Documentation

Author: Igor Sereda
Date: 25-Jul-2014 18:41
URL: https://wiki.almworks.com/display/structure/Documentation

Table of Contents

Links to the available documentation collections:

- Structure 101
- Structure User's Guide
- Structure Administrator's Guide
- Structure Developer's Guide
- Structure FAQ
- Structure Troubleshooting

Download documentation:

Name	Version	Date

1 Structure 101

Structure 101

Installing Structure

1. Install the Plugin - open **Plugin Manager**, search for Structure and click **Install** or **Try Now**.
2. Get Evaluation License
    a. Check if you need a license by going to **Administration | Structure | License Details**.
    b. If needed, click **Get Evaluation License** and install the received license.
    c. See Setting Up Structure License for details.
3. Configure Enabled Projects
    a. Open **Administration | Structure | Configuration** and enable at least one project.
    b. See Selecting Structure-Enabled Projects for details.

Structure Crash Course

- Please watch the introductory video on the right.
- During initial installation, Structure plugin has created an empty, default *Global Structure*. You can decide to use only that single structure, or you can create as many structures as needed. To create a new structure, use **Structure | Create Structure** menu.
- A structure is a "container" for issues — it is initially empty, and someone needs to put issues into it and arrange them in a hierarchy. This can be done manually, issue by issue, or automatically with an Import function or a synchronizer.
    - To add an issue to a structure, open that issue, expand Structure section, and click **Add to Structure** button. Alternatively, use **Search** button on the Structure Board to find the issues you need to add, and place them at a specific position in Structure with drag-and-drop. (Click left-side dots to select multiple issues.)
    - To create a sub-issue, select the issue in a structure and click +Sub-Issue button.
    - To make structure always contain issues that satisfy a certain JQL filter, install a Filter Synchronizer — open **Structure | Manage Structures** page, locate your structure, click **Settings** in the **Sync With** column, and install a Filter synchronizer.
- Structure is integrated with **JIRA Agile (GreenHopper)** — the issue details panel on the Agile board shows the position of that issue in structure. You can install JIRA Agile (GreenHopper) Synchronizer to maintain a structure with **Theme-Epic-Story-Task** hierarchy, or to synchronize backlog/sprint rank with the position of issues in a structure.

Getting Started - Walkthrough

1. Open **Structure | Manage Structure** menu and click **Create Structure**. Create your own private structure to play with.
2. On the Manage Structure page, click on your structure name to go to the Structure Board.
3. On the Structure Board, click Search, then click **JQL** and type `assignee=currentUser() order by updated` . You can use any other query - see more about Searching and Filtering.
4. When some issues are found by the search, click "More Issues in JIRA" button if it's not switched on, and then drag-and-drop several found issues into the Structure.
5. When you have several issues added to the structure, try to rearrange them to create a hierarchy. Hold **Shift** and drag issues up and down or left and right to place them at a different position in the hierarchy. You can also drag by the drag handle at the left side. See Using Drag and Drop to learn more.
6. Use **keyboard arrows** to navigate structure or **Ctrl+Arrow** to move issues up/down or indent/unindent.
7. Use Structure Toolbar for other actions like Copy/Paste.
8. Click on an issue's summary or hit "o" to open issue page - there's structure on that page too.
9. Hit **Ctrl+?** to see a cheat sheet with Keyboard Shortcuts.

Structure Introduction and Overview:

Version 2.0 Highlights:

2 Structure User's Guide

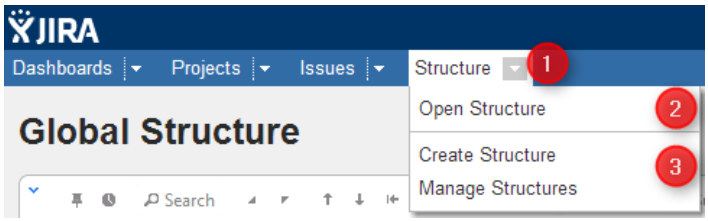
This section contains information for Structure users.

Contents:

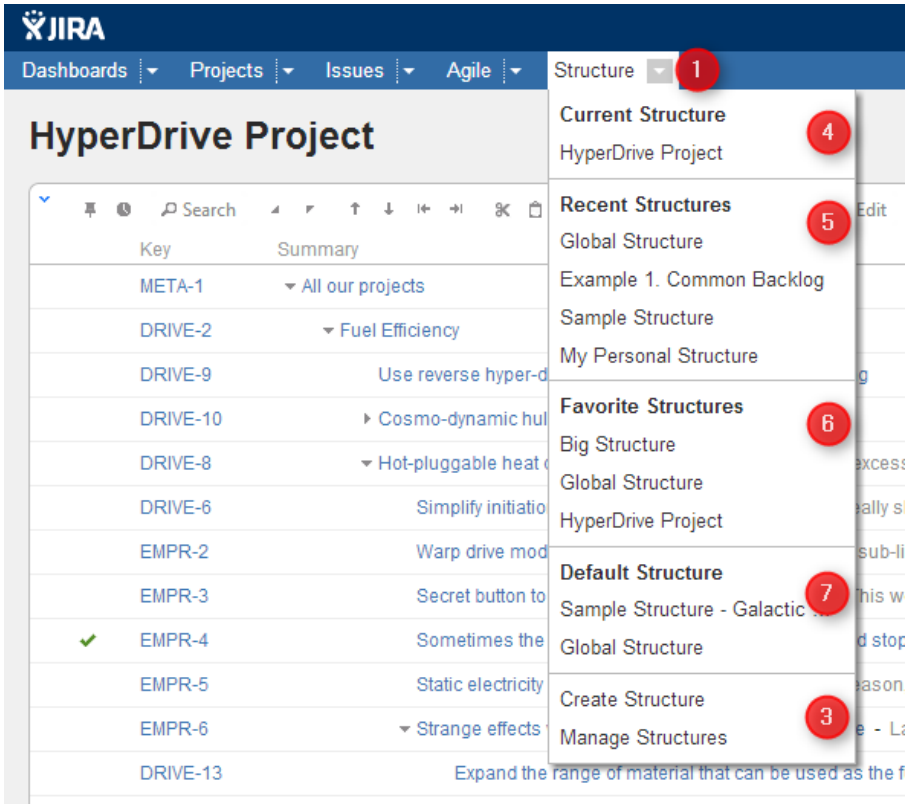
2.1 Structure Menu

Structure Plugins adds another menu to the top-level navigation bar.

In the simplest setup, when you have access to only a single structure, the menu looks like this:



If you have created additional structures, or have access to more than one structure, the menu looks more like this:



More about specific parts of the menu (identified by numbers on the screenshots):

1. The **Structure** drop-down - use it to show the menu. You can also click **Structure** to open Structure Board.
2. Opens Structure Board and shows the structure. (The only one that exists or that you can see.)
3. **Create Structure** takes you to Creating New Structures and **Manage Structure** takes you to Managing Structures.
4. **Current Structure** section shows the last viewed structure, click the structure name to open Structure Board with that structure. You can also click the top-level menu (1) to open that structure.
5. **Recent Structures** section shows structures that you've visited recently, or those which have been recently updated. Click a structure to open it.
6. **Favorite Structures** lists structures you have marked as your favorite. This section is absent if you don't have favorite structures.
7. **Default Structure** section shows the system-wide default structure. Also, if another default structure is defined for the current project, it is also shown in this section.

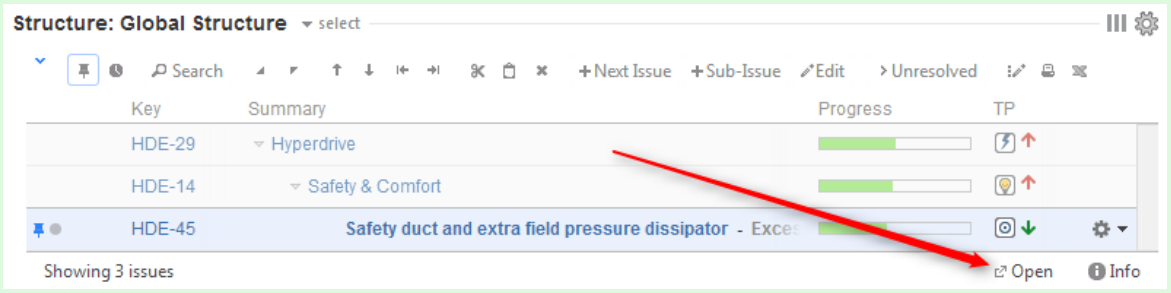
2.2 JIRA Pages with Structure

Issue structure is displayed by the **structure widget**, which is located in several places in JIRA - on a dedicated Structure Board, on the issue page, on the project, component and version pages. The widget displays a scrollable grid with the hierarchical list of issues and lets you work with the structure as well as with every issue.

Most functionality of the structure widget is the same on every page, however there are few specific things that the structure does on an issue page and on project/component/version pages. You can work with the structure on the page that's most convenient for you:

- Structure Board
- Structure on the Issue Page
- Structure Gadget
- Structure on the Project, Component and Version Pages
- Structure on Agile Boards
- Structure on the Issue Navigator Page

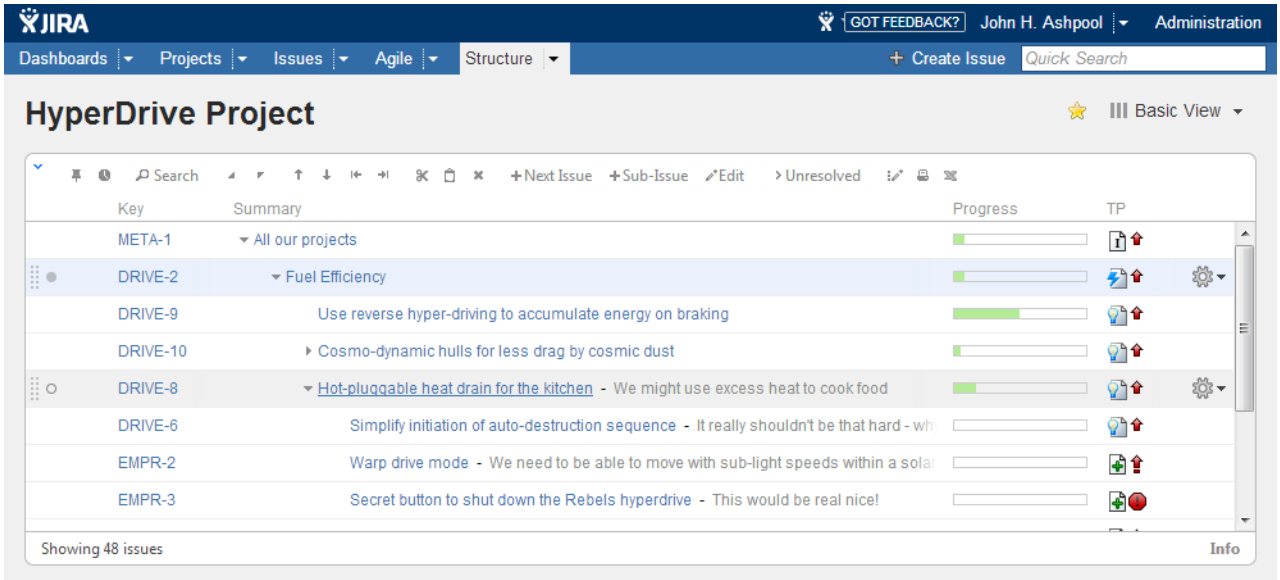
If you need to go to the Structure Board from any other page with the widget, click the **Open** link in the bottom of the widget. This will open the currently viewed structure on the Structure Board.



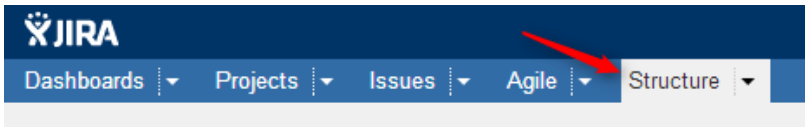
See also: Working with the Structure Widget

2.2.1 Structure Board

Structure Board is a full-screen view with the structure widget and without anything else, so it's good for focusing your work on the structure.



Structure Board is opened when you click on the **Structure** top navigation menu in JIRA.



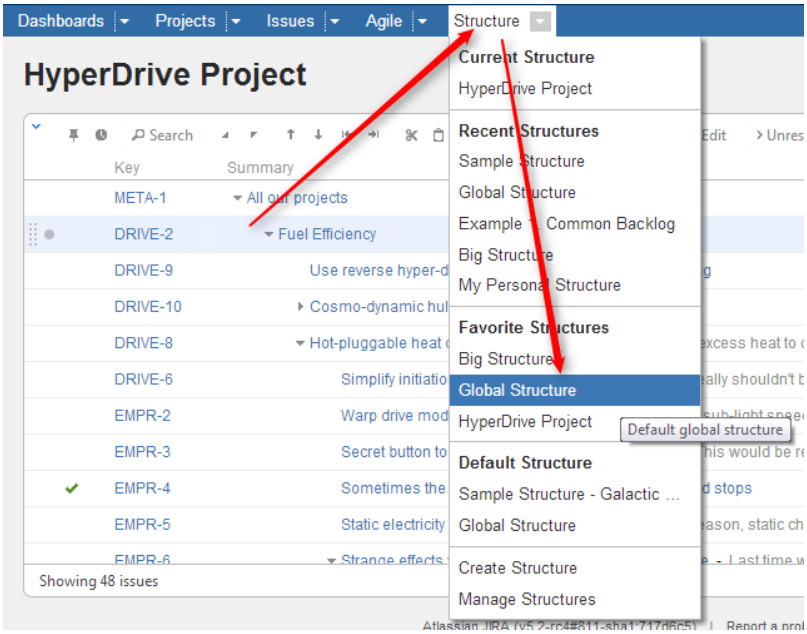
You can press **g** and then quickly **s** on any JIRA page to open the structure board. (*Go Structure*)

You can make the structure board your JIRA Home page.

If you have several structures in JIRA, clicking **Structure** top navigation menu opens the most recently used structure.

You can open a specific, not current structure on the structure board using any of the following options:

- Click on the drop-down arrow beside **Structure** top navigation menu and choose one of the recently viewed structures.



- Click on the drop-down arrow beside **Structure** and select **Manage Structure** - this will take you to the Manage Structure page where you can browse for the structure you need, and then click on the structure name to open it.

- If you know the ID of the structure you need opened, you can directly open an URL: http://*your.jira.address*/secure/StructureBoard.jspa?s=*structure-id*

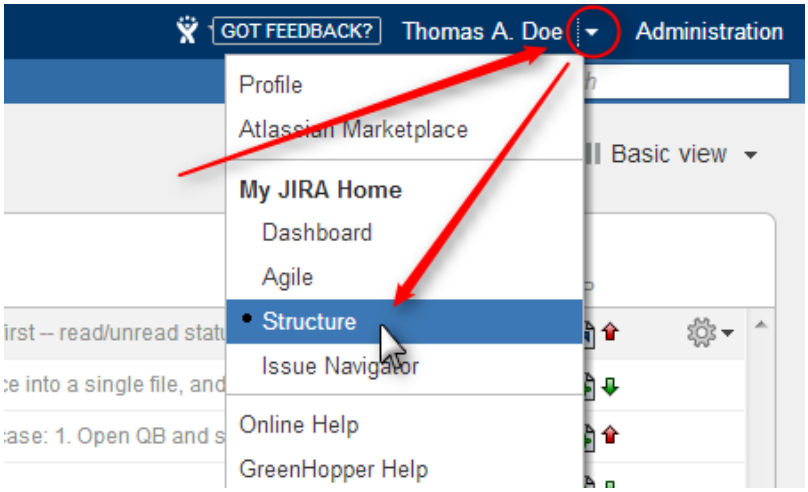
Structure Board will try to accomodate structure widget nicely on your browser page to take advantage of most of the available space.

You can always share the way you see a Structure Board by creating a Structure Perspective.

Making Structure Board Your JIRA Home

If you want to go straight to the Structure Board when you log in to JIRA, you can make it your JIRA Home page. To do so:

1. open the dropdown next to your user name in the top right of the page;
2. choose **Structure** in the **My JIRA Home** section.

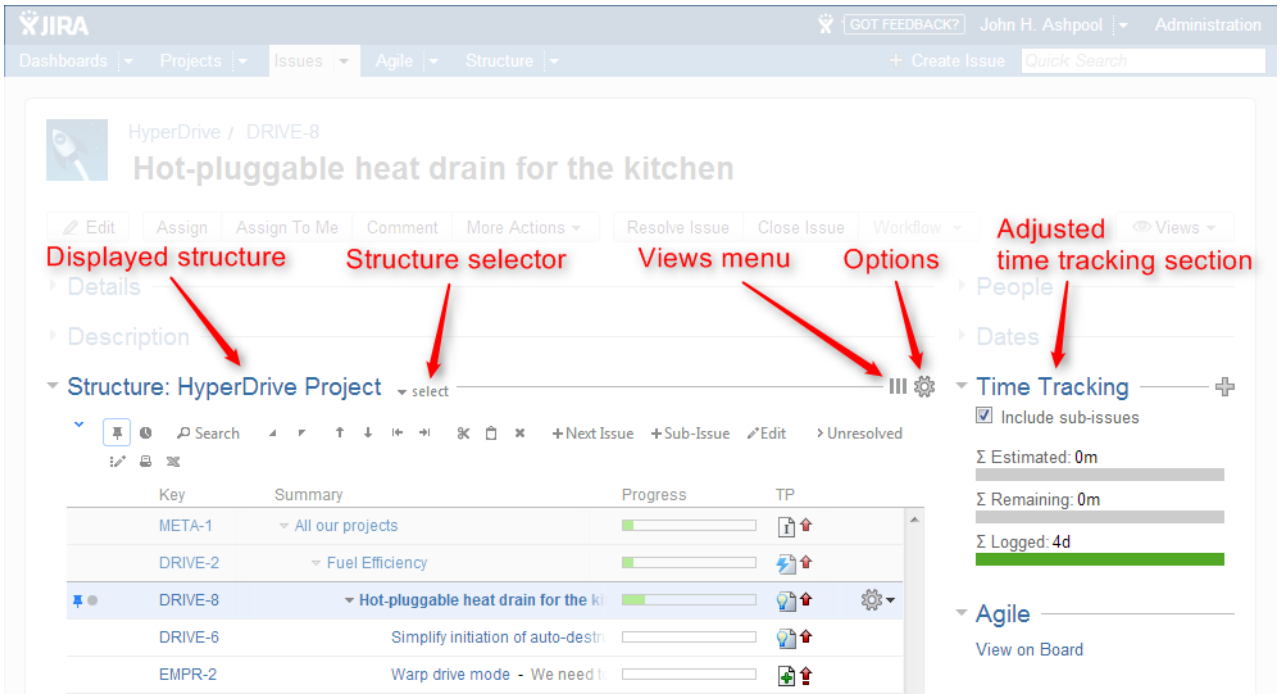


When used as a JIRA Home page, the Structure Board will show your current structure, as if you clicked the top-level Structure menu.

You can also go to your JIRA Home at any time by clicking the JIRA logo in the top-left corner of any JIRA page.

2.2.2 Structure on the Issue Page

Structure widget is displayed on an issue page in case the issue belongs to a project enabled for the Structure Plugin. The widget is presented as a separate section, located right above **Activity** section.



Structure section name includes the name of the currently selected structure.

When you open an issue page with structure section, Structure plugin makes a decision, which structure to display on that page initially – determined by the Structure Options for the Issue Page. The viewed issue is automatically located and selected in the structure.

When there's only one structure in the system (that you have access to), the structure section is simplified – structure name and structure selector are not shown.

Starting with JIRA 6, search results on the Issue Navigator page can display the details of a selected issue. The details panel also contains Structure section. You may want to configure view to fit only the necessary information in a narrower space left for the Structure widget.

There are several specific features on the Issue Page that are not present on the Structure Board:

- Collapsible Structure Section
- Current Structure Selector
- Views and Options Drop-Downs
- Adjusted Time Tracking Section
- Current Issue Panel
- Activity section

Collapsible Structure Section

The section with the structure can be hidden, as any other section on the page. Once you hide the structure section, it will remain hidden even if you open another issue page.

Also, Structure section is automatically hidden if the issue you view does not belong to the selected structure. (This behavior can be adjusted by changing Structure Options.)

When the structure section is hidden, the issue hierarchy is not loaded from the server – it will be downloaded only when you first open the structure section.

The *hidden* flag is stored in a browser cookie or local storage, along with flags for other sections.

By default, structure widget in the Structure section is in Pinned Issue Mode - it displays only the parent issues and sub-issues of the viewed issue. You can unpin the structure and view the whole hierarchy by using **Pin** button on the toolbar or a keyboard shortcut.

Current Structure Selector

By default, structure section selects a structure that contains the issue being displayed (the initial structure selection can be tuned by the user by clicking Options icon).

When structure section is expanded, there's **select** button near the section header and the name of the structure, which allows you to switch to a different structure. When you switch to another structure, the data is automatically reloaded and the selected structure becomes your current structure.

Views and Options Drop-Downs

Located at the right corner of the Structure section header are Views and Options icons.

Click Views icon to open Views Menu and select another view for the displayed structure.

Views icon turns **blue** when the currently used view has been locally adjusted.

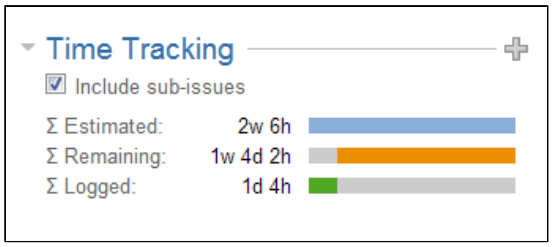
Click Options icon to open Structure Options for the Issue Page.

Adjusted Time Tracking Section

Structure plugin automatically sums up time tracking information from the sub-issues and displays aggregate values in the time tracking section. Whenever any change is detected in the child issues, the time tracking information is refreshed.

You can turn off time tracking aggregation and revert to viewing Time Tracking panel as JIRA would present it (without Structure plugin) by turning off **Include structure sub-issues** check box. The browser will remember your preference and will show you original Time Tracking panel when you open other issues, until you turn **Include structure sub-issues** checkbox on again.

When time tracking section is not present, it means that neither the current issue nor its sub-issues have any time tracking info.



