issue in structure("My todo list")
```

If you want to find issues added to the Default Structure, you can omit the structure name:

```
issue in structure()
```

^ up to the list of examples

2. Quick Filter for JIRA Agile's (GreenHopper) Scrum Board to display only low-level issues in a structure

Setup: Suppose that you are using a structure named "Project work breakdown" to organize tasks under higher-level "container" issues that provide an overview of your team's work. In this setting, the actual tasks are at the bottom level of the hierarchy. Also, suppose you are using JIRA Agile's Scrum Board to manage your sprints.

Goal: You want to see only the actual tasks in backlog, hiding the container issues.

How to achieve: Add a Quick Filter to your JIRA Agile (GreenHopper) board with the following JQL:

```
issue in structure("Project work breakdown", leaf)
```

If your structure is organized such that *two* lower levels matter to you on the JIRA Agile board, you'll search for leaf issues and their parents with this JQL:

```
issue in structure("Project work breakdown", "leaf or parent of leaf")
```

^ up to the list of examples

3. Retrieve all Epics in a certain status and all of their children

Setup: You have a structure named "Enterprise Portfolio" with Epics on the top level, Stories beneath them, and Tasks with their Sub-Tasks occupying the lower levels of the hierarchy.

Goal: You need to see Epics in status *Assigned* with all of their children.

How to achieve: In the Issue Navigator, switch to Advanced Searching and run the following query:

```
issue in structure("Enterprise Portfolio", "issueOrAncestor in [type = Epic and status = Assigned]")
```

If you want to see these issues in the structure, go to Structure Board and type this query in the Search Area in the JQL mode.

Also, you can type only the last part of the query if you use S-JQL search mode:

```
issueOrAncestor in [type = Epic and status = Assigned]
```

^ up to the list of examples

4. Find Test Cases associated with Stories in an active sprint

Setup: Suppose that you have a structure named "Enterprise Portfolio Testing", where you have Epics on the top level, Stories on the second level, then come Test Sub-Tasks, and finally Test Cases.
You are also using JIRA Agile (Greenhopper) to manage your sprints, which contain Stories. The fact that a Test Case is associated with an Story is recorded only in the structure.

Goal: You need to find those Test Cases that are associated with Stories in an active sprint.

How to achieve: You can use Issue Navigator's Advanced Searching capability or open the structure on the Structure Board and use its Search Area in the JQL mode to run this query:

```
issue in structure("Enterprise Portfolio Testing", "[type = 'Test Case'] and ancestor in [type = Story and
sprint in openSprints()]")
```

Or, you can type only the last part of the query if you use S-JQL search mode on the Structure Board:

```
[type = 'Test Case'] and ancestor in [type = Story and sprint in openSprints()]
```

^ up to the list of examples

5. Find all issues that are blocking critical issues

Setup: Suppose that you have a structure named "Dependency structure" where parent-child relationship corresponds to dependency: each child blocks its parent. (You might have configured a Links Synchronizer to synchronize this structure with the "Dependency" JIRA issue link.)
Let's also suppose that you consider critical those issues that have priority *Critical*.

Goal: You want to see all issues that are blocking critical issues, according to the structure.

How to achieve: You'll need to find children of critical issues. You can use Issue Navigator's Advanced Searching capability or open the structure on the Structure Board and use its Search Area in the JQL mode to run this query:

```
issue in structure("Dependency structure", "child of [priority = Critical]")
```

Or, you can type only the last part of the query if you use S-JQL search mode on the Structure Board:

```
child of [priority = Critical]
```

^ up to the list of examples

6. Find all unassigned issues in a part of a project

Setup: Suppose that you use a structure named "Project work breakdown" to break down your project into smaller pieces, so that if you have an issue somewhere in the structure, all of its children at all levels constitute a separate part of a project.

Goal: You are focusing on a part of a project under the issue with key `PROJ-123`, and you want to see unassigned issues in that part of the project.

How to achieve: Use this JQL query to find all unassigned descendants of `PROJ-123`:

```
issue in structure("Project work breakdown", "[assignee is empty] and descendant of PROJ-123")
```

^ up to the list of examples

7. Top-level view on unfinished parts of a project

Setup: Let's continue with the "Project work breakdown" structure from the previous example. Suppose that there are several top-level issues representing different parts of the project.

Goal: You want to have a view on the parts of the project that are yet unfinished.

How to achieve: In the Structure terms, you need to see the root issues that have unresolved descendants. To have a persistent view, create a Saved Filter with the following JQL:

```
issue in structure("Project work breakdown", "root and descendants in [resolution is empty]")
```

^ up to the list of examples

8. Find violations of the rule "Tasks must be under Epics or Stories"

Setup: You have a structure named "Planning" where you put issues of types Epic, Story, and Task. Your team follows the convention that Tasks are always put under Epics or Stories. However, as humans are fallible, sometimes a Task ends up being in a wrong place — either on the top level, or under another Task.

Goal: You need to find Tasks that violate the rule, so that you can put them in the right place.

How to achieve: In the Search Area on the Structure Board, run the following JQL search:

```
issue in structure("Planning", "[type = Task] and parent not in [type in (Epic, Story)]")
```

^ up to the list of examples

9. Find violations of the rule "An issue cannot be resolved if it has unresolved children"

Setup: Suppose that "Planning" is a work breakdown structure. Your team follows the convention that an issue cannot be resolved unless all of its children are resolved.

Goal: You need to find the issues violating this rule.

How to achieve: In the Search Area on the Structure Board, run the following S-JQL search:

```
[resolution is not empty] and child in [resolution is empty]
```

^ up to the list of examples

10. Find issues that can be resolved because all their children are resolved

Setup: Suppose that "Planning" is a work breakdown structure. Your team follows the convention that once all children of an issue are resolved, the issue can be resolved as well.
The best solution for this would be to use a Status Rollup Synchronizer, but suppose that for some reason you want to do it manually.

Goal: You need a way to manually resolve those issues that have all of their children resolved.

How to achieve: Open the structure on the Structure Board. When you paste the query given below into the Search Area (ensure that the JQL mode is selected), the issues that you can resolve will be shown. You can resolve them one by one. Here's the query you need:

```
issue in structure("Planning", "[resolution is empty] and not(child is empty or child in [resolution is
empty])")
```

^ up to the list of examples

11. Get a view of a second (third, ...) level of the hierarchy

Setup: There is a large structure named "Joint Effort" where different users track their issues on several levels: Customer Relations department works with the top-level issues, Project Managers break them down in several issues on the second level, Team Members work with issues under second-level issues.

Goal: Each user wants to see only the relevant part of the structure. Customer Relations department wants to filter out lower-level issues to focus on the top-level ones, and Project Managers sometimes want to focus on just the second-level issues in the context of their parent requests.

How to achieve: use the Search Area on the Structure Board to run the specific queries (ensure that the S-JQL mode is selected.) Toggle the Filter button to hide the issues on the lower levels.

To see top-level issues, run this query:

```
root
```

To see second-level issues (top-level issues will be still displayed, but greyed out), run this query:

```
child of root
```

If you would need to dig even deeper, to see the third level but not the lower ones, you'd use this query:

```
child of (child of root)
```

^ up to the list of examples