Current Issue Panel

When the currently selected structure does not contain the issue which page you are viewing, an automatic Secondary Panel is shown in the structure widget with this issue. It is called Current Issue panel.

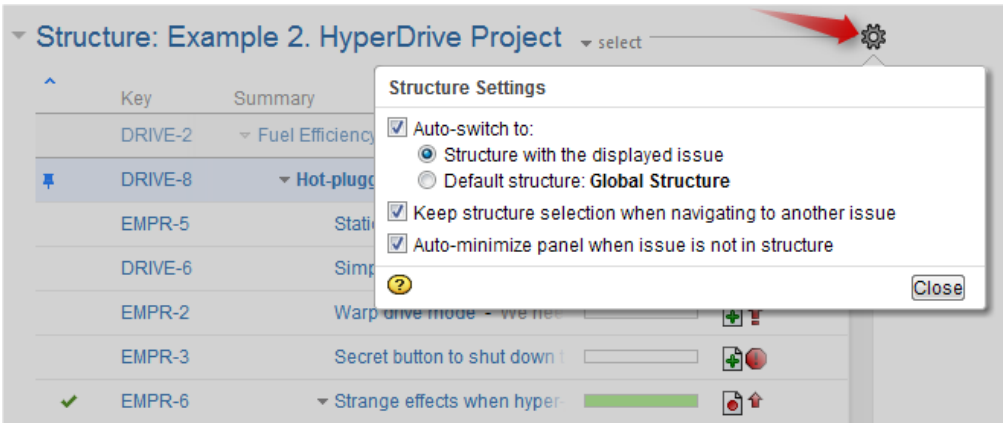
Current Issue Panel allows you to use drag-and-drop or copy-paste to easily place the current issue at some position in the current structure. After you have added the current issue to the structure, the panel automatically disappears. You can also hide the panel by clicking on its "pin" icon in the top left corner of the structure widget.

Activity section

The **Activity** section of the issue page displays information about structure changes that affect this issue. This may be useful, for example, if you want to find out why this issue is in that particular position within a structure, who and when added or moved it there. See Structure Activity Stream for more information.

Structure Options for the Issue Page

There are a few options that let you tune how Structure section on the issue page works. Click on the gear button in the section header to bring them up. The changes are saved to the server and applied immediately.



1. Which Structure to Select Initially?

When you have multiple structures, an issue might be present in more than one structure. When issue page is opened, Structure plugin has to decide which structure to display initially in the Structure section.

This is controlled by a number of parameters:

Auto-switch	When auto-switch is turned on, the structure is selected based on which project and structures the issue belongs to. When auto-switch is turned off, the Structure section shows the structure that the user viewed previously on the Structure Board (the *current* structure).
Auto-switch: structure with displayed issue	When this auto-switch mode is selected, Structure plugin looks for a structure that contains the issue displayed on the page.
Auto-switch: default structure	When this auto-switch mode is selected, the Default Structure for the issue's project will be always selected (even if the issue is not in that structure yet).
Keep structure when navigating	When you click on another issue within the Structure Widget, the browser takes you to that issue's page. If this option is turned on, the new page displays the same structure as the page you navigated from (auto-switch is not applied). ⚠ It's better to leave this option on! It lets you avoid inadvertent change of the viewed structure when you go through the structure's issues.

✓ When there's only one structure you can view, these options are not displayed.

⚠ **Keep structure when navigating** option currently does not work when you hit **Back** button in your browser – the structure on the issue page you return to will be selected based on the Auto-switch settings.

2. Should Structure Section be Minimized Automatically?

The setting **Auto-minimize panel when issue is not in structure** controls whether Structure plugin automatically collapses the Structure panel in case the initially selected structure does not contain the displayed issue.

You can always click on the section header to open the Structure panel and proceed with adding the issue to the structure, viewing the whole structure or selecting another structure.

3. Options Scope and Default Options

When you adjust Structure Options, the changed settings apply whenever you view any other issue on this JIRA instance. (The settings are saved in your account settings.)

The default values of these options can be configured by JIRA Administrator on the Structure Defaults page.

2.2.3 Structure Gadget

Structure provides a dashboard gadget that allows you to view and edit structure. The gadget may be also be imported into Confluence and included on a Confluence page.

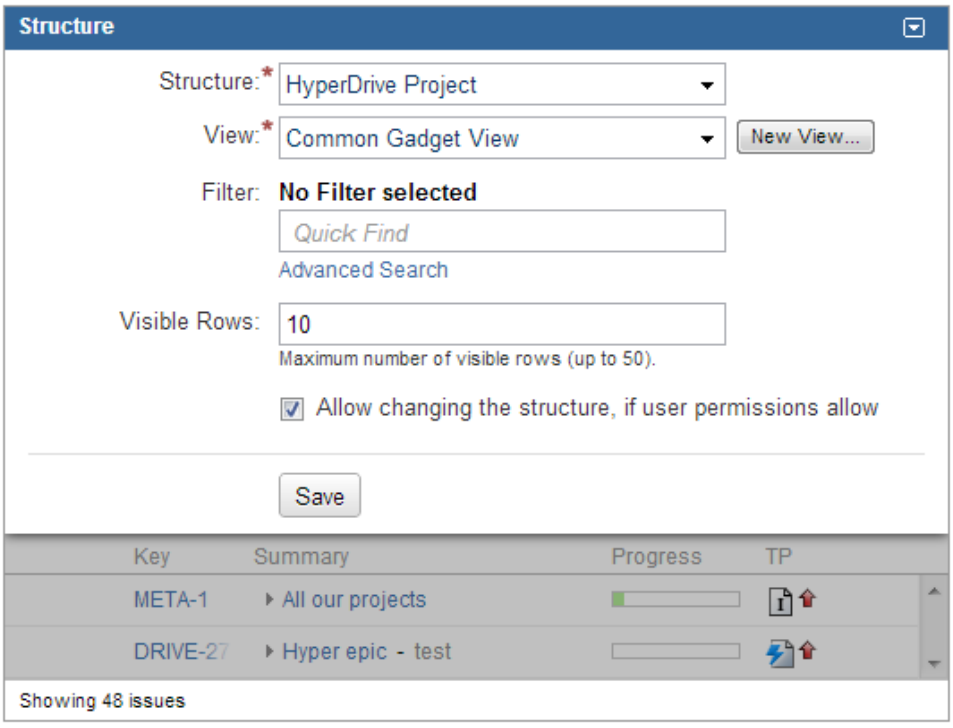
Adding Structure Gadget to Dashboard

Structure gadget is added as any other gadget: click **Add Gadget** button in the top right corner of the dashboard, find "Structure" and click **Add It Now**. You need to have change permissions on the dashboard (if you don't have permissions to change the dashboard, you can try to create a copy using **Tools | Copy Dashboard**).

You can add several gadgets showing different structures on the same dashboard.

Configuring the Gadget

When you first add a gadget to dashboard, gadget configuration panel appears with a dimmed preview of the gadget below. (The same panel is shown when you use **Edit** command from the gadget header drop-down or when you edit macro with Structure gadget in Confluence.)



To configure the gadget:

1. Select a **Structure**. Click arrow down in the Structure selector to view recently used and favorite structures, or start typing structure name and let the drop-down suggest the matching structures.

2. Select a **View**. Click arrow down to choose from views associated with the selected structure, or start typing and let Structure suggest matching views. The selected view determines which columns gadget displays. (You will be able to adjust the view later.)
3. Optionally, select a saved **Filter**. Displayed structure will be filtered in the same way it can be filtered on Structure Board – see Searching and Filtering Within Structure.
4. Decide how large the gadget is allowed to be and specify **Visible Rows** number. If there are fewer visible rows, the gadget shrinks; if more, a vertical scroll bar appears. Pick any number between 2 and 50.
5. Decide if you'd like dashboard viewers to make changes to the structure or issues (subject to the user's permissions) and select or un-select the **Allow Changes** checkbox.
6. Click Save.

Deselect **Allow Changes** to protect the structure from accidental changes, such as changes caused by drag-and-drop or hitting Delete key.

Configuring Gadget View

There are several ways to configure view (columns) for the gadget.

- **Select a predefined view from drop-down.**
  Just select a view or start typing and allow Structure to suggest matching views. Set up other gadget parameters and click **Save**.
- **Start with an existing view and modify it.**
  To use this method you need to have Update permission on the modified view.

  Note that if the view you're changing is used in other gadgets, you will be modifying other gadgets' columns configuration as well.

  1. Select a view in the gadget configuration panel.
  2. Click **Save**.
  3. Adjust view by adding, removing or rearranging columns – see Customizing Columns for details.
  4. A message "*View has been adjusted. Save | Revert*" will appear in the gadget footer. Click **Save**.

- **Start with an existing view, adjust it and save as a new view.**
  Use this method if you don't have **Update** access to the view you start with or if you don't want to change it to avoid messing up other gadgets configuration.
  1. Select a view in the gadget configuration panel.
  2. Click **Save**.
  3. Adjust view by adding, removing or rearranging columns – see Customizing Columns for details.
  4. Open gadget configuration again by clicking **Edit** in the gadget header drop-down menu.
  5. Click **New View** button, located beside view selector. Additional form appears – enter new view name and click **Create View**.
  6. If this gadget is going to be visible to other users, make sure they have access to the view you've created. Gadget configuration panel will suggest to make this view public – click **Let everyone use this view** to make the view available to everyone.
- **Start with a new view and adjust it.**
  1. Without selecting a view in the gadget configuration panel, click **New View** button.
  2. Additional form appears – enter new view name and click **Create View**.
  3. If this gadget is going to be visible to other users, make sure they have access to the view you've created. Gadget configuration panel will suggest to make this view public – click **Let everyone use this view** to make the view available to everyone.
  4. Click **Save**.
  5. The created view will have basic default columns (issue key and summary). Adjust view by adding, removing or rearranging columns – see Customizing Columns for details.
  6. A message "*View has been adjusted. Save | Revert*" will appear in the gadget footer. Click **Save**.

If the user viewing the gadget does not have **Use** permission on the configured view, the gadget will show a default view with only Issue Key and Summary as columns.

When you see "*View has been adjusted. Save | Revert*" message in the gadget footer, it means that you have changed columns configuration for this gadget. These changes are local and are effective only in the same browser they were made in. Click **Save** to save and share the changes or **Revert** to go back to the configuration stored on the server. See Saving and Sharing Views for details.

Using the Gadget

The Structure gadget contains a stripped-down version of the standard Structure widget that you see on other pages. Because the screen space available to a gadget is usually limited, it lacks features like search and secondary panels. It also doesn't have a toolbar. However, most keyboard shortcuts are functional. If the gadget allows editing, you can rearrange issues with drag-and-drop; you can also move, create, edit, and delete issues using the keyboard.

Structure Dashboard Gadget is a bit limited where it comes to editing issue fields, due to some incompatibilities between field editors and gadget framework. Because of that, only a handful of fields can be edited from within Structure Gadget. See Editing from Gadget for more details.

If gadget is displayed in its "home" JIRA dashboard (not in Confluence or elsewhere), the last column lets you use action drop-down for the issues.

Using Structure Gadget in Confluence

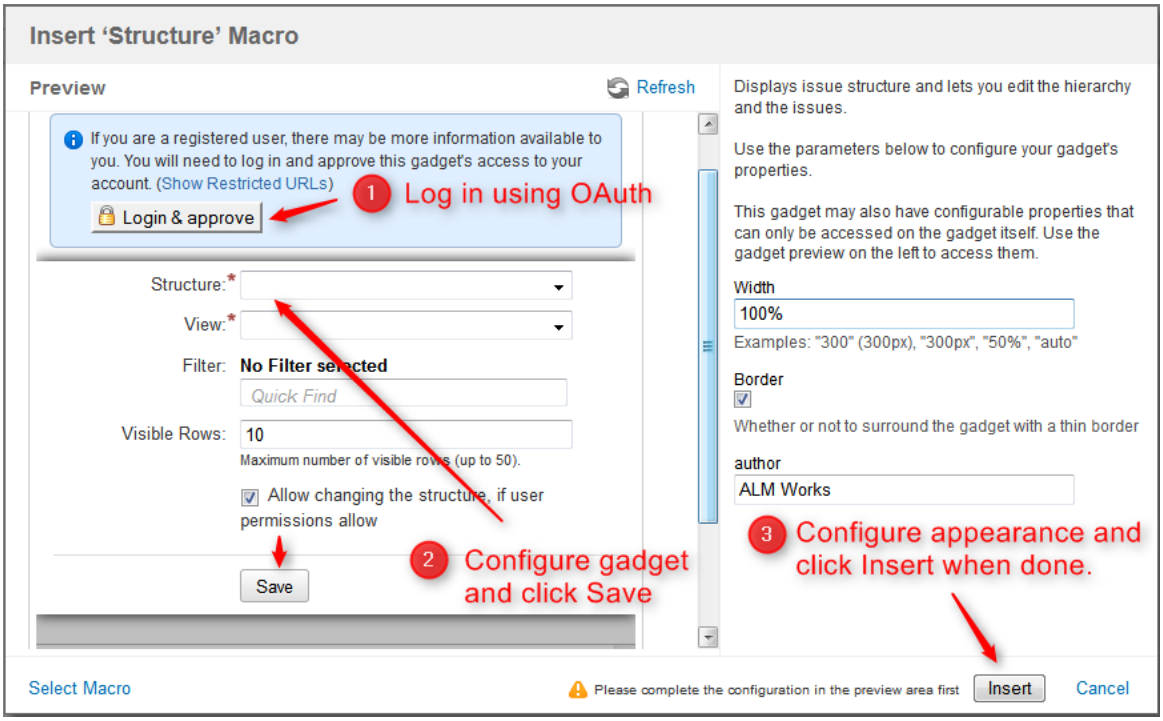
You can embed Structure Gadget in a Confluence page and view or edit structure in Confluence.

Before you can use Structure Gadget on a Confluence page, your Confluence administrator must add Structure Gadget to Confluence Configuration. If you try to insert a macro and don't see *Structure* in the list, most likely the gadget is not configured.

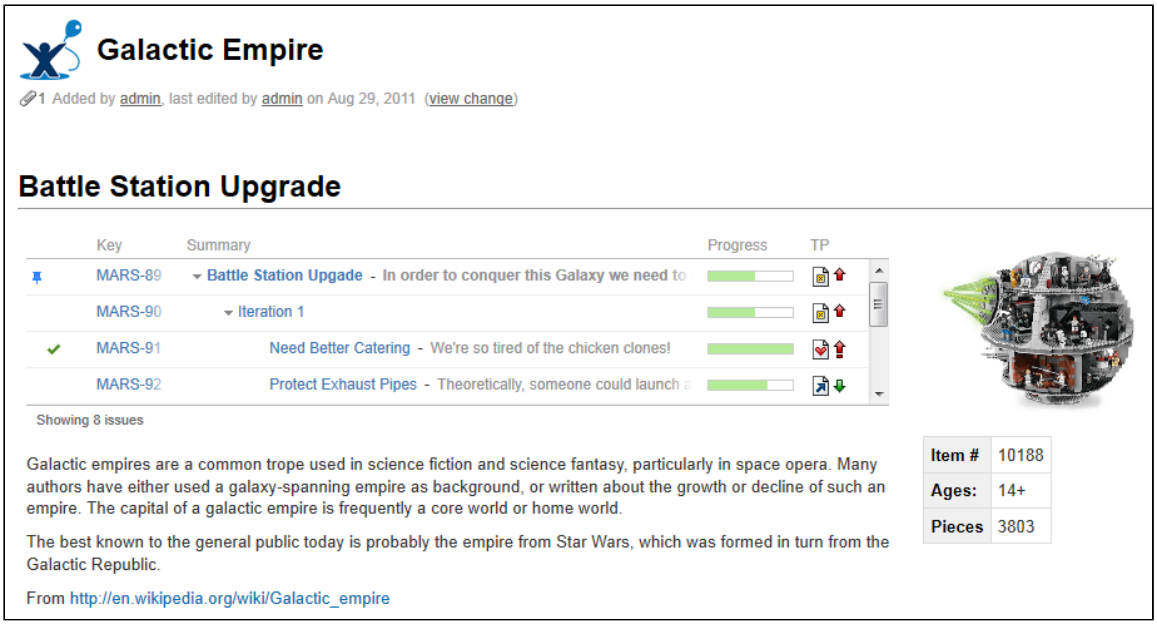
The displayed Structure gadget is not suitable for printing. Support for printable Structure gadget is coming next. So far, please use Printable Page to print a structure separately.

How to Add Structure Gadget

1. When editing a page, click Insert/Edit Macro, and select **Structure**. Macro configuration dialog appears.



2. If **Login & approve** button is shown, you need to log in into JIRA first.
3. If **Structure plugin not available** message appears, then you currently don't have any visible structures. Probably you need to login.
4. Configure gadget - select the structure to be displayed and configure other parameters, then click **Save**.
5. Configure gadget appearance, for example, set **width** to **100%** and **border** to **not selected**.
6. Click **Insert** and you're done!



Adding Structure Gadget to Confluence Configuration

Adding JIRA gadgets to Confluence is covered by Atlassian documentation. Here's a list of references to get you started.

1. Unless you'd like to see Structure as anonymous user, connect Confluence to JIRA using **Application Links**. You'll need to enable outgoing authentication from Confluence to JIRA.
   Documentation: Configuring Application Links
   a. Use **OAuth Authentication** to let the Confluence page viewer authenticate separately with JIRA. (Preferred)
      Documentation: Configuring OAuth for an Application Link
   b. Use **Trusted Applications** authentication if you'd like confluence users act in JIRA under the same usernames without additional authentication.
      Documentation: Trusted Application Authentication

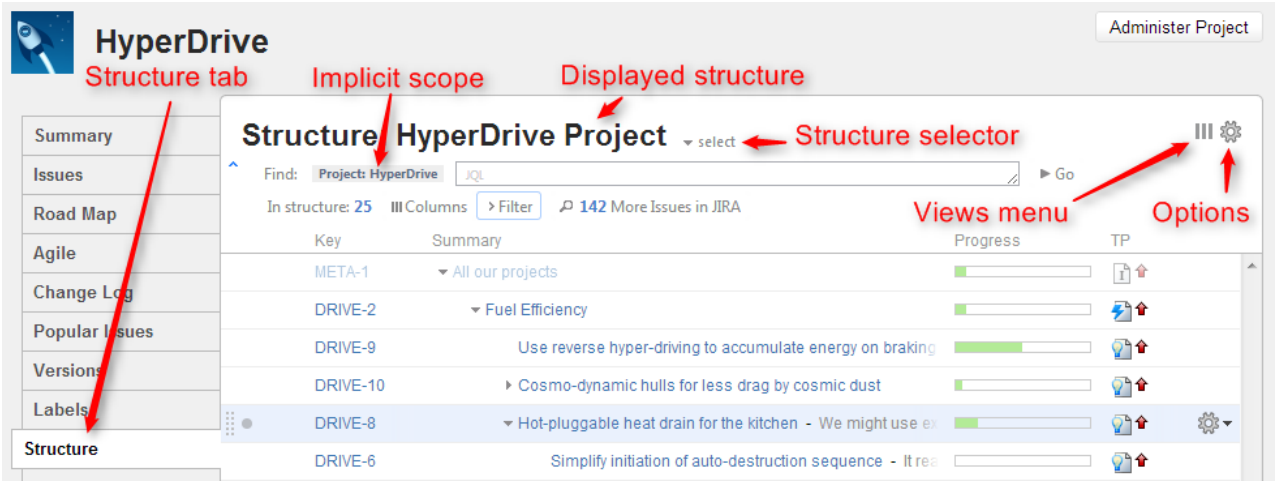
> Structure Gadget may allow modification of structure, updating and creating issues under the account that is used by Confluence to access JIRA. Make sure you understand how Trusted Applications work before allowing production structures to be accessed with this kind of authentication. Using OAuth is more secure because the end-user will never be able to do anything that they are not able to do directly in JIRA.

2. Add Structure Gadget to the list of **External Gadgets**. Remember that you can copy the URL of the Gadget from the gadgets selection dialog, when you click **Add Gadget** on JIRA dashboard. Documentation: External Gadgets
3. Check on a sample page if you can include Structure macro and get data from JIRA.

Main article: Adding JIRA Gadgets to a Confluence Page

2.2.4 Structure on the Project, Component and Version Pages

Structure widget is displayed in a separate tab on a Project page, and on the Component and Version pages as well - if the project is enabled for Structure.



The selected Project (or Component / Version) defines the **scope** of the current structure view - which is used to filter the structure (see details below). For the sake of clarity, we describe the functionality of the structure on the Project page tab; it works in the same way on the Component page tab and Version page tab.

The widget on the project page is the standard fully functional structure widget, but it has several specific features not found on the Structure Board:

- Current Structure Selector
- Implicit Scope
- Views Menu
- Structure Options
- Simplified View When There's Only One Structure

Current Structure Selector

By default, structure tab displays the current structure (selected previously by the structure selector or by opening a structure board with that structure). There's a **select** button at the top of the page near the current structure name, which allows you to switch to a different structure without leaving the page.

Pressing the **select** button brings up a drop-down menu with your recent, favorite, and default structures.

When you switch to another structure, the data is automatically reloaded and the selected structure becomes your current structure.

This works exactly like the Current Structure Selector on the issue page.

Implicit Scope

When you open the Structure tab, the search mode is automatically turned on and the implicit search scope is used to filter out issues that don't belong to the current Project (or Component, or Version). This implicit scope is displayed with the *Project: project name* marker on the search panel.

If **More Issues** button is turned on, all issues from this project that are not part of the selected structure are displayed in the Search Results Secondary Panel. This allows to quickly place other issues in the project in the structure.

You still can use searching and filtering - but every search condition will be AND-ed with the scope condition. If you turn search mode off, full structure will be displayed.

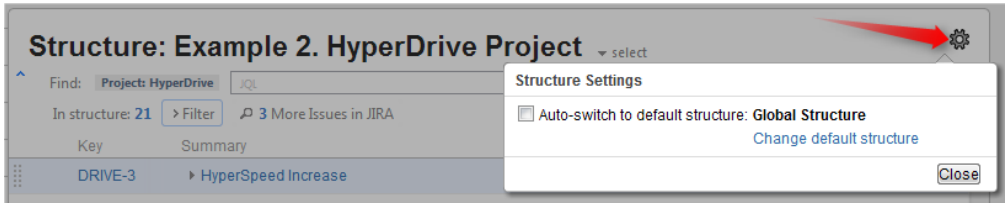
Views Menu

Click Views icon in the top right corner to open Views Menu and select another view for the displayed structure.

Views icon turns **blue** when the currently used view has been locally adjusted.

Structure Options

Similar to Structure Options for the Issue Page, there are options for the project page. Likewise, you click on the gear button at the top of the section to bring up the dialog. The changes are saved to the server and applied immediately.



Currently, the only option for the project page is **Auto-switch**. When it is turned **on**, the Default Structure for the displayed project is always shown initially. (After the page has loaded, you can switch to another structure using structure selector.) When auto-switch if **off**, the structure recently selected on the Structure Board or other pages (the *current* structure) is shown initially.

If you are the Project Administrator, the options dialog will also show the link to a page where you can change the default structure for your project.

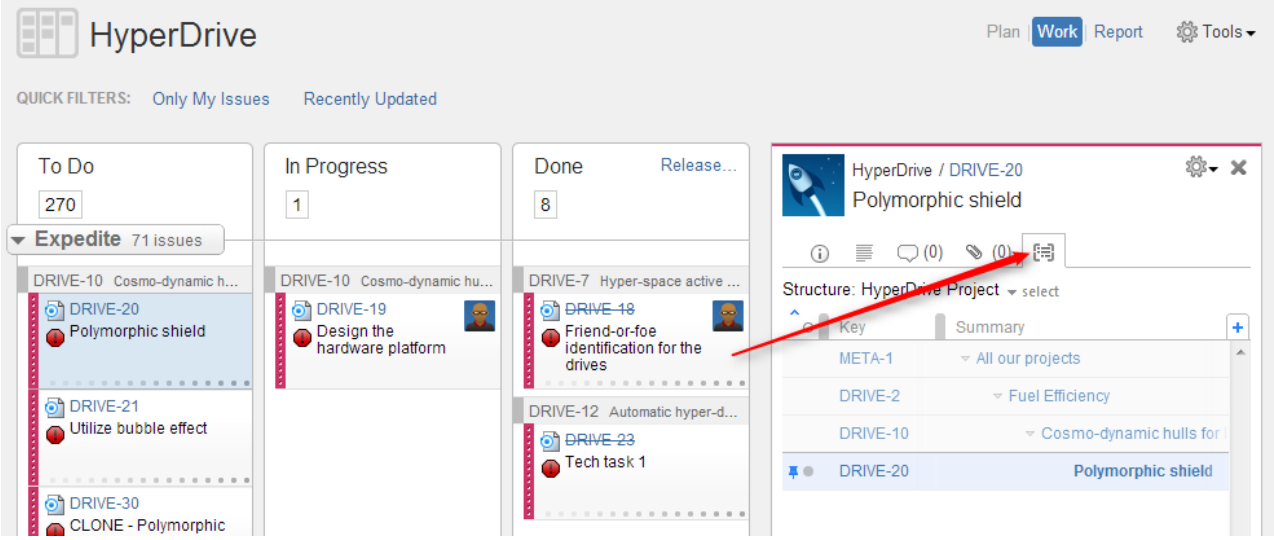
The default value for this option can be configured by JIRA Administrator on the Structure Defaults page.

Simplified View When There's Only One Structure

When there's only one structure that you can see, structure name, selector and options are irrelevant and are not displayed.

2.2.5 Structure on Agile Boards

If you have JIRA Agile (formerly GreenHopper) version 6.0 or newer installed, it will show additional Structure tab in the issue details panel on Scrum and Kanban boards.



Structure tab displays the standard Structure widget that lets you quickly identify, where in the structure is selected issue located. The widget is by default in Pinned Issue Mode, highlighting the position of selected issue and its sub-issues. You can un-pin selected structure by clicking Pin button on the toolbar or hitting Ctrl+.

Due to rather constrained horizontal space, Structure initially displays only Key and Summary in the Agile tab. However, you can add more columns if needed. (And get a larger display!)

When you click on another issue on an Agile board, Structure widget automatically selects that issue in structure, and, in pinned mode, pins that issue instead of previously selected.

You can switch to viewing a different structure using structure selector.

You can also edit all fields inline in the structure widget, if space allows. After editing is done, Structure signals JIRA Agile to reload the page and you will see the updated values on the board.

Only the users who have access to Structure will see the Structure tab on Agile boards.

2.2.6 Structure on the Issue Navigator Page

Viewing Structure Details of a Selected Issue

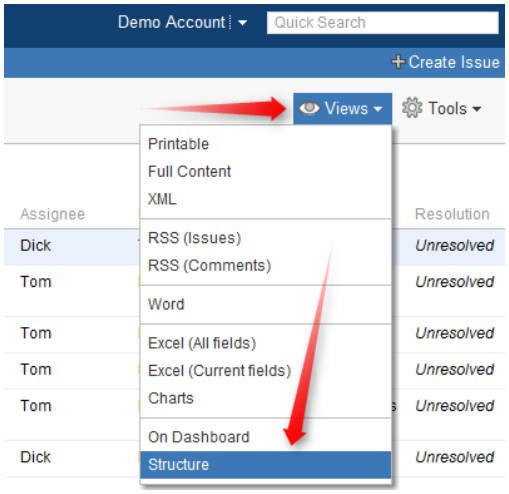
Starting with JIRA 6, search results on the Issue Navigator page can display the details of a selected issue. The details panel works in the same way as the separate issue details page.

See Structure on the Issue Page for more information on how Structure works there.

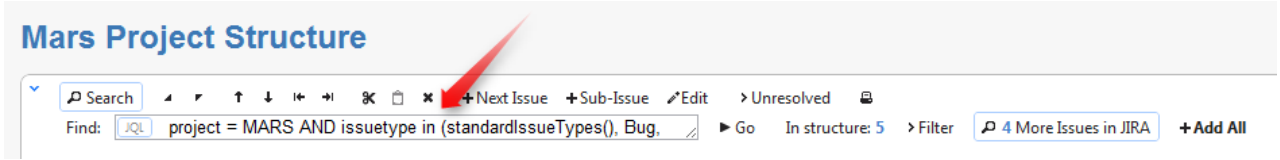
Opening Search Results on the Structure Board

When you use JIRA search to find specific issues, you can open search results on the Structure Board using Structure's integration with the Issue Navigator page.

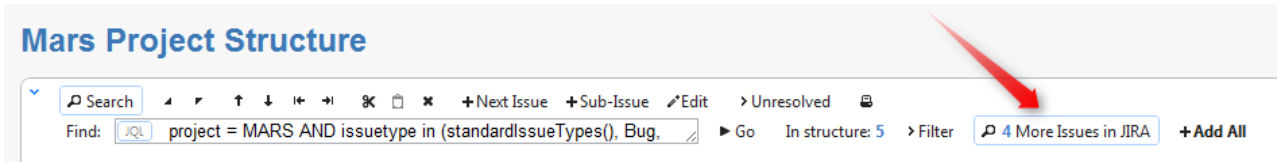
To do that, select **Views | Structure** menu (on JIRA 6 and later - **Export | Structure**), and that will get you to the Structure Board with the current structure, and with search mode automatically turned on, and the query used on the search page posted as the search term into the structure's search field.



The result is that you see the current structure, with issues from the search result highlighted (or if you have **Filter** turned on, the other issues are filtered out).



If some of the issues from the search result are not in the structure, you can see them if you turn on **More Issues** button.



Structure widget displays only issues from projects that are enabled for Structure. If search result contains issues from other projects, those issues are ignored.

When you open search results on the Structure Board, the view automatically changes to display the same columns as in the Issue Navigator. Turn off **Columns** button to get back to the usual view. Mode details: Using Issue Navigator Columns

2.3 Basic Concepts

Structure plugin lets you organize issues into multiple hierarchical lists. To get started, you need to get acquainted with a few basic notions we're using throughout this manual and in the Structure plugin itself.

Structure plugin	That's the product! We differentiate between Structure plugin and structures by writing plugin name with a capital letter.
a structure	A single hierarchical list of issues. By default, there's only one structure called "Global Structure", but you can create as many structures as needed.
a structure contains an issue	A structure is initially empty - it does not contain any issues. Issues can be added to a structure manually, or they can be imported or added automatically by a synchronizer. A single issue may be present in many structures at the same time, or not be present in any.
structure widget	The main tool for working with the structure - a grid with columns that shows the hierarchy and allows to work with it. That's what you saw on our live demo server. Structure widget is present at several places in JIRA.
a view	configuration of columns in the structure widget
a sub-issue is under a parent issue	When you place an issue under another issue in the hierarchy, you can say that it's a sub-issue and the other one is the parent issue. A sub-issue may contain sub-issues on its own (and so it's a sub-issue and parent issue at the same time). To make matters worse, an issue that's a sub-issue in one structure may not be a sub-issue in another structure.
children issues	Sometimes we call sub-issues "children issues", but "sub-issues" is the preferred term.
JIRA has sub-tasks	JIRA has a very limited hierarchy based on sub-tasks, and since Structure integrates nicely with the sub-tasks we need to refer to them as, well, "sub-tasks". So "sub-tasks" are JIRA issues of the specific issue types ("sub-task types"), while sub-issues are just any issues that happen to be placed under a parent issue in a structure.

issue – subissue relationship	Structure does not mandate what kind of relationship is between a parent issue and a sub-issue. It may be compositing (whole contains parts), planning (big task is done when smaller tasks are done), dependency (an issue can be addressed only if sub-issues are addressed first) – it is up to you. Structure provides user interface and tools that work great for most of the relationship types.

2.3.1 Default Structure

Default structure is displayed to the user when there's no specific structure selected - by default. Also, the user can tune Issue Page and Project Page to display the default structure when the page is loaded.

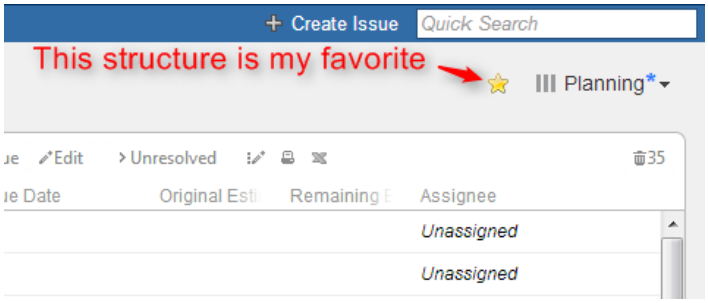
There's a **system-wide default structure**, which is initially set to be the automatically created *Global Structure*. JIRA administrator can change the system default structure.

Also, a **project-level default structure** can be specified for a project that is enabled for Structure. JIRA administrator can set project-level default structure on the Defaults page. Project administrator can change default structure on the project administration tab (JIRA 4.4 or later) or in the Options dialog on the Structure tab on the Project page.

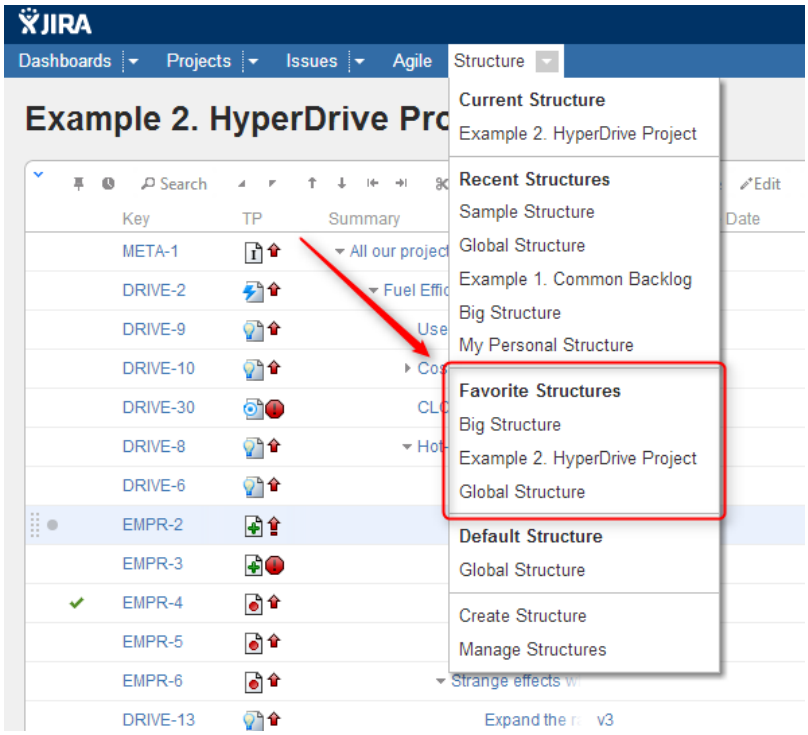
2.3.2 Favorite Structures

When you are logged in, you can mark one or more structures as your favorite, so you can quickly access them later.

To make a stucture your favorite, click on a grey star ( ) near the name of that structure. The star will turn yellow ( ) to indicate that the structure was added to the list of your favorite structures. The star icon is displayed near the structure name on the Structure Board, on the Manage Structure page and in a few other places.



Your favorite structures are shown in the Structure drop-down menu, so you can quickly switch to them:



To manage your favorite structures, use **Favorites** tab on the Manage Structure page.

Structure Popularity

Structure **popularity** is the number of users who have this structure marked as favorite. Manage Structure page has **Popular** tab, which shows most popular structures.

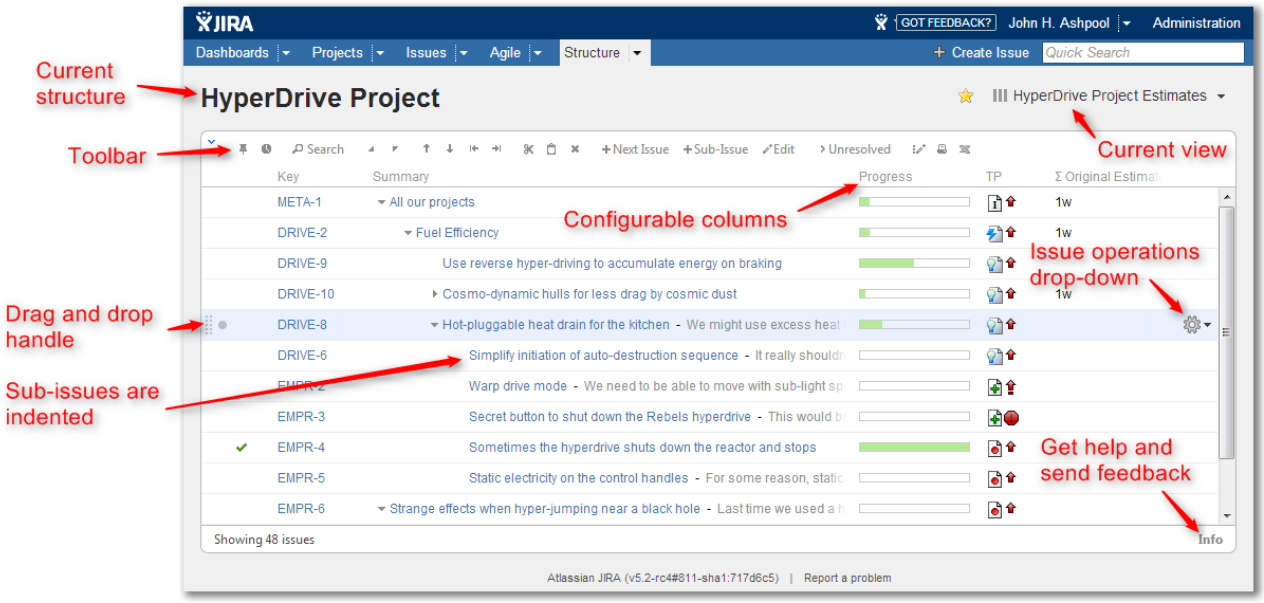
2.4 Working with the Structure Widget

Structure widget is the main tool for working with a structure and issues therein. It's a component of the Structure plugin that is used on the Structure Board, on the Issue Page and in other places where structure is displayed.

The following sections describe how to use the structure widget in detail.

- Structure Widget Overview
- Navigating Structure
- Structure Toolbar
- Configuring View
- Widget Columns
- Searching and Filtering
- Changing Structure
- Working with Issues
- Secondary Issue Panels
- Viewing History of a Structure
- Printing Structure
- Exporting Structure to XLS (Excel)
- Real-time collaboration

2.4.1 Structure Widget Overview



Structure widget is a grid with adjustable columns that displays the issues as a hierarchical list. Structure widget is displayed on the Structure Board, Issue Page and in other places in JIRA.

Structure lets you navigate the hierarchy and search for specific issues.

Besides showing the Structure and allowing to navigate it, structure widget is the primary tool to change structure by rearranging issues in the hierarchy or using JIRA actions to work with every issue.

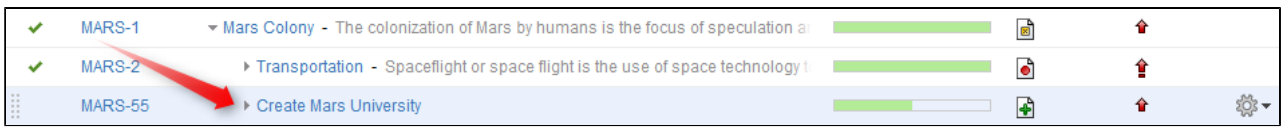
Next: Navigating Structure

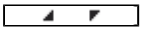
2.4.2 Navigating Structure

Navigating with Mouse

You can select issues, scroll up and down, as you would do with any table. Clicking on a link would usually open the link, so if you'd like to just select an issue, click anywhere in the row except on the underlined links. The row is also selected when you click on the JIRA actions icon at the end of the row.

To show sub-issues or hide sub-issues of an issue, click the **Expander** button near the issue summary.



To expand or collapse the whole hierarchy, use **Expand All** or **Collapse All** buttons  on the toolbar.

If there are many issues in the structure, not everything is loaded from the server. As you scroll down or expand sub-issue lists, the data is loaded on demand, which means there might be a delay before the grid is filled with the data for the displayed issues.

Navigating with Keyboard

You can use **arrow keys** to select next or previous issue in the list. Left and right arrows expand and collapse sub-issue list.

To expand all sub-issues, press the **Plus** keyboard button twice. To collapse all sub-issue lists, press twice the **Minus** keyboard button .

Using **Ctrl+Arrow Key** moves the selected issue up or down in the hierarchy or indents/out-dents it.

You can press **Alt+Down** to open JIRA actions menu for the selected issue.

There are a lot more keyboard shortcuts that let you work with the Structure almost without touching the mouse. Press **Ctrl+?** to see the shortcuts cheat sheet or click **Info** at the bottom of the structure widget.

Selecting Multiple Issues

Structure allows you to select multiple issues and do most of the operations with selected issues at once.

Usually, you navigate structures and select a single issue for further actions with keyboard arrows or mouse. The selected issue is highlighted with the blue background, and the actions (like moving) apply to the highlighted issue only.

Entering Multi-Select Mode

You can select multiple issues and switch Structure widget into multi-selection mode in one of the following ways:

- Press *Space* to add currently focused issue and move to the next issue.
- Click grey dot in the beginning of an issue row to toggle its selection.
- Hold *Shift* and use Up and Down arrows to select a range of issues.
- Hold *Shift* and use Right/Left arrows to select/deselect the focused issue with all its sub-issues.
- Hit *Ctrl+A* (*Command+A* on Mac) to select all issues.

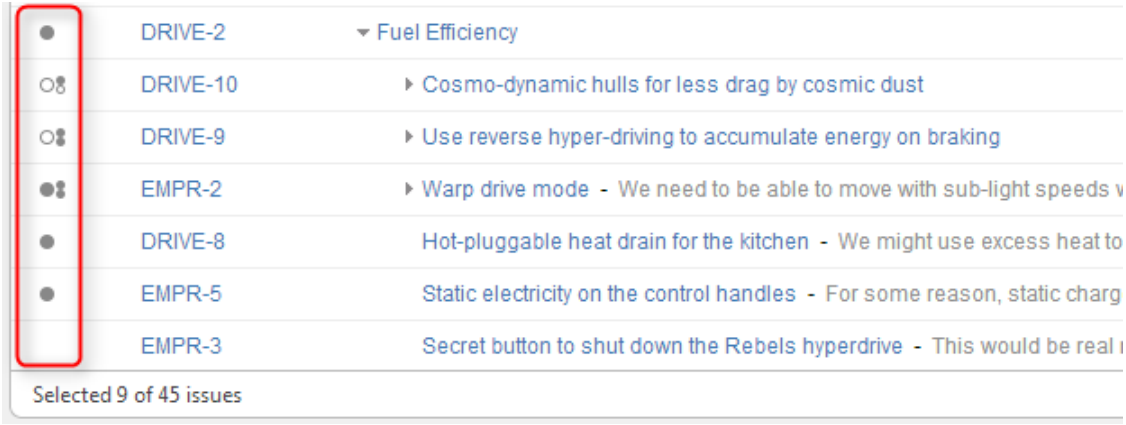
Selected issues are marked with a filled circle, and the bottom line shows the total number of selected issues.

Some of the issues are selected.

Special Selection Markers

If you collapse a list of sub-issues, and some of the sub-issues are selected, then the visible parent issue will display hints about whether it contains selected sub-issues.

For example, if you collapse sub-issues of *DRIVE-10*, *DRIVE-9*, and *EMPR-2* in the example above, you will see these selection markers:



Some of the issues are selected; some of the selected issues are collapsed under their parent issues. The meaning of the markers is the following:

	DRIVE-10	DRIVE-10 itself **is not** selected, but **some** of its sub-issues are selected
	DRIVE-9	DRIVE-9 itself **is not** selected, but **all** of its sub-issues are selected

	not on the example	the issue **is** selected, and **some** of its sub-issues are selected
	EMPR-2	EMPR-2 **is** selected, and **all** its sub-issues are also selected

Exiting Multi-Select Mode

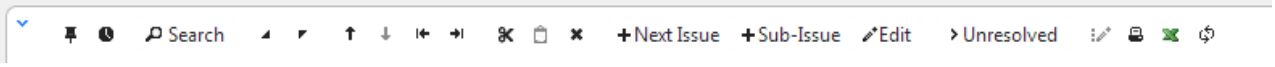
Hit *Escape* key to clear multiple selection and exit multi-select mode. You can also hit Ctrl+A (Command+A) twice – first key stroke will select all issues, second one will un-select all issues.

Restoring Selection After Navigation

If you navigate to a different page while having multiple issues selected and then return back, the selection will not be automatically restored. You can click **Restore Selection** link at the bottom of the Structure widget to select the same issues that were selected previously.

2.4.3 Structure Toolbar

The structure toolbar provides access to the main functions of the structure widget.



When you move mouse pointer over the toolbar, the buttons get darker. Some buttons may remain disabled (light grey) because the action cannot be carried out. For example, Bulk Change action is not possible unless you have selected several issues, and so Bulk Change button will remain light grey and not clickable in that case.

Once you move your mouse pointer over the toolbar button, a tooltip with the description of the action is shown.

Below is the table describing the set of actions available through the toolbar.

Button	Action	More Information	Keyboard Shortcut
	Pin / Unpin current issue.	Pinned Issue Mode	**Ctrl+.**
	Turn History View on and off.	Viewing History of a Structure	
Search	Turn Search on and off.	Searching and Filtering	**Alt+/**
	Expand/collapse the whole hierarchy.	Navigating Structure	**++** / **--**

Button	Action	More Information	Keyboard Shortcut
↑ ↓	Without changing the issue's parent, move the issue up/down and place it before/after the previous child - if possible.	Moving Issues Within Structure	**Ctrl+Up** / **Ctrl+Down**
⇤ ⇥	Unindent / Indent the issue one level, if possible.	Moving Issues Within Structure	**Ctrl+Left** / **Ctrl+Right**
✂	Cut the selected issues to the Issue Clipboard.	Issue Clipboard	**Ctrl+x** or **Command+x**
📋	Paste the issues from the Issue Clipboard into the structure.	Issue Clipboard	**Ctrl+v** or **Command+v**
✖	Remove the currently selected issues from the structure.	Removing Issues	**Delete**
+ Next Issue	Create an issue following the currently selected issue on the same level.	Creating New Issues	**Enter**
+ Sub-Issue	Create a sub-issue under the currently selected issue.	Creating New Issues	**Shift+Enter** / **Insert**
Edit	Edit the currently selected issue / stop editing and save changes.	Editing Issues	**F2** / **ss** / **Tab**
› Unresolved	Filter the structure to show unresolved issues only.	Searching and Filtering	**rr**
	Bulk change multiple issues.	Bulk Change	
	Open a printable page with the structure.	Printing Structure	
	Export structure to Excel format. (*On Mac OS X, this button is not shown unless switched on with **xx** keyboard shortcut.*)	Exporting Structure to XLS (Excel)	
	Turns on/off continuous updating of the Structure even if there's no user activity. (*This button is not shown unless switched on with **xx** keyboard shortcut.*)	Real-time collaboration	

You can hide/show toolbar clicking the arrow icon in the top left corner of the structure widget.

2.4.4 Configuring View

The way Structure Widget displays the structure and issues it contains is very much configurable.

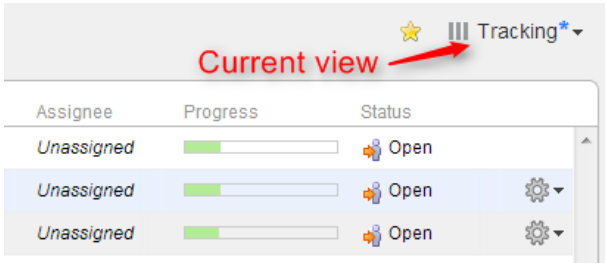
- You can configure how each issue is represented by customizing columns or selecting a pre-defined View.
- You can display only part of the whole structure that is relevant for some issue by pinning that issue.
- You can filter the displayed structure using text or JQL and display only the matching issues and their parent issues.

Using Views

A **view** is a visual configuration of the Structure Widget, which defines which columns are displayed and in what configuration.

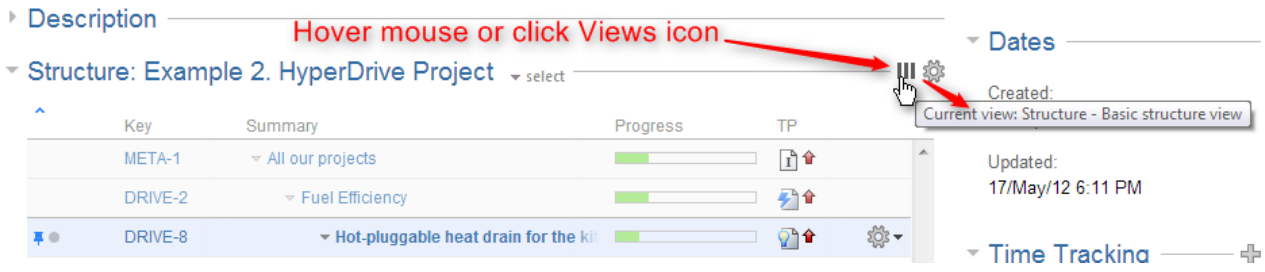
> Structure comes with a number of pre-installed views, but you can also define your own views - see Managing Views.

On the Structure Board, the currently view is displayed in the top right corner:



On picture: Current view is called "Tracking". The blue asterisk tells that the view has been adjusted by the user.

On other pages with structure the current view may be identified if you hover mouse over the Views icon:



You can change which columns are displayed by switching to another view or by manually adding, removing or rearranging columns.

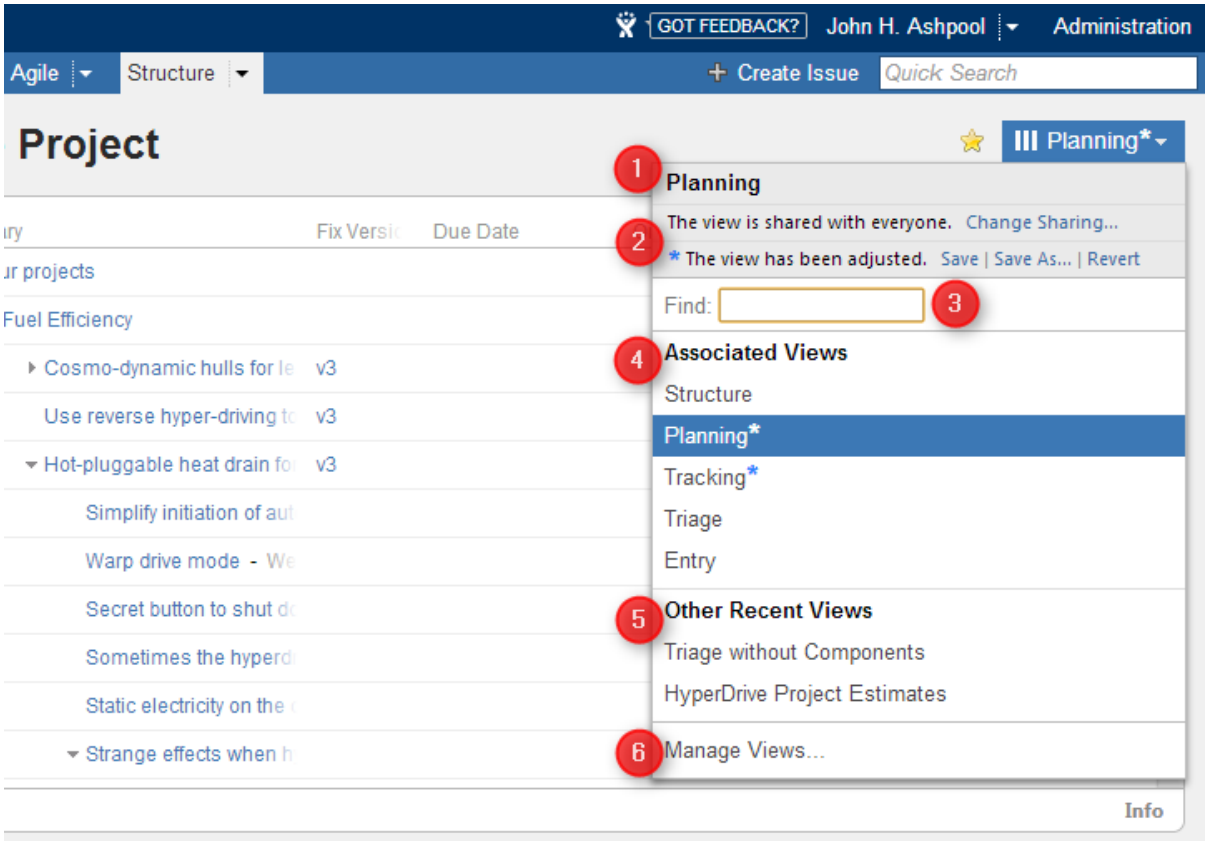
When you manually change column configuration, you create your local adjustments to the currently select view. You can then save the changes (if you have permissions to change the view) or save and share your customization as a new view – see Saving and Sharing Views.

Views Menu

You can switch to another view by pulling down Views menu and selecting one of the views it offers, or searching for a different view.

To open Views menu on the Structure Board, click current view name; on an issue page and project page click Views icon.

Besides a list of other views, the Views drop-down contains important information about the current view.



The following elements are numbered on the screenshot:

1. Current view name. Hover mouse over the name to see view description.
2. Important messages about the current view.
3. View search. Enter part of the view name to look for views on the server.

> View search looks for any view that matches the entered name, not only those in this list.

4. Associated views list. This list can be customized for each structure by the structure owner or anyone who has Control access level on the structure – see Customizing View Settings.
5. List of views you have recently used (excluding the views shown in the section above).
6. Manage Views link opens View Management dialog.

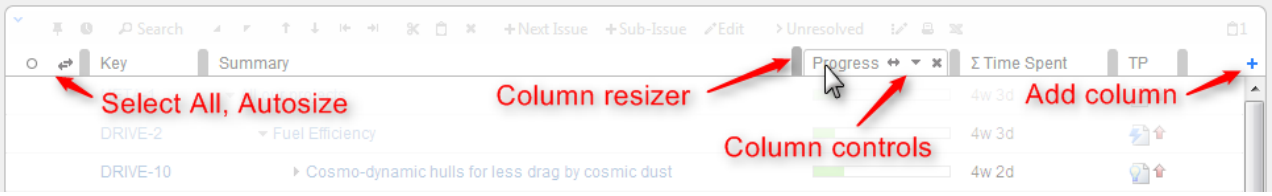
Switching View with Keyboard

You can switch current view only using keyboard:

1. Use **vv** shortcut to open Views menu (hit "v" twice);
2. Use arrows to select a view, or enter text to search for matching views;
3. Hit **Enter** to switch to selected view or **Escape** to close the menu.

Customizing Columns

To configure structure columns, position mouse pointer over the structure header for a second to have grid controls appear. These controls let you select which columns to show and how much space each column gets.



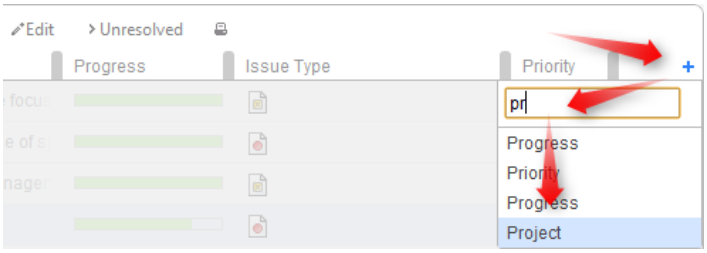
Structure header overview

> When you add, configure, remove or rearrange columns, you make adjustments to the *View* that's being used to display the structure. Adjusted view is marked with a blue asterisk (*). The adjustments are stored in your browser and affect only yourself – to make the changes persistent and available to others, you need to save the view or create a new view.

Adding Columns

To add a column, click on the **+** button at the right corner of the table header. A drop-down with available column presets appears. To select the desired column, you can:

- use the mouse to find a specific column, or
- use keyboard **arrow keys** to select the column and hit **Enter** when done, or
- start typing column name and get the column list filtered, then use arrow keys if needed and hit Enter when done.

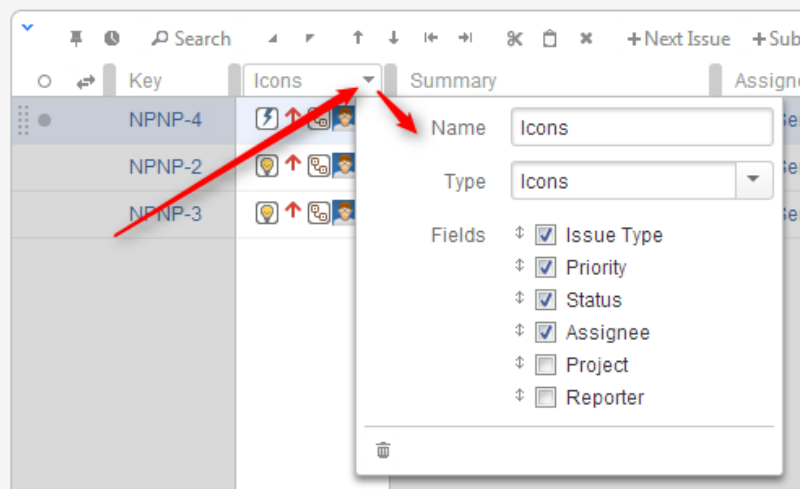


To abort adding new column, hit **Escape**.

Use keyboard shortcut **TT** to quickly open Add Column dialog (hit "t" twice).

Configuring Columns

To configure a column, click the arrow icon in the column header. The column will be highlighted and its configuration drop-down will appear, allowing you to change column name, type and other options.



The particular set of options available for the column is determined by its type. For example, the Field column type lets you select the issue field to display and enable aggregation for numeric and time-tracking fields.

Any changes you make are applied immediately, so you can see the effect almost instantly. When you are happy with the column, simply close the configuration panel by clicking the arrow icon again or clicking anywhere outside the panel.

To cancel all of your changes use the **"Revert"** button at the bottom of the configuration panel. The column will be restored to its original state.

Removing Columns

To remove a column, click the arrow icon in the column header, then use the **"Remove"** button at the bottom of the column configuration panel.

You cannot remove Summary Column and Special Columns.

Rearranging Columns

You can change position of a column by grabbing the column name with the mouse and dragging it to the left or right until the column position hint snaps into the desired location for that column.

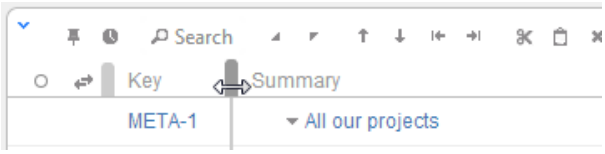
Resizing Columns and Autosize

Structure automatically tries to give all displayed columns enough space to display all information, but sometimes you might need to give more space to a column or two.

Column widths are not part of the *View* and are not saved on the server or shared.

There are a number of ways to change column widths:

- **Grab the resizer and drag.** When you hover your mouse over a column, the resizer that is responsible for that column's width grows darker and taller. When column size is close to what Structure considers ideal width (based on the displayed data), the resizer turns yellow and "snaps" to the perfect position.



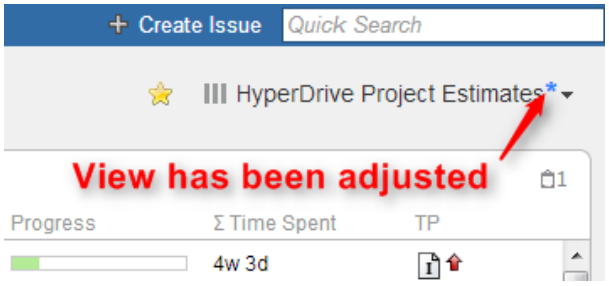
- **Hold CTRL and drag resizer.** Works same as above, but without snapping. You can use it to fine-tune column width.
- **Hold ALT (Option) and drag resizer.** In this mode, you will redistribute the space between two adjacent columns - increasing the width of one column and decreasing the width of another. (*This is how resizers used to work prior to Structure 2.0.*)
- **Double-click the resizer or the column header.** The column will automatically assume the best size.
- **Click "Autosize" icon (⇄) or double-click Summary column.** All columns will be resized automatically based on the displayed data.

Structure columns always take 100% of the horizontal space available to the structure widget, no more, no less. After all columns sizes are determined, Summary Column takes all the remaining width.

If the browser window is too narrow or the structure widget has too little horizontal space, the resizing may not work exactly as expected because it becomes problematic to accommodate all the columns with all the data. In that case, consider removing some columns or giving the browser window more width.

Saving and Sharing Views

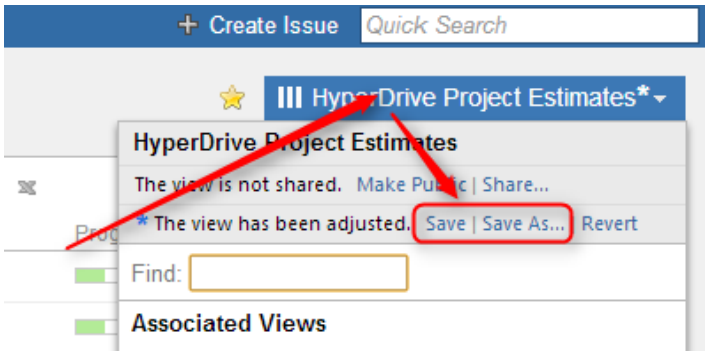
When you have added, removed or rearranged columns, you have adjusted the view that is used to display the structure. The view is marked as "adjusted" with the blue asterisk (on issue page, Views icon turns blue).



Saving View Adjustments

The adjustments you have made to the view are local, they are stored in your browser. To make the changes persistent and to push them to other people using the same view, you need to save a new version of the view. To do that, open Views drop-down and click **Save** link.

To save view changes you need to have *Update* access level for that view (see View Sharing and Permissions). If you do not have permissions to change the view, you can create a new view based on your modifications with **Save As** link.



If you need to remove your adjustments and get back to the original view as it stored on the server, click **Revert** link.

Sharing a View

A view has a set of permissions, just like a structure. When you initially create a view with **Save As** link, the view is **private** - it "belongs" to you and noone else can use it. You, however, can use this view with any structure.

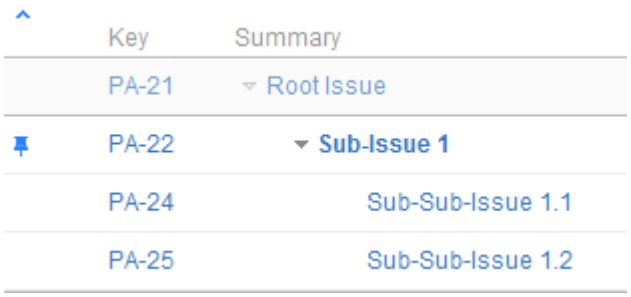
To share a view with other people, you can either make view **public**, allowing everyone to locate and use this view, or define more fine-grained permissions for the view.

To make current view public, click **Make Public** link in the view drop-down. After that, everyone will be able to find and use the view, but only you will be able to modify it.

To define fine-grained permissions or modify sharing, click **Share** or **Change Sharing** link in the Views drop-down to open View Management dialog. See View Sharing and Permissions for details.

Pinned Issue Mode

You can view only a part of a structure that is related to a specific issue, by pinning that issue with a **Pin** icon on the toolbar. Structure Panel on the Issue Page automatically pins the issue being displayed, so you only see the relevant part of the structure.



Sub-Issue 1 is pinned, only part of the structure related to Sub-Issue 1 is shown

What is Displayed in Pinned Issue mode

When the structure widget is in Pinned Issue mode, only the following issues are displayed:

- The pinned issue itself
- All parent issues of the pinned issue, up to the top-level issue - those are displayed above the grey line
- All sub-issues of the pinned issue, down to the deepest level

The issues that are "siblings" or located somewhere else in the hierarchy are not displayed.

The issues that are not displayed when an issue is pinned are not just filtered out, they are not loaded from the server, which provides quicker page load time.

Turning Pinned Mode On and Off

You can turn Pinned Mode on or off by clicking on the Pin button on the toolbar or by using **Ctrl+.** keyboard shortcut.

On the Issue Page and JIRA Agile (GreenHopper) Rapid Board page, you can only pin the issue currently viewed - you cannot pin any other issue from the structure. On the Structure Board, you can pin any issue.

Limitations Imposed by the Pinned Issue Mode

When you have Structure with a pinned issue, you can't change the hierarchy from the pinned issue upwards. That is, you can add/move/delete sub-issues of the pinned issue, but you can't add issues to the pinned issue's parents or move pinned issue somewhere else.

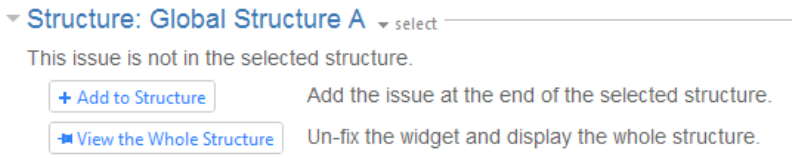
On the screenshot example above, the following actions are available for the displayed issues:

Issue	Relation to the Pinned Issue	Possible Hierarchy Changes
Root Issue	Parent	None
Sub-Issue 1	Pinned Issue Itself	Add sub-issues, Delete from structure
Sub-Sub-Issue 1.1	Sub-Issue	Any changes except moving to the top level or away from under Pinned Issue

Even though you can't move parent issues when the view is in pinned mode, you still can select them, edit or apply JIRA operations.

When Pinned Issue Is Missing from Structure

If it happens that the pinned issue is missing from structure, the structure widget will not be able to display any data and will ask for your action:



In this case you have to either unpin the view to see the whole structure, or add the pinned issue at the end of the structure.

2.4.5 Widget Columns

Structure widget provides a number of columns that display information about issues in the structure. You can customize the displayed columns by adding new columns, changing each column configuration, or switching to a new view.

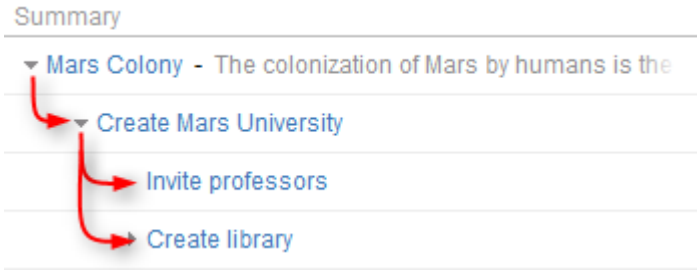
Out of the box, Structure provides the following columns:

- Summary Column
- Field Columns
- Icons Column
- Progress Column
  - Progress Based on Time Tracking
  - Progress Based on Resolution Only
  - Progress Based on Status
  - Progress Based on Percent Field
- Images Column
- Special Columns
  - Flags Column
  - JIRA Actions Column

Structure also contains extension API, so the selection of available columns may be extended by a third-party plugin.

Summary Column

The Summary column displays the issue summary and, optionally, part of the issue description. Sub-issues have the text in the Summary column indented relative to their parent issue.



Summary can be edited right in the structure widget and it's the only field required for creating new issues.

To turn off descriptions in the Summary column, use the column configuration panel.

Summary column cannot be removed from the Structure grid or reconfigured to a different column type because it displays the hierarchy.

Field Columns

For each issue field in your JIRA, Structure offers a column that displays that field's value.

Displaying Aggregate Values

For numeric and time-tracking fields, Structure also offers to display an aggregated value, calculated as a sum of the field values over sub-issues.

- To display an aggregated value, use the column configuration panel and select **Sum over sub-issues**. If there's no such option for a given field, then the aggregate cannot be calculated.
- Alternatively, you can add a predefined column from the **Totals** sub-section in the **Add Column** drop-down menu.

When aggregate value is displayed for an issue that also has an own value in the field, own value is displayed next to the aggregate value in grey color.

Note that values of the totals may change depending on the selected structure.

Aggregation is not available on secondary panels.

Editing Values

Most field columns are editable – you can edit field value by double-clicking it (if the field is added to the Edit Screen in JIRA). When aggregate value is displayed, you can still edit the issue's own value.

Icons Column

Icons column displays icons for issue type, priority, status, project, reporter, and assignee.

Its narrow width and short name allow to save horizontal space for other columns. You can configure which icons to display and arrange them in any order.

Progress Column

The Progress column displays configurable aggregate issue progress, which includes progress values from sub-issues.

Progress column allows you to customize how the progress is calculated – based on time tracking, Resolution and Status field, or custom fields. There are several predefined configuration of Progress column, available under Add Column menu. You can add any available preset configuration and then customize it using the column configuration panel (shown when you click the grey arrow in the column header).

Progress is the custom Structure column, not available in the Issue Navigator or other standard JIRA views.

How is Progress Calculated?

Configuration of the progress calculation is divided into two parts:

1. How individual issue progress is calculated, regardless of its position in the structure.
2. How progresses from sub-issues are aggregated and combined with individual progress of the parent issue.

Individual Issue Progress Calculation

There are several progress calculation modes. Mode is selected by the **Based On** option:

- Progress Based on Time Tracking — The progress is calculated based on the issue's Resolution field, time tracking data and the progress of sub-issues. Best estimate of the issue's completion is given, with extrapolation of the sub-issue estimates if needed.
- Progress Based on Resolution Only — The progress is calculated based on the issue's Resolution field and the progress of sub-issues.
- Progress Based on Status — The progress is determined by issue's Status field. Percentage values are assigned to specific statuses.
- Progress Based on Percent Field — The progress is assigned to each issue manually in a custom field, and aggregated for parent issues.

Total Progress Calculation

When individual issue progress is calculated based on Status, Percent Field, or Resolution Only, you can specify how individual sub-issue progresses are aggregated into parent issue progress. This is defined by **Weight** option:

- **All Sub-Issues Are Equal** – All sub-issues are considered equal when calculating aggregated progress for the parent issue. Weights do not accumulate, so sub-issues of each level are considered equal irrespective of how many sub-sub-issues they have.
- **Time Estimate** – Sub-issues' progresses are weighted proportionally to their total time estimate (*Time Spent + Remaining Estimate*). This option is akin to **Time Tracking**, yet allows to get individual progress from other sources (e.g. numeric custom field or Status field). If time information is not present, it is counted in as an average, based on the mean total time (time spent + remaining estimate).
- **Custom Numeric Field** – Sub-issues are weighted according to a value in the specified numeric field, for example, *Story Points*. Weights are accumulated upwards. If field value is not present, it is counted in as an average, based on the mean field value across sub-issues.

⚠ Zero value in the field configured as weight will discard any issue's progress in parent issue aggregation.

Progress Based on Time Tracking

The progress is calculated based on the issue's Resolution field, time tracking data and the progress of sub-issues. Best estimate of the issue's completion is given, with extrapolation of the sub-issue estimates if needed.

Calculating Progress for Issue Without Sub-Issues

If the issue does not have sub-issues:

- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100%.
- Otherwise, if the issue has time tracking information, the progress is calculated proportionally to this issue completion%: `(Time Spent) / (Time Spent + Remaining Estimate)`
- Otherwise, the progress is 0%.

Calculating Progress for Issue with Sub-Issues

If the issue does have sub-issues:

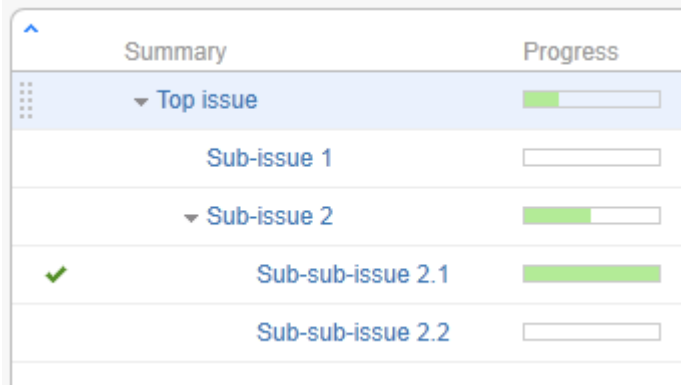
- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100% - regardless of the sub-issues progress.
- If the issue and its sub-issues do not have estimates or work logged (or if time tracking is turned off), the progress is calculated as the average from the sub-issues progresses.

- If time tracking is used and all issues have an estimate (either original estimate or remaining estimate) - the estimates and total work logged are summed up and the progress is calculated as the total completion%: `(Total Time Spent) / (Total Time Spent + Total Remaining Estimate)`
    - If a sub-issue does not have time tracking information, it is counted in as an average sub-issue, based on the mean total time (time spent + remaining estimate)

If the issue has both its own time tracking information and sub-issues with progress, and if **Ignore Parent Issue Progress** is turned off, issue's own progress value is counted as if it was the progress of one another sub-issue.

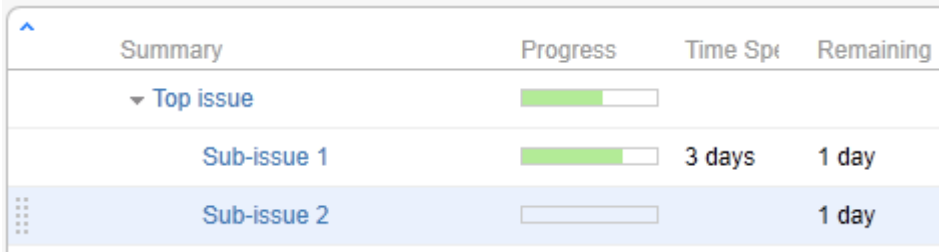
Examples

1. Example without time estimates



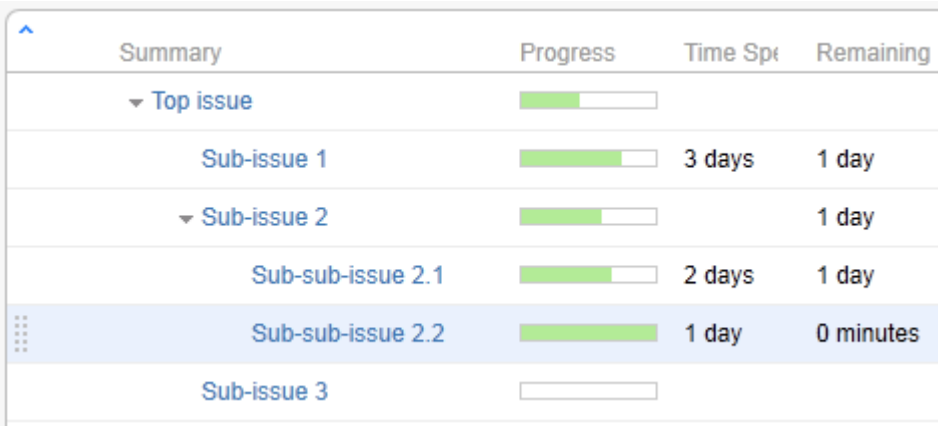
Issue	Explanation	Progress
Sub-sub-issue 2.1	This issue is resolved (indicated by the green mark) - so it is complete	100%
Sub-issue 2	It has two sub-issues with 100% and 0% progress, the total progress is average value	50%
Top issue	It has two sub-issues: sub-issue 1 is 0% done and sub-issue 2 is 50% done, the mean value is 25%.	25%

2. Example with time tracking information



Issue	Explanation	Progress
Sub-issue 1	It has 3 days of work logged with 1 day remaining, so its progress is `time spent / total time = 3 / (3 + 1)`	75%
Sub-issue 2	This issue does not have any work logged, is not resolved and does not have sub-issues	0%
Top issue	The top issue has total time spent of 3 days (work logged on sub-issue 1) and 2 total days remaining (estimates on sub-issue 1 and sub-issue 2), so its progress `3 / (3 + 2)`.	60%

3. More complex example



Issue	Explanation	Progress
Sub-sub-issue 2.1	It has 2 days of work logged and 1 day remaining, the progress is `2 / (2 + 1)`	66%
Sub-sub-issue 2.2	This issue has 1 day of work logged and no work remaining - so even though it is not resolved, it's considered completed	100%
Sub-issue 2	It has total time spent of 3 days, and total remaining estimate of 2 days (the remaining time from sub-sub-issue 2.1 and its own 1 day, which is considered additional work, besides sub-issues). The progress is `3 / (3 + 2)`.	60%
Sub-issue 1	This one has 3 days of work logged and 1 day remaining - the progress is `3 / (3 + 1)`	75%
Top issue	The progress of the *top issue* is calculated as follows. The obvious total time spent is 6 days, total remaining estimate is 3 days (count in all sub-issues on all levels). But there's also *sub-issue 3*, which does not have estimates or work logged, so it's estimated based on the average among the Top Issue's children issues - *sub-issue 1* and *sub-issue 2*: the average between total time of *sub-issue 1* (3 + 1 = 4 days) and total time of *sub-issue 2* (3 + 2 = 5 days) is 4.5 days. So *sub-issue 3* is treated as if it has total time 4.5 days (and given its 0% progress that's 0 days spent and 4.5 days remaining). That yields for	44%

Issue	Explanation	Progress
	the *top issue*: total time spent is 6 days, total remaining time is 7.5 days, and the progress is `6 / (6 + 7.5)`, which gives 44% value.	

Progress Based on Resolution Only

The progress is calculated based on the issue's Resolution field and the progress of sub-issues.

Calculating Progress for Issue Without Sub-Issues

If the issue does not have sub-issues:

- If the issue's Resolution field is not empty, the progress is 100%.
- Otherwise, the progress is 0%.

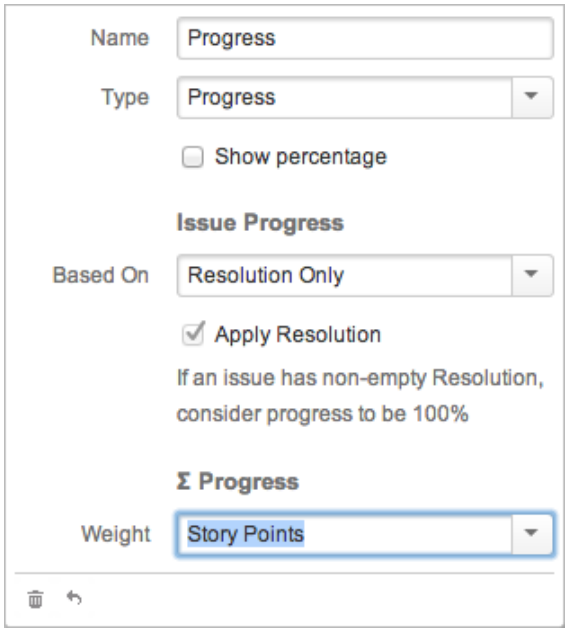
Calculating Progress for Issue with Sub-Issues

If the issue does have sub-issues:

- If the issue's Resolution field is not empty, the progress is 100% - regardless of the sub-issues progress.
- Otherwise, sub-issues progress is aggregated sum with specified weights.

Example: Resolution Only with Story Points

Individual progress is 0% or 100% based on Resolution field; total progress is calculated as weighted average, with weights contained in a *Story Points* field.



Column Configuration

Sample Structure

Issue	Explanation	Progress
Sub-sub-issue 1.2	This issue is resolved (indicated by the green mark) - so it is complete	100%
Sub-issue 1	It has two sub-issues with 0% and 100% progress, and story points are 2 and 3 respectively. So the total progress is weighted average value of (0 x 2 + 100 x 3) / (2 + 3)	60%
Top issue	It has two sub-issues: sub-issue 1 is 60% done and sub-issue 2 is 0% done, and their cumulative story points are (2 + 3) and 1 respectively. So progress is (60 x 5 + 0 x 1) / (5 + 1)	50%

Progress Based on Status

The progress is determined by issue's Status field. Percentage values are assigned to specific statuses.

Calculating Progress for Issue Without Sub-Issues

If the issue does not have sub-issues:

- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100%.
- If the issue's Status is assigned a value (%) in the column configuration, the progress is equal to that value.
- Otherwise, the progress is undefined, so the issue neither shows any progress, nor affects the progress of its parent issue.

Calculating Progress for Issue with Sub-Issues

If the issue does have sub-issues:

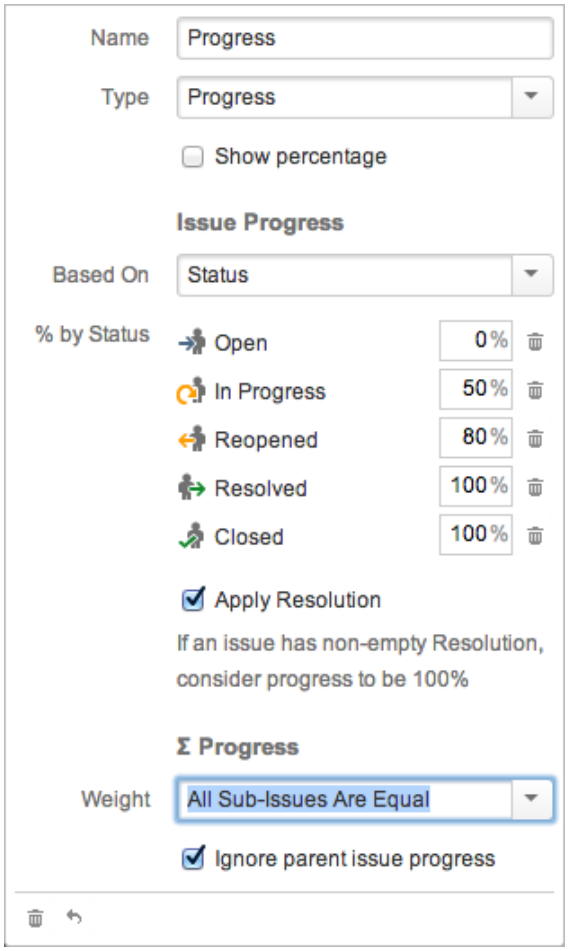
- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100% - regardless of the sub-issues progress.
- Otherwise, sub-issues progress is aggregated sum of progresses with chosen weights.

If the issue has both its own status and sub-issues with progress, and if **Ignore Parent Issue Progress** is turned off, issue's own progress value is counted as if was the progress of one another sub-issue.

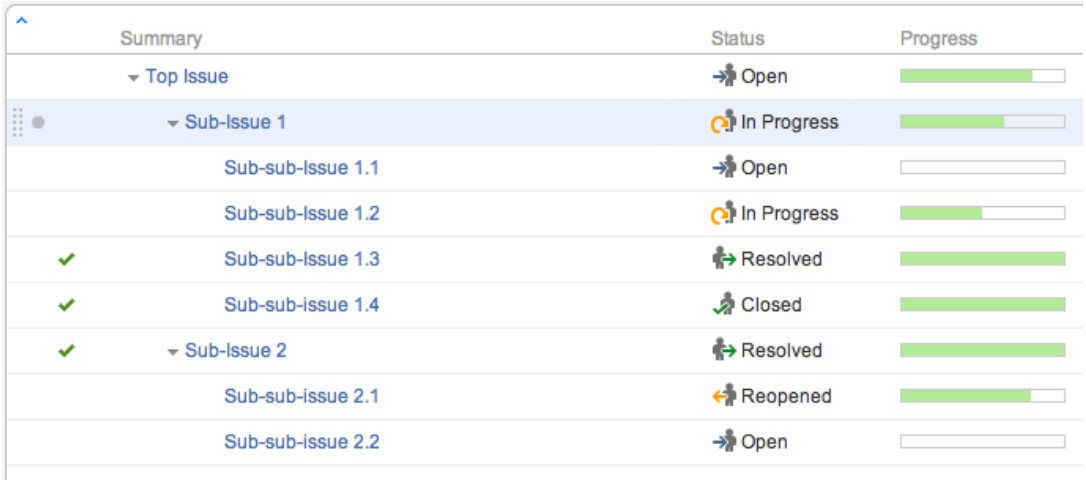
If some of statuses don't have any percentage configured, issue progress is considered undefined.

Example: Progress Based on Status, All Sub-Issues Are Equal

In this example, statuses have the following percentages: Open = 0%, In Progress = 50%, Resolved or Closed = 100%, Reopened = 80%. **Apply Resolution** is turned on, **Ignore Parent Issue Progress** is turned on.



Column Configuration



Sample Structure

Issue	Explanation	Progress
Sub-sub-issue 1.1	This issue is Open, so it is 0%	0%
Sub-sub-issue 1.2	This issue is In Progress, so it is 50%	50%
Sub-sub-issue 1.3	This issue is Resolved, so it is 100%. Also, according to workflow, it has non-empty Resolution, which also means it's complete.	100%
Sub-sub-issue 1.4	This issue is Close, so it is 100%. Also, according to workflow, it has non-empty Resolution, which also means it's complete.	100%
Sub-issue 1	Average progress is (0+50+100+100)/4. Issue's own status is In Progress, but it's percentage is ignored because of "Ignore parent issue progress in aggregation" option	63%
Sub-sub-issue 2.1	This issue is Reopened, so is 80%	80%
Sub-sub-issue 2.2	This issue is Open, so is 0%	0%
Sub-issue 2	Average progress is (80+0)/4 = 40%. But issue itself has Resolution and "Issues with Resolution are 100% done" option is turned on, so this overrides sub-issues progress and makes issue complete	100%
Top issue	It has two sub-issues: sub-issue 1 is 63% done and sub-issue 2 is 100% done. Average progress is (63+100)/2	82%

Progress Based on Percent Field

The progress is assigned to each issue manually in a custom field, and aggregated for parent issues.

You can use any numeric JIRA custom field to store the current progress % – a value from 0 to 100.

Calculating Progress for Issue Without Sub-Issues

If the issue does not have sub-issues:

- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100%.
- If the issue's Custom Field value is not empty and is between 0 and 100, it's considered as the completion progress in percents.
- If the issue's Custom Field value is less than 0, the progress is 0%, if greater than 100, the progress is 100%.
- Otherwise, the progress is undefined, so such issue neither shows any progress, nor affects progress of its parent issue.

Calculating Progress for Issue with Sub-Issues

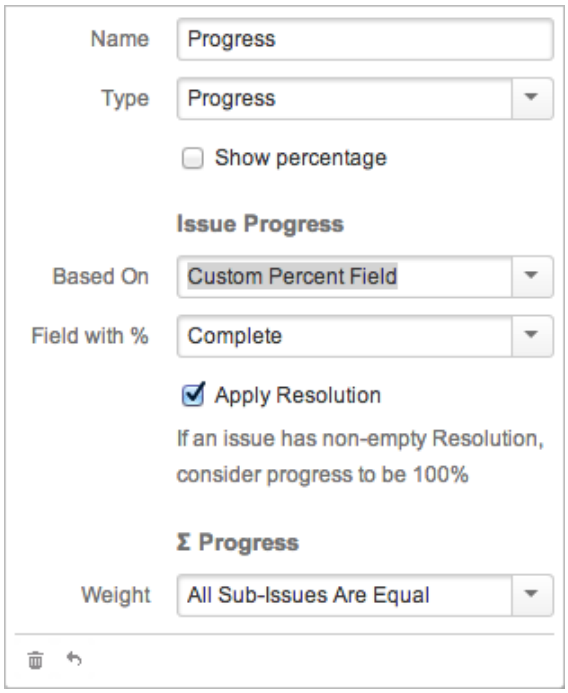
If the issue does have sub-issues:

- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100% – regardless of the sub-issues progress.
- If the issue's Custom Field value is not empty, it's considered as that issue's completion progress in percents (from 0 to 100) – regardless of the sub-issues progress.
- Otherwise, sub-issues progress is aggregated sum of progress with chosen weights.

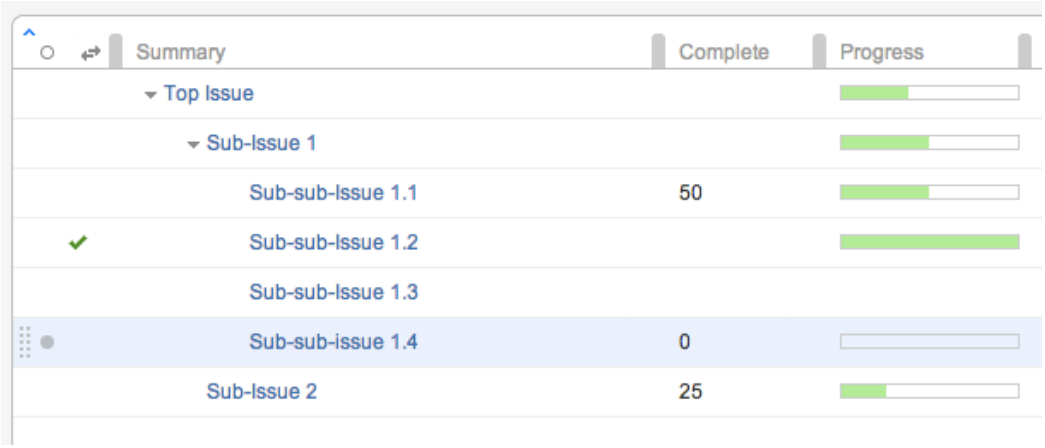
Examples

A: Percent Field, All Sub-Issues Are Equal

Custom field named *Complete*, total progress is based on **All Sub-Issues Are Equal**, and **Apply Resolution** is turned on.



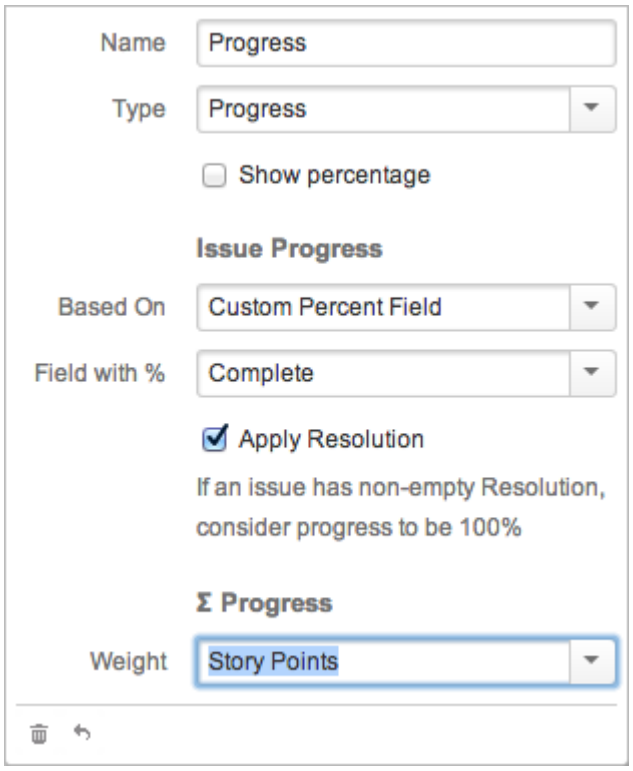
Column Configuration



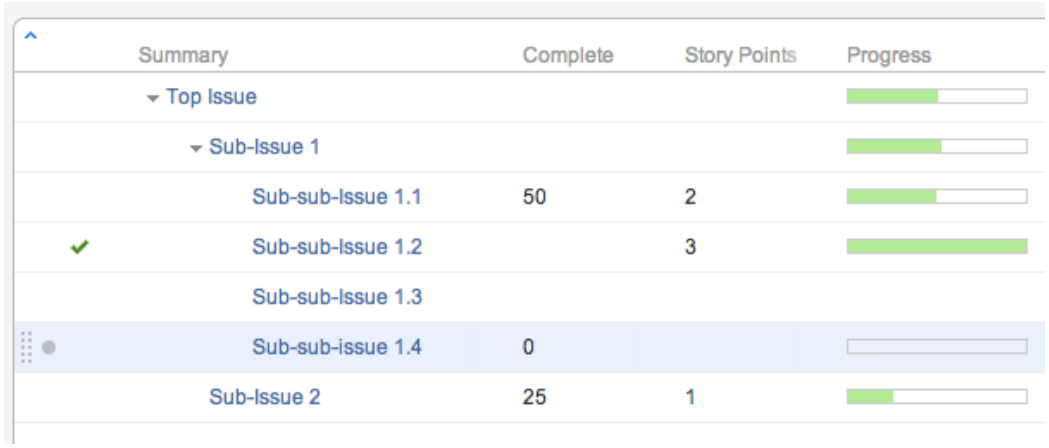
Sample Structure

Issue	Explanation	Progress
Sub-sub-issue 1.1	This issue is 50% complete as specified by custom field	50%
Sub-sub-issue 1.2	This issue is resolved (indicated by the green mark) - so it is complete, even if "Complete" field is empty	100%
Sub-sub-issue 1.3	This issue has no progress information (neither "Resolution" nor "Complete" fields), so progress is undefined and not counted at all.	n/a
Sub-sub-issue 1.4	This issue has 0 "Complete" value, which means it's 0% complete	0%
Sub-issue 1	It has two four sub-issues, but 1.3 is ignored. So the total progress is average of the rest: (50 + 100 + 0) / 3	50%
Sub-issue 2	The issue is 25% complete as specified by custom field	25%
Top issue	It has two sub-issues: sub-issue 1 is 50% done and sub-issue 2 is 25% done. So the progress is average between two (25 + 50) / 2	38%

B: Percent Field, Story Points
Custom field named *Complete*, total progress is based on the field *Story Points*, and **Apply Resolution** is turned on.



Column Configuration



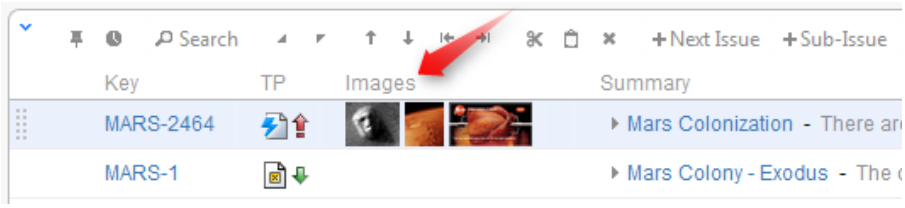
Sample Structure

Issue	Explanation	Progress
Sub-sub-issue 1.1	This issue is 50% complete as specified by custom field and has 2 story points	50%
Sub-sub-issue 1.2	This issue is resolved (indicated by the green mark) - so it is complete, even if "Complete" field is empty and has 3 stroy points	100%
Sub-sub-issue 1.3	This issue has no progress information (neither "Resolution" nor "Complete" fields), so progress is undefined and not counted at all.	n/a
Sub-sub-issue 1.4	This issue has 0 "Complete" value, which means it's 0% complete. It has no story points, so it's counted as mean of 2 and 3 = 2.5	0%

Issue	Explanation	Progress
Sub-issue 1	It has two four sub-issues, but 1.3 is ignored. So the total progress is weighted average of the rest: (50 x 2 + 100 x 3 + 0 x 2.5) / (2 + 3 + 2.5)	53%
Sub-issue 2	The issue is 25% complete as specified by custom field and has 1 story point	25%
Top issue	It has two sub-issues: sub-issue 1 is 53% done and sub-issue 2 is 25% done. So the progress is calculated as (53 x 7.5 + 25 x 1) / (7.5 + 1)	50%

Images Column

Images column displays small thumbnails of the attached image files and allows to view those images in a pop-up dialog.



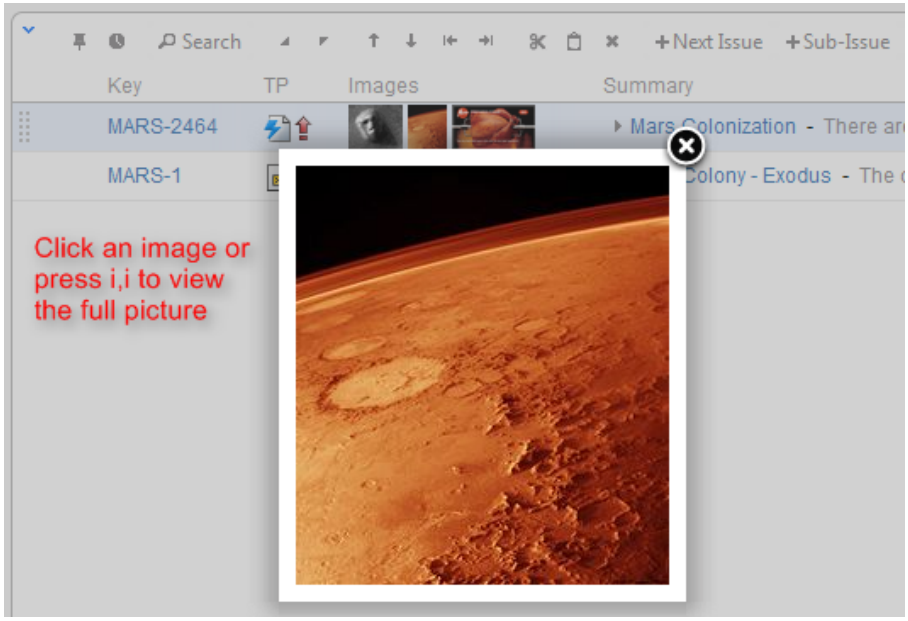
Viewing Full-Size Images

Using your mouse:

1. Click the image thumbnail to see the full-size image in a dialog box.
2. Click on the left or right side to view the previous or next image.
3. Click on the close button at the top right corner to close full-size image view.

Using your keyboard:

1. Select the issue that contains images.
2. Press **i,i** ("i" twice) to view the first image.
3. Press and to go to the next or previous image.
4. Press **Esc** to close full-size image view.



Images from Wikipedia

Special Columns

Some columns in the Structure widget are special. They either display structure-specific information or allow you to perform actions with the issues in the structure.

- Flags Column — The flags are the small icons displayed at the left side of the table to mark specific issue states.
- JIRA Actions Column — The JIRA actions column displays the gear button that calls out the menu with available JIRA actions for the issue.

Flags Column

The flags are the small icons displayed at the left side of the table to mark specific issue states.

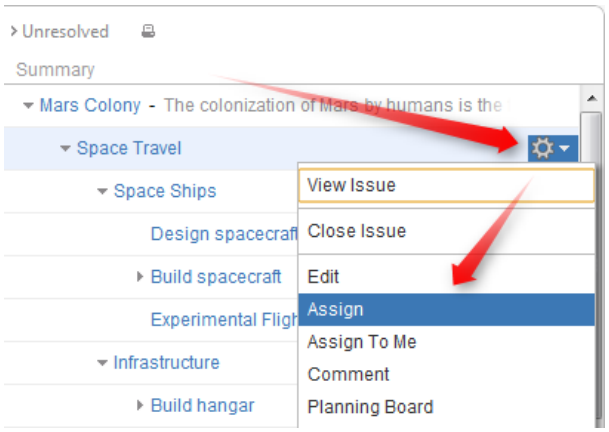
Structure displays the following flags:

✔	**Resolved** flag means that the issue's Resolution field is not empty. Such issue is considered completed and filtered out by the Unresolved button.
🔒	**Read-only** flag means that the current user does not have **Edit Issue** permission on this issue, so you cannot edit this issue. Additionally, if the structure is configured to require Edit Issue permission on Parent Issue, you cannot change or rearrange the immediate children of this issue.

JIRA Actions Column

The JIRA actions column displays the gear button that calls out the menu with available JIRA actions for the issue.

This column works like the similar column on the JIRA's Issue Navigator page and lets you log work, apply workflow actions and use other JIRA actions available for the issue.



You can click the gear button and select the desired action with the mouse, or you can use keyboard shortcut **Alt+Down Arrow** to open the menu for the currently selected issue and then use **Up** and **Down** arrow keys and **Enter** key to select the action.

2.4.6 Searching and Filtering

The Search feature provides several important functions:

- Find and highlight issues in your structure,
- Filter your structure so that it only displays specific issues,
- Find issues outside the structure and add them to the structure on the spot.

To access Search function, click the **Search** button on the Structure Toolbar.

The Search area will appear below the toolbar. By default, the Search looks for the entered text in issues' summary.

The search starts once you start entering the query, refining results as you keep typing. The latest search query is saved automatically, so even if you navigate to other pages, the search query will still be there when you return.

Search function allows you to search for issues in Simple, JQL, and S-JQL modes, within and outside the Structure. You can also switch to Issue Navigator columns when displaying search results. For more details on this functionality please refer to the corresponding sections.

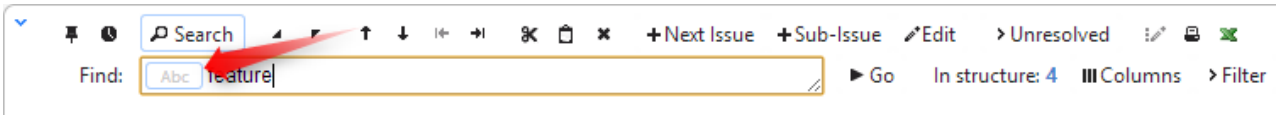
In Simple and JQL modes, if data changes on the server, search results are automatically refreshed for the structure. So issues can be hidden and shown in the structure in realtime.

However, the search results for issues outside structure are not automatically refreshed. Also, results for S-JQL search within the structure are not refreshed. You can make sure that you are viewing the up-to-date search results by clicking the **Go** button next to the **Find** field.

You can turn searching on by pressing **Alt+/**. You can cancel search and close the search panel by pressing **Escape**.

Simple, JQL, and S-JQL Search

In the Search Area, you can specify a simple text condition, a JQL condition, or a Structure query. To switch between these search modes, click the Search Mode button or press **Alt+j**. The button displays the currently selected mode.



Simple Search

Simple search mode is selected by default (the button reads "Abc".) In this mode, you can specify the following search conditions:

Condition Type	Example	How it works
Simple text	*structural hierarchy*	Look for issues that have **all** mentioned words in the **Summary** field. Each word in the search sentence must be present in the summary, or the summary must have a word that **begins with** the specified word. The words may come in any order.
Quoted excerpt	*"the quick brown fox"*	Look for the whole phrase in the summary (but see below about Lucene indexes).
Issue keys	*MARS-1, MARS-331*	If the text looks like one or more issue keys (delimited by comma or whitespace), search for exactly these issues.
All issues	*	Use single asterisk to search for "all issues". Only issues from the projects enabled for Structure are found. This can be useful when searching outside structure.

Structure relies on the JIRA search engine to run text searches. The engine is based on Lucene index which has a few peculiarities that may cause unexpected results. For example, short words may not be found. The result also depends on the Indexing Language specified in the JIRA General Configuration.

JQL Search

In the JQL mode, the search condition is treated as a JQL (JIRA Query Language) query. JQL lets you specify arbitrarily complex conditions to find very specific issues.

When the JQL mode is on, the usual JQL auto-complete suggests fields, operators and values as you type. Whenever you have a correct JQL in the search field, the Search Mode button will have white background. When the JQL is incorrect or not complete, the Search Mode button will turn red.

More information on JQL is available in the JIRA documentation.

S-JQL Search

In the S-JQL mode, the search condition is treated as a Structure query. S-JQL is a special language that allows to search for issues by their relations in the current structure, e.g., `root` matches all top-level issues, `root or child of root` matches first two levels, and `child of [priority = Critical]` matches all children of critical issues. See S-JQL documentation for more information.

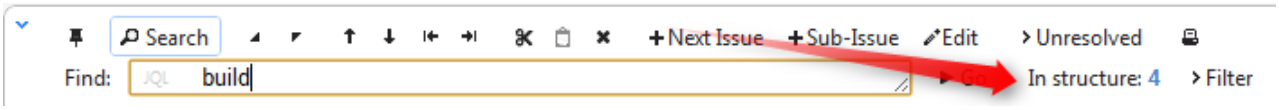
Unlike other search modes, you can only search for issues within structure using S-JQL search. Also, S-JQL search is not automatically updated when issues are changed in JIRA; click the **Go** button to ensure you have the up-to-date results.

As with the JQL mode, the Search Mode button will turn red if the specified S-JQL query is incorrect or incomplete.

Searching and Filtering Within Structure

Searching within the Structure

When the structure widget gets search results from the server, the number of matching issues found in the structure is displayed next to the **Find** field.



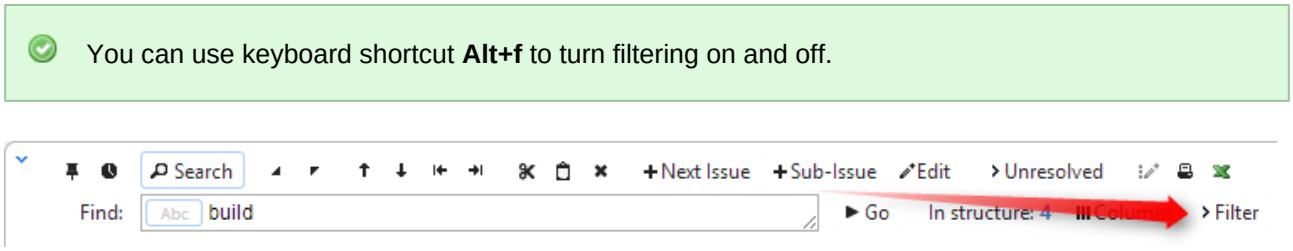
Structure widget grays out non-matching issues in the structure in order to highlight the matching issues.

The selection is also moved to the first matching item, and if you press **Down/Up Arrow Keys** while the input focus is still in the search field, the selection will go to the next/previous matching issue in the structure.

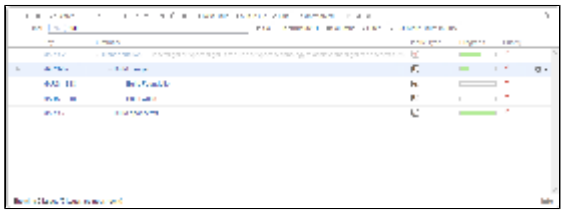
If you click somewhere else and the input focus is no longer in the search field, you still can navigate to the next or previous matching issue by pressing **]** or **[**.

Filtering Structure

If you wish to see only the issues that match the criteria specified in the search field, click the **Filter** button next to the issue count.



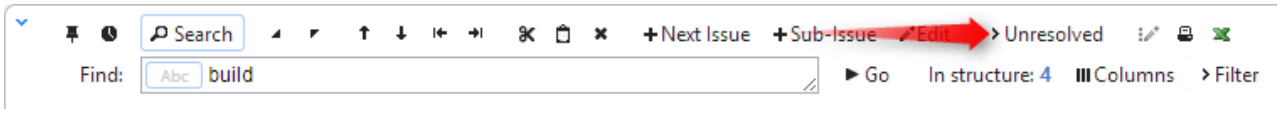
Once Filtering is turned on, you will only see the matching issues and their parent issues. (Parent issues of a matching issue are always shown to preserve the hierarchy view, even if they don't match the search criteria. Non-matching issues are grayed out.)



Filtering mode remains even if you navigate to another page.

Showing Unresolved Issues Only

Structure toolbar has **Unresolved** button, which works as a shorthand for filtering using JQL: *Resolution is EMPTY*. Clicking **Unresolved** button would filter the structure in the same way Filtering would do.



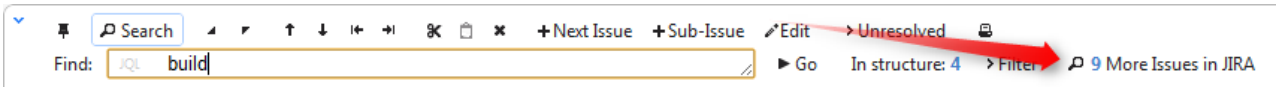
You can turn on Unresolved button and use search or filtering at the same time.

Press **RR** ("r", then quickly "r" once again) to turn Unresolved filter on and off.

Searching Outside Structure

Searching for Issues Outside Current Structure

If there are matching issues that do not belong to the structure, an additional **More Issues** button is displayed on the search panel, telling exactly how many matching issues are there in JIRA that are not in the structure.



You can turn **More Issues** mode on or off by clicking this button. When it is turned on, the extra issues are displayed in the JIRA Search Results secondary panel.

The search is run through all projects for which the Structure plugin is enabled (see Selecting Structure-Enabled Projects).

You can turn **More Issues** mode on and off with keyboard shortcut **Alt+m**.

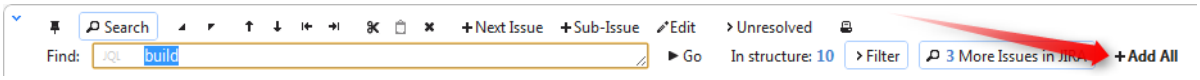
If the search results is large, only the first 1,000 issues are displayed in the Search Results Panel. If you add those issues to the structure, the next 1,000 issues are pulled. Use `ORDER BY` JQL phrase to sort the issues in the result to have the most important issues come first.

Adding Issues to Structure

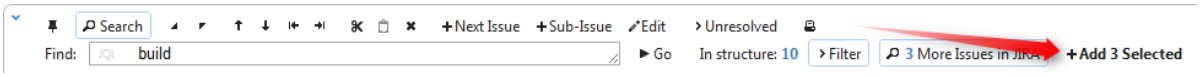
When you have issues found outside Structure with the **More Issues** mode on, you can add them from the Search Results secondary panel to the structure.

There are several options:

- Using Drag-And-Drop, move issues from the Search Results Panel to a specific location in the structure.
- Select several issues or all of them using check boxes and use drag-and-drop to move all the selected issues to the structure.
- Select an issue in the structure and click **Add All** button to place all issues from the Search Results Panel **after** the selected issue.



- Select an issue in the structure, then select several issues in the Search Results Panel and click **Add N Selected** button to place the selected issues from the Search Results Panel **after** the selected issue.



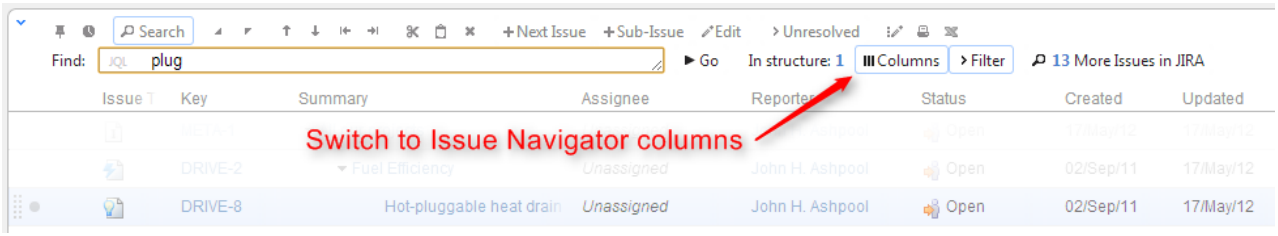
- While still having input focus (input caret) in the search criteria field, hit **Ctrl+Enter** to add all (or only selected) issues from the Search Results Panel after the currently selected issue in the structure. (Similar to Add All button.)
- Or hit **Ctrl+Shift+Enter** to add all (or only selected) issues from the Search Results Panel **under** the currently selected issue in the structure - the selected issue becomes the parent of the newly added issues.
- Use Cut/Paste actions and Issue Clipboard.

In Pinned Issue Mode, issues can be added only as sub-issues of the pinned issue. The result of using **Ctrl+Enter** or other means of adding issues to structure may be automatically adjusted to place the new issues as sub-issues of the pinned issue.

Using Issue Navigator Columns

Structure can display the same columns as the Issue Navigator when showing search results.

You can toggle Issue Navigator columns by clicking **Columns** button in the Search toolbar.



When Issue Navigator columns are turned on, Structure temporarily switches to a system view that displays the same columns as Issue Navigator would display for that query. To switch back to your usual view, switch off the **Columns** button or switch off **Search** panel.

A separate columns configuration may be defined for a Saved Filter. When you use a saved filter in your JQL query (`filter = "filter-name"`), Structure detects that and will display columns defined for that filter. (If **Columns** button is turned on, of course.)

If you have a **Progress** column in the Issue Navigator, then the Structure's **Progress** column will be shown instead. The same applies to aggregate columns like **Original Estimate**. Please keep in mind that these Structure columns and their JIRA counterparts work by different rules, and may display different values.

2.4.7 Changing Structure

There are several basic operations you can do with a structure. They include:

- Adding issues that exist in JIRA to the structure;
- Moving issues within the structure;
- Removing issues from the structure.

There are several ways to make these changes. Some of these operations can be applied to a group of issues and some to individual issues only. See the respective subsections for more details.

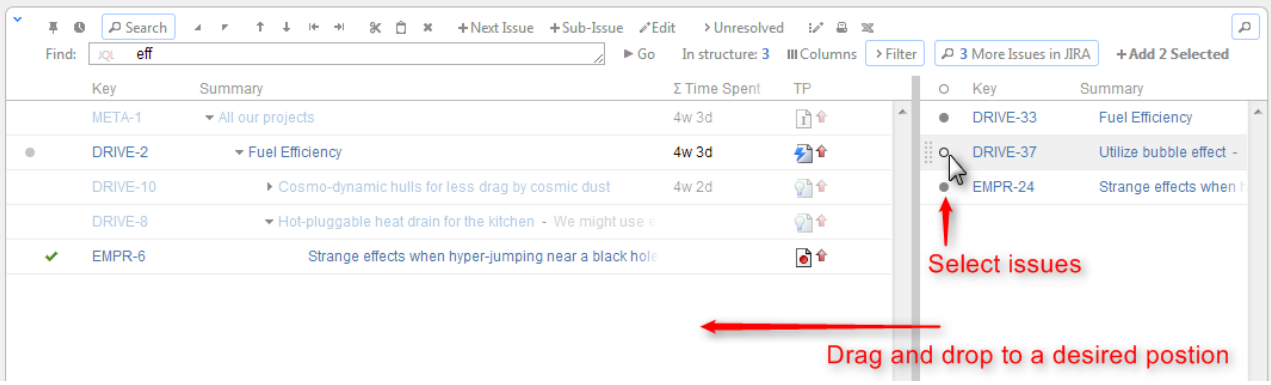
See also: Creating New Issues

Adding Issues to Structure

You can add an issue to a structure both from the Structure Board and from the Issue Page.

On the Structure Board, use Search to find the desired issues and add them to a structure using drag-and-drop, copy/paste, or the **Add** button on the search panel.

When searching, make sure the JIRA Search Results secondary panel is switched on. Use **More Issues** button on the search panel.



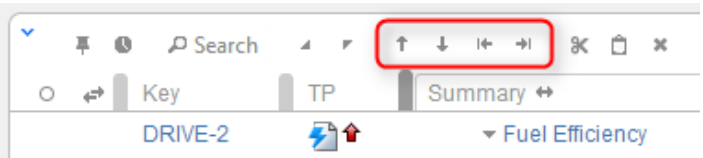
On an Issue Page, if the issue you are viewing is not in the current structure, the Structure section contains Current Issue secondary panel and you can drag the issue to the current structure from there. You can also select a different structure using Structure Selector. Besides, you can use Search on the issue page as well and add other issues to the structure.

Adding issues to a structure can be undone.

Moving Issues within Structure

Basic Moves

There are four basic operations that change structure. All of them are available on the toolbar, and they also can be done from keyboard. Hover mouse pointer over the operation button in the toolbar and a tooltip with the keyboard shortcut will appear.



Operation	Keyboard Shortcut	What it does
Move Up	Ctrl + Up	Without changing the issue's parent, moves the issue up and places it before the previous child - if possible.
Move Down	Ctrl + Down	Without changing the issue's parent, moves the issue down and places it after the next child - if possible.
Level Up / Unindent	Ctrl + Left	Makes the issue follow its current parent. The new issue's parent is the previous parent's parent. (Confusing enough? Simply speaking, you're moving the issue one indent level to the left.)
Level Down / Indent	Ctrl + Right	Move the issue to be a sub-issue of its current preceding sibling. You guessed it, it's one indent level to the right.

When you move an issue that has sub-issues, the whole sub-tree is moved.

When you make changes in the structure, they are uploaded to the server asynchronously, allowing you to continue working regardless of the network delay. You can do a rapid succession of the basic moves, for example, regardless of the time it takes to effect these changes on the server side. There's a **synchronizing** icon in the widget status bar that tells whether there are pending uploads or downloads.

Moving an Issue to an Arbitrary Position

The basic moves can only adjust issue position one place at a time, so if you need to place an issue at a specific position not close to its current position, you can do that with Drag-and-Drop or Cut & Paste. Cut & Paste also allows to copy hierarchy from one structure to another.

Moving issues with Drag-and-Drop or Cut & Paste can be undone.

Multiple Selection

You can select multiple issues and move them all together in one action. Move Up/Down and Level Up/Down support moving multiple issues only if they are at the same level in the hierarchy and under the same parent. Drag & Drop and Copy & Paste support multiple issue selection in any configuration.

See also: Selecting Multiple Issues

Removing Issues from Structure

To remove an issue from the current structure, select this issue and press **Delete** button on the keyboard or click **Delete** button on the toolbar. The issue is removed with all its children issues.

You can select multiple issues and remove them all in one action.

The issue will be removed from the structure and placed into the Removed Issues secondary panel. You can access it by clicking the **Trash** icon in the top right corner. To move removed issues back into the structure, simply select them in the Removed Issues panel and use Drag-and-Drop or Copy & Paste to place the issues in the structure.

Once you leave the current page, the Removed Issues panel is emptied.

Removing an issue from a structure does not delete the issue itself. It just removes it from the current structure.

Removing issues can be undone.

Changing Multiple Issues

You can apply most of the changes to multiple issues in one action. Select multiple issues and use toolbar, keyboard shortcut or drag and drop.

Some actions may have limitations of applicability when multiple issues are selected. For example, if you select both a parent issue and a sub-issue, the "Unindent" action will not be possible.

The following actions work with the multi-selection:

- Drag and drop lets you move a selection of issues within a structure or add them to a structure from the secondary issue panels, such as Issue Clipboard or Search Results.
- Cut and paste allow you to move issues both within a structure and between different structures.
- Remove button or *Delete* key lets you remove multiple issues from the structure.
- Toolbar buttons *Move Up*, *Move Down*, *Indent*, *Unindent* are allowed on multiple issues only if all issues in the selection are at the same level in hierarchy and have the same parent issue.
- Bulk Change button lets you use JIRA bulk change wizard with selection of issues from the structure.

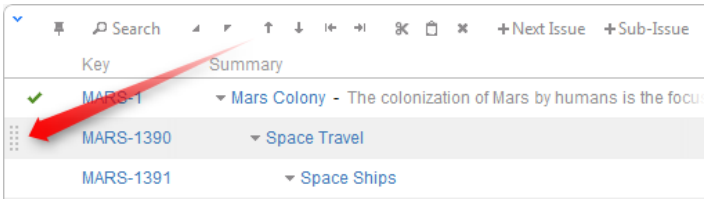
See Selecting Multiple Issues for details about working with multi-selection.

Using Drag and Drop

Drag-and-drop feature allows you to quickly move issues or selections of issues within the structure or add them from the secondary issue panels to the structure.

Basic Drag-And-Drop

To grab an issue, move your mouse pointer over the "handle" of the issue (the pointer will change to  when it's over the handle).



Then press and hold down the mouse button and start moving the issue.

> You can also start dragging by pressing and holding **Shift** on the keyboard and pressing mouse button anywhere on the row with issue (don't click on the links though).

As you move the issue over the grid, the structure will rearrange itself to show the possible positions for the dragged issues. Once the issue is in the correct place, release the mouse button and the issue will be moved.

Drag direction	Effect
Up / Down	Moves issues up and down the hierarchy without changing the indentation level, if possible.
Left / Right	Changes indentation level of the moved issues, if possible – without moving them up or down.

> It may be hard to unindent an issue by using drag-and-drop if you grab it by the drag handle, since the drag handle is usually close to the edge of the screen. Hold **Shift** and drag by some other place on the issue row in that case.

> After dragging has started you can release **Shift** keyboard button.

Dragging Multiple Issues

To move more than one issue, first select multiple issues (start with hitting **Space** button) and then move them using the "handle" of one of the selected issues. Holding **Shift** and dragging by any other place on the issue row also works.

> If you have multiple issues selected, but start dragging an issue that's not included in the multiple selection, only that issue is dragged.

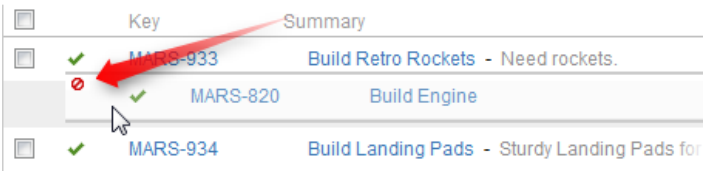
Cancelling Drag

If you need to cancel drag-and-drop without dropping issues at some random position, hit **Escape** keyboard button.

Drag-and-drop can also be undone.

Impossible Moves

If it's not possible to move the dragged issues onto the current position (for example, due to structure permission settings), an icon will be displayed at the top left corner of the dragged row.



Scrolling Structure While Dragging

If you have a large structure, you may need to have Structure grid scrolled up or down while you're dragging issues. Just move the issues over the top or bottom edge of the structure widget and the issues will be scrolled up or down. The further you move the dragged issues, the faster the scrolling is.

Using Cut & Paste may be more effective than Drag-and-Drop if you need to move several issues to distant positions.

Using Copy and Paste

Copy/cut and paste is a handy way to move issues around.

When you copy or cut issues (with toolbar button or Ctrl+C / Ctrl+X (Command+C / Command+X), selected issues are put into Issue Clipboard. When you paste with Ctrl+V (Command+V), issues are added **after** the currently focused issue. You can use Ctrl+Shift+V (Command+Shift+V) to paste issues from clipboard **under** currently focused issue.

While drag and drop is nice and visual way to rearrange issues, it might get tedious if you have to drag issues across long distances. Copy and paste solves that problem perfectly – copy issues to clipboard and scroll through the structure looking for a place to paste issues to.

See Selecting Multiple Issues for details about multi-selection.

Copy / Paste Scenarios

There are two main scenarios for using the Issue Clipboard:

- Copying Issues Between Structures — The contents of the clipboard is preserved in the current browser window, which allows you to copy issues from one structure and paste them into another.
- Moving Issues Within A Structure — Instead of using drag-and-drop function to move the issues within a structure, you can use the cut/paste feature. This is especially convenient, if you have a large

structure and, for example, need to move some issues from the top of the structure to the bottom or the other way around.

Please note that if you really have some text selected on the page, **Copy/Cut/Paste** keyboard shortcuts would operate on that text – you'll get a copy of the text in the system clipboard, and Structure clipboard will not be affected.

Copying Issues Between Structures

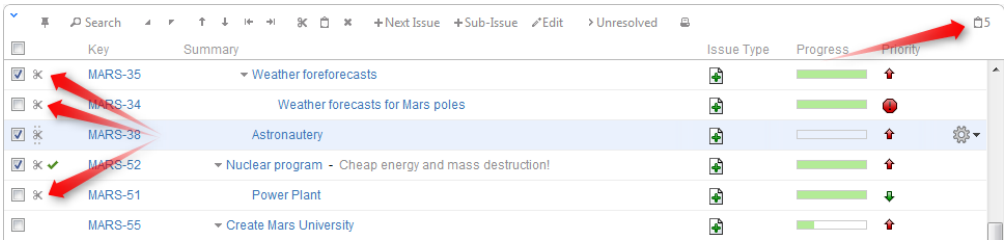
The contents of the clipboard is preserved in the **current browser window**, which allows you to copy issues from one structure and paste them into another.

To copy issues (with their sub-issues) from one structure to another do the following:

Cut

First add the desired issues to the clipboard:

1. Open the structure to copy from.
2. Select the issues you want to cut. Either select a single issue, or use multiple select.
3. Click the **Cut** button on the structure toolbar (or press **Ctrl+x** or **Command+x**).
4. Selected issues will be added to the clipboard and marked with a small scissors icon .
5. The **Clipboard** icon with the number of the cut issues 3 will appear in the top right corner of the structure widget.



Note, that the cut issues are not removed from the structure.

If the cut issue contains sub-issues, these sub-issues are cut with their parent.

Paste

After you have cut the issues, you can now paste them to any other structure:

1. **In the same browser window**, switch to a desired structure (you can use Structure Board or any other JIRA page with Structure).
2. If want to see the **Issue Clipboard** panel, you can open it clicking the **Clipboard** icon 3 . However, this is not necessary to use the Cut/Paste function.
3. In the structure grid select the issue after which the issues from the clipboard should be placed.

4. Either click **Paste** button on the toolbar (or press **Ctrl+v** or **Command+v**) to place the issues **after** the selected issue at the same indentation level, or press **Ctrl+Shift+v** (or **Command+Shift+v** on Mac) to place the issues **under** the selected issue (as the children).

When you paste issue hierarchy from a different structure, it's possible that the target structure already contains some of the issues. In this case, iterative **Merge** is performed, where issue is either moved to a new position if it is present in the structure or added if it is not present.

If you need to copy the same set of issues to several different structures, you can use drag-and-drop operation to move the issues from the secondary panel to the main grid instead of the Paste. In this case the issues will not be removed from the clipboard.

If the cut issue contains sub-issues, these sub-issues are pasted with their parent.

The Paste operation can be undone.

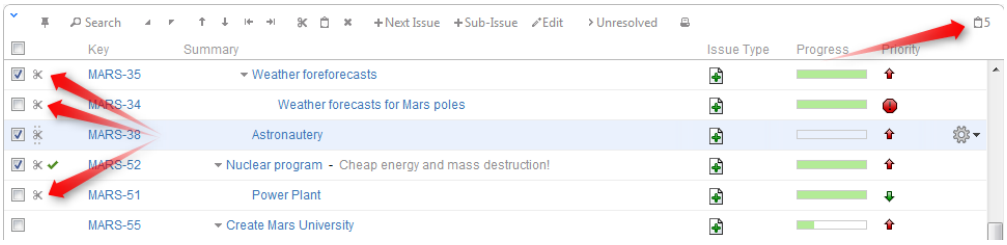
Moving Issues Within A Structure

Instead of using drag-and-drop function to move the issues within a structure, you can use the cut/paste feature. This is especially convenient, if you have a large structure and, for example, need to move some issues from the top of the structure to the bottom or the other way around.

Cut

First add the desired issues to the clipboard:

1. Select the issues you want to cut. Either select a single issue, or use multiple selection.
2. Click the **Cut** button on the structure toolbar (or press **Ctrl+x** or **Command+x**).
3. Selected issues will be put into the clipboard and marked with a small scissors icon .
4. The **Clipboard** icon with the number of the cut issues 3 will appear in the top right corner of the structure widget.



Note, that the cut issues are not removed from the structure.

If the cut issue contains sub-issues, these sub-issues are cut with their parent.

Paste

After you have cut the issues, you can now paste them back to any place in the structure:

1. If want to see the **Issue Clipboard** panel, you can open it clicking the **Clipboard** icon 3 . However, this is not necessary to use the Cut/Paste function.
2. In the structure select the issue after which the issues from the clipboard should be placed.
3. Either click **Paste** button on the toolbar (or press **Ctrl+v** or **Command+v**) to place the issues **after** the selected issue at the same indentation level, or press **Ctrl+Shift+v** (or **Command+Shift+v** on Mac) to place the issues **under** the selected issue (as the children).

If the cut issue contains sub-issues, these sub-issues are pasted with their parent.

After Paste the clipboard is cleared.

The Paste operation can be undone.

Undoing Changes

Structure lets you undo a potentially destructive operation if you realize that you have made a mistake or that the result is not what you expected. These operations can be undone:

- Adding issues from a search result;
- Removing issues from a structure;
- Drag-and-Drop;
- The Paste operation of a Cut & Paste sequence.

When you perform an operation that can be undone, a corresponding hyperlink appears in the footer at the bottom of the Structure widget. For example, if you drag and drop some issues, the link will read "Undo Drag and Drop". If you click the link, your changes are reverted, and the link itself changes to a "redo" link, allowing you to reapply the operation.

Only the last operation can be undone. There is no undo history.

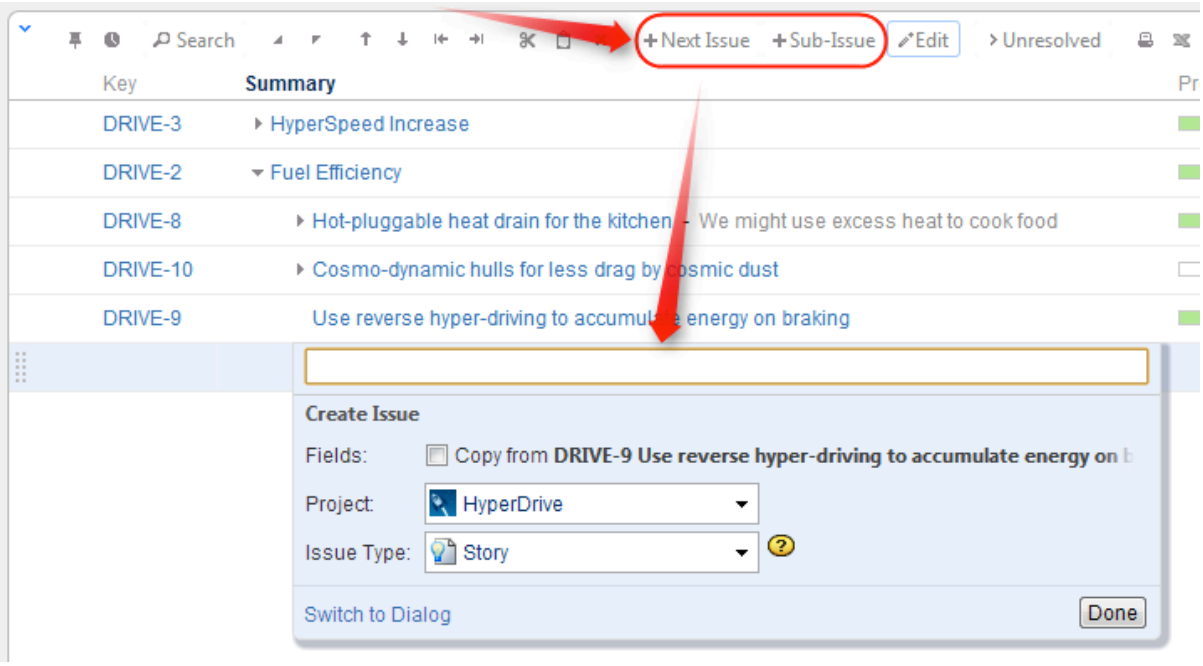
If the operation being undone has been uploaded to the server already, then a new operation (or several operations) will be uploaded in order to revert the changes. You will see both the original operation and the undo operation in the structure history.

2.4.8 Working with Issues

Structure lets you work with issues right in the structure widget.

Creating New Issues

Structure plugin lets you create new issues right in the structure widget or use standard "Create Issue" dialog and add newly created issue to structure automatically.



There are two modes of creating new issues, "creating from scratch" and "cloning". The mode used depends on whether **Categories** checkbox is selected on the summary editor shown above. When "creating from scratch", you can select the Project and Issue Type for the new project.

The selected options (creation mode, selected project and issue type for the "creating from scratch" mode) are remembered in the browser and are automatically suggested next time you create an issue.

Creating New Issues from Scratch

This is basically the same as using JIRA's **Create Issue** dialog.

1. Click **+New Issue** or **+Sub-Issue** on the toolbar (you can also hit **Enter** or **Shift+Enter** on the keyboard).
2. Make sure **Categories: Copy from...** checkbox is cleared.
3. Select **Project** and **Issue Type**.
4. Enter summary and, optionally, other fields before submitting the new issue.

   > To be able to edit other fields, you need to add the corresponding columns to your current view, before you start creating a new issue. Then, when editing a field, you can navigate to other fields by using **Tab**, **Shift+Tab**, or **Ctrl+Alt+arrows** keyboard shortcuts, or simply by clicking on a cell you wish to edit.

5. Hit **Enter** or click **Done** to finish editing and create a new issue on the server.
   a. Hit **Escape** to cancel creating a new issue.

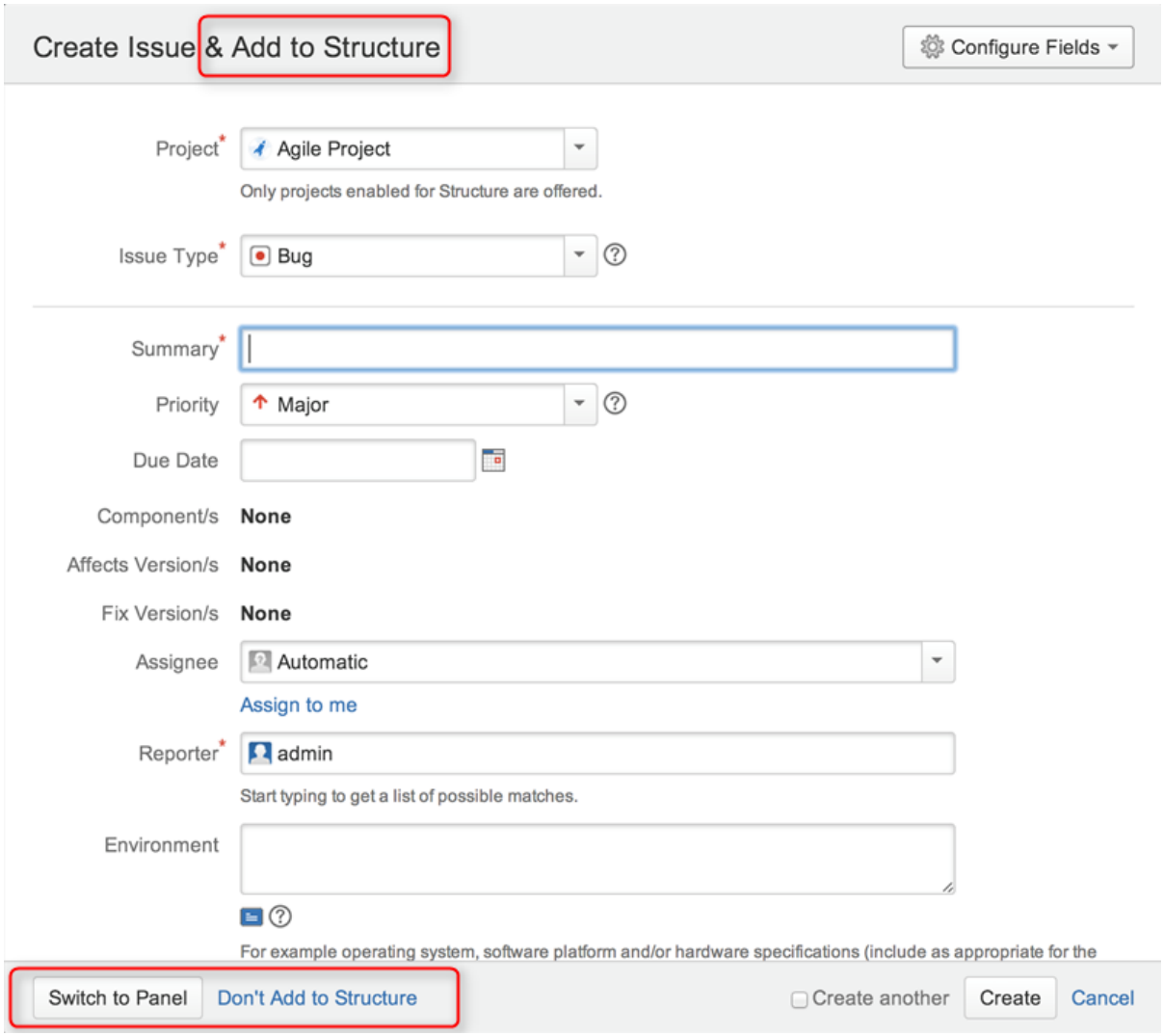
All fields that are not edited before you hit **Done** will assume their default values.

> If some fields are required and do not have default values, and you send a new issue to the server without those required fields, the operation will fail – but you can fix it, just add the required fields as columns, edit the field values and hit **Done** again. The other way to achieve this is to use **Switch to Dialog** functionality and use standard dialog to populate issue.

> You cannot create a JIRA sub-task from scratch in the structure widget. (This is about special issue type, **not** about creating sub-issues - you can create a new issue anywhere in the structure.)

Using "Create Issue" Dialog

You can click **Switch To Dialog** in any editor panel to display JIRA's **Create Issue** dialog, which lets to fill in fields not editable or not currently displayed in structure widget.

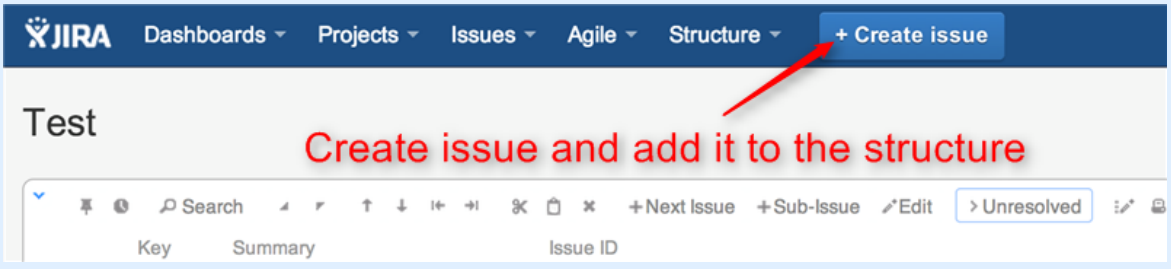


Once you're done, you can click **Create** and the created issue will be automatically added to the structure.

You can switch back to structure widget editor by clicking **Switch to Panel**, this will preserve all entered data and populate existing columns if possible. For other fields data will be preserved and sent back to server, once you click **Done**. You can also switch back to dialog mode at any time. The system will remember last used mode (dialog or panel) and use it next time you hit **+New Issue** or **+Sub-Issue**.

You can click **Don't Add to Structure** if you don't wish to add issue to the structure.

The standard **Create issue** button action has changed on structure widget page. Now it will open structure-enhanced dialog, allowing to add issue to structure immediately. This behaviour is indicated by the **+** sign.



On all other pages basic **Create issue** behaviour remains.

Cloning Existing Issue

The new issue becomes a "clone" of the selected issue - it has the same Project, Type and most other attributes, but you need to enter the summary for the new issue in the Summary field.

1. Select an issue in the structure.
2. Click **+New Issue** or **+Sub-Issue** on the toolbar (you can also hit **Enter** or **Shift+Enter** on the keyboard).
3. Make sure **Categories: Copy from...** checkbox is selected.
4. Enter summary and, optionally, other fields before submitting the new issue.
5. Hit **Enter** or click **Done** to finish editing and create a new issue on the server.
    a. Hit **Escape** to cancel creating a new issue.

When you create an issue this way, it copies the following fields from the issue that was previously selected:

- Project and Issue Type
- Parent Issue if the cloned issue is a JIRA sub-task
- Component, Affects Versions, Fix Versions, Environment, Assignee, Priority, Security Level
- All custom fields that are **required** by the fields configuration for that particular Project and Issue Type

Please note that the archived versions are skipped when copying Affects Versions, Fix Versions and version-based custom fields.

Creating JIRA Agile (GreenHopper) Epics

GreenHopper 6.1 or later introduces the Epic Name custom field, which is required by default for any epic. To simplify the process of creating multiple epics in the widget, Structure will copy the new epic's summary to its Epic Name field, if the latter is empty. This way you can simply type an epic name into the Summary field, and proceed to the next issue. The copying only happens once, when an epic is created. You can change the summary or the epic name at a later time if you want them to be distinct. Of course, you can also add the Epic Name column to the table and enter new epic names explicitly.

Additional Keyboard Shortcuts

Immediately after you have hit **Enter** or **Shift+Enter** or **Insert** to start editing a new issue, you can also use keyboard to change the creation mode.

Use the following keyboard shortcuts **while the summary field is still empty**:

Enter or **Tab**	Cycle through Project, Issue Type and Summary field. When Project or Type field is selected, use arrows or start typing to select a project or type.
Ctrl+Enter	Toggles cloning mode (**Categories: Copy** checkbox).
Alt+Enter	Switches editor to dialog mode and back to panel.

If you already have entered the summary, you can use mouse to change creation mode, project or issue type.

Uploading New Issue to the Server

After you've provided the summary and pressed **Enter**, the structure widget displays only the Summary field for a short moment as it takes some time to actually create an issue in JIRA. After the widget receives the confirmation from the server that the issue has been created, other columns for that issue are loaded.

While the new issue is being uploaded to the server, you can start creating the next issue.

Using Edit Mode

Note that when you are creating a new issue, Structure widget is in the Edit Mode – you can also enter values for other fields besides Summary by clicking on the field to be modified, or using keyboard shortcuts. When you have hit **Enter** or clicked **Done**, the new issue will be created with those values you have entered.

Editing Issues

In the Edit Mode, Structure widget lets you change fields of an issue right on the issue grid. This lets you quickly update issues without leaving the web page.

Editing works on every page where Structure widget is displayed. However, there are some limitations when editing issues from the Structure Gadget.

- Entering Edit Mode
- Changing Fields
- Using Keyboard in Edit Mode
- Correcting Input Errors
- Editing from Gadget
- On E-mail Notifications

You need Edit Issue permission on the issue to edit its fields. If you don't have the permission, a read-only flag is shown.

Entering Edit Mode

You enter Edit Mode by either editing an issue or creating a new issue. In general, to edit a value displayed in the Structure widget, do one of the following:

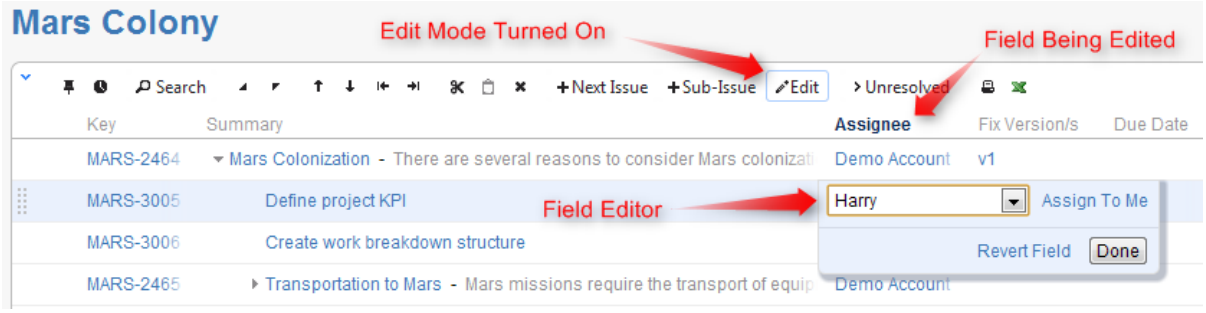
- **double-click** that value;
- select the issue and click **Edit** button on the toolbar;
- select the issue and use a keyboard shortcut – either **Tab**, or **F2**, or **s,s** ("s" twice);

If the value is a link (like in the Summary or Assignee fields), you can still double-click it: the browser will not open the link but will start editing instead.

If you are already in the Edit Mode, you can simply **click** the value you need to edit, or navigate there with special keyboard shortcuts (see Using Keyboard in Edit Mode).

In the Edit Mode:

- a field editor is shown in the currently edited cell;
- the edited column is highlighted in the table header;
- **Edit** button on the toolbar is toggled on.



Changing Fields

When editing a field, make the change with the field editor and click **Done** (or hit **Enter**) to have the change saved on the server. If you'd like to change several fields at once, click the other field you need to change or use **Tab**, **Shift+Tab**, or **Ctrl+Alt+arrow** to navigate and edit other fields. The changes will be saved on the server as soon as you have finished editing, or switched to editing another issue.

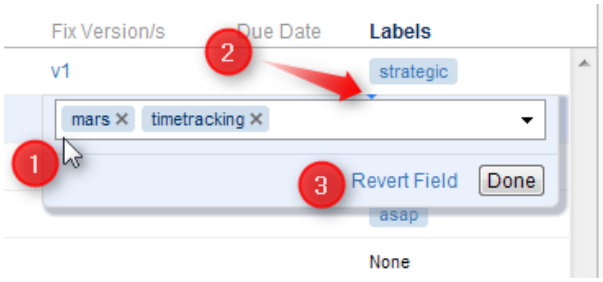
If your JIRA is configured to send e-mail notifications about changes, then a notification will be sent as soon as you have finished editing an issue - see On E-mail Notifications.

You can hit **Escape** to cancel changes that you have done to the edited field and exit Edit Mode. Click **Revert Field** link to restore the original value of the field and stay in the Edit Mode for further editing.

Hitting **Escape** only reverts the value of the currently edited field. Changes to other fields remain. So if you edit fields Summary, Assignee and Components, and hit Escape while editing Components, the changes to Summary and Assignee will still be uploaded!

The Field Editor

The editor for each field is the same as the one on the Edit Screen, although it is made more compact.



On this screenshot:

1. All help texts, descriptions and field labels are not shown. Hover mouse pointer over the input field to see help and field description.
2. Normally, the editor is aligned with the top left corner of the edited cell. However, if it does not fit horizontally on the page, its position is adjusted and a small blue triangle is shown to mark the place where the edited cell starts. (You can also look at the table header to see which field is being edited.)
3. The controls below the line are the same for all editors.

Allowed Changes

In the Edit Mode, you can change fields that are added to the Edit Screen for the edited issue. If a field is not on the Edit Screen, or if it can't be edited directly (such as the Status or Resolution fields), the editor won't be shown or it will display a corresponding error.

Additionally, each field may have particular limitations – such as Original Estimate being not editable after work has been logged (in JIRA's legacy time tracking mode).

Using Keyboard in Edit Mode

You can use keyboard shortcuts to quickly edit issues in the Structure widget.

Entering Edit Mode

Keyboard Shortcut	Action
Tab *or* **ss** *or* **F2**	Edit issue. The editing starts for the Summary field of the currently selected issue, or for the field that was edited previously.
Enter **Insert** or **Shift+Enter**	Enters Edit Mode for a new issue or sub-issue.

Keyboard Shortcuts in the Edit Mode

Keyboard Shortcut	Action
Enter **Ctrl+Enter** *(in large text fields)*	Exit Edit Mode and save all values on the server.

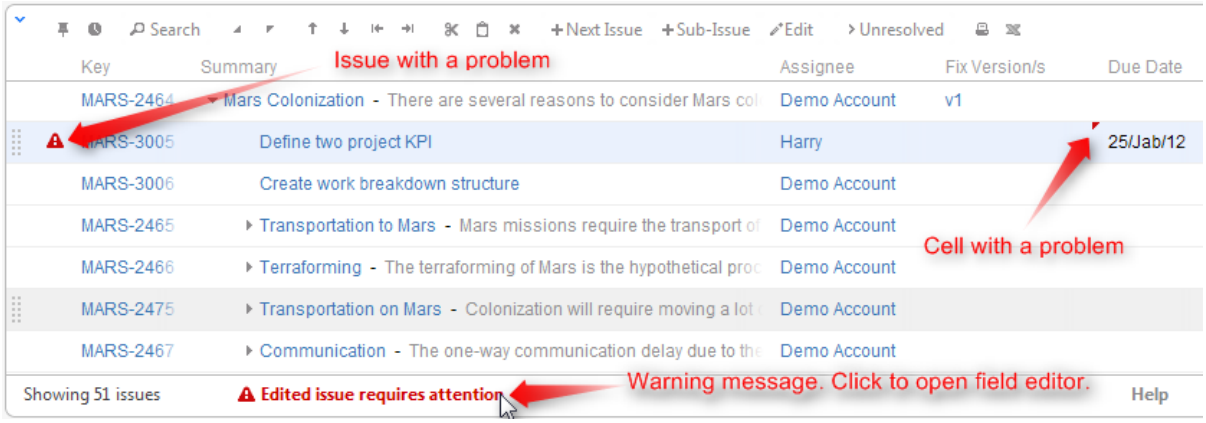
Keyboard Shortcut	Action
Escape *(hit twice in combo boxes and drop-downs)*	Revert the field to the value that was there before editing has started and exit Edit Mode. Note that if there are pending changes in other fields, they will be saved on the server.
Tab	Edit next editable field. If the currently edited field is the last editable field for the selected issue, start editing next issue.
Shift+Tab	Edit previous editable field. If the currently edited field is the first editable field for the selected issue, start editing previous issue.
Ctrl+Alt+	Edit the same field of the next editable issue.
Ctrl+Alt+	Edit the same field of the previous editable issue.
Ctrl+Alt+	Edit next editable field. Unlike **Tab**, this combination will not move editing to the next issue.
Ctrl+Alt+	Edit previous editable field. Unlike **Shift+Tab**, this combination will not move editing to the previous issue.
or **Alt+** *(in drop-downs)*	Opens drop-down list or selects the next value in the list. If the drop-down is shown, use **Enter** to select a value or **Escape** to cancel selection.
Alt+ *(in date/time fields)*	Opens date picker. Use arrows to navigate dates in the date picker and use **Enter** to select a date or **Escape** to close date picker.
and	Move between multiple fields on the same editor (for example, between the two editors of a Cascade custom field). Does not work if the input is a text field.
and *(for checkboxes and radio buttons)*	Move between multiple fields on the same editor (for example, between the checkboxes of a Multiple Checkboxes custom field).
Space	Select / unselect a checkbox or a radio button.
, , **Shift+**, **Shift+**	Select / unselect values in a Multi-Select custom field.

Note that **Tab** key moves editing to the next cell, so if you have multiple input fields on a single field editor, you need to use arrow keys to switch between them.

See Also: Keyboard Shortcuts

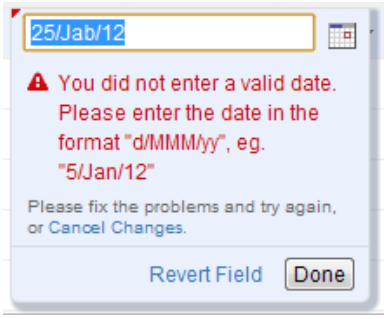
Correcting Input Errors

If you enter an incorrect value when editing a field, or if there are any other problems saving that value on the server, Structure widget will display a warning message and mark the cells with the problems.



Click on the warning message or on the cell with the error to enter Edit Mode, see problem details and correct the error. You can:

- correct the value and hit **Enter** or click **Done** to try to save the values on the server again, or
- click **Revert Field** to restore a previous value of the field, known to be valid, or
- click **Cancel Changes** to cancel all changes to this issue, including possible changes to other fields.



You can edit other issues and otherwise work with the Structure widget before fixing the editing problem. However, it is advised to correct the error as soon as possible.

Input Errors when Creating a New Issue

If the error happens when saving a new issue on the server, saving any further changes on the server is suspended – until the error is fixed or the creation of the new issue is cancelled. This is a necessary measure as the success of the following changes may depend on the success of the creation of that new issue.

When you have errors in the fields of a new issue, fix them as soon as possible or cancel the creation of that issue. Otherwise, any further changes are not uploaded until the problem is fixed and you risk losing them!

You can cancel creation of a new issue if you select it and click **Delete** button or hit **Delete** key.

Editing from Gadget

Structure Dashboard Gadget is a bit limited where it comes to editing issue fields, due to some incompatibilities between field editors and gadget framework. Because of that, only a handful of fields could be edited from within Structure Gadget (of course, if the user has permissions to edit them and only if those fields are added to the Edit Screen).

The following fields are editable from Structure Gadget:

- Summary
- Assignee
- Issue Type
- Priority
- Reporter
- Security Level
- Original Estimate
- Remaining Estimate

All other fields, including custom fields, cannot be edited in the Gadget. To edit those fields, open Structure Board or issue page or any other page with the structure.

On E-mail Notifications

Usually, when an issue is edited, an e-mail notification is sent to everyone involved with that issue.

When editing an issue with the Structure widget, the changes are saved on the server and the e-mail notification is sent when you:

- hit **Done** button;
- or start editing another issue.

So if you switch from editing one field of an issue to editing another field of the same issue immediately, no update will have happened and no mail will have been sent yet.

If you need to change several fields of an issue and avoid multiple e-mails being sent, edit one field then navigate to the next field. Only hit **Done** or **Enter** when you have finished editing all fields.

To switch from editing one field to editing another field, you can:

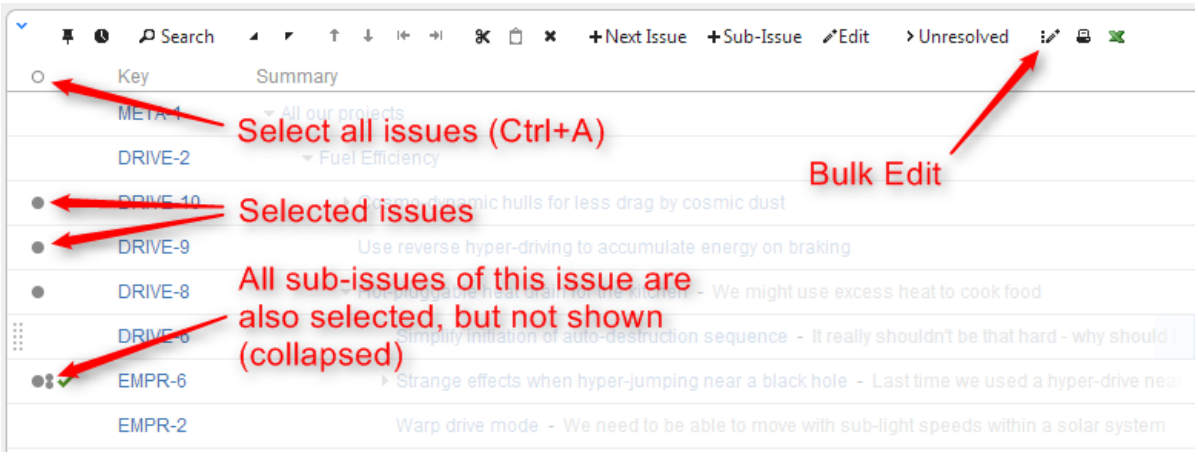
- click on another field that you need to edit;
- use Tab, Shift-Tab, Ctrl+Alt+arrows keyboard shortcuts to move to the next/previous fields.

So if you edit a field, click Done, then edit another field - that's two edits and there will be two notifications. If you edit a field, then edit another field, and only then click Done - that's one edit and one notification.

Bulk Change

With Structure, you can quickly select multiple issues and open JIRA's standard bulk change wizard for those issues.

1. Select issues by clicking on issue selectors, or pressing *Space*, *Shift+Space*, or other Keyboard Shortcuts for selecting issues.
2. Click **Bulk Change** action on the toolbar.



3. Standard JIRA bulk operation wizard opens. Select the action you'd like to take and proceed.
4. At the end, the browser will be redirected back to Structure Board.

Cloning Multiple Issues

Structure allows you to copy the whole structure and clone all issues in the structure. See Copying Structure and Cloning Issues.

If you need to clone only some of the issue in the structure, you can use the following procedure:

1. Select issues you'd like to clone using multiple selection.
2. Use **Copy** action on the toolbar (or hit Ctrl+C / Command+C) to copy the issues to the Issue Clipboard.
3. Use **Structure | Create Structure** menu and create a new temporary structure, let's call it **T1**.
4. Open the new structure and use **Paste** action to add issues from clipboard.
5. Copy and clone structure **T1** – see Copying Structure and Cloning Issues. Let's name the resulting copy **T2**.
6. Open **T2**, select all issues (use Ctrl+A / Command+A).
7. Use issue clipboard in the same way to copy cloned issues back to the structure where they are needed.
8. Delete structures **T1** and **T2**.

Using JIRA Actions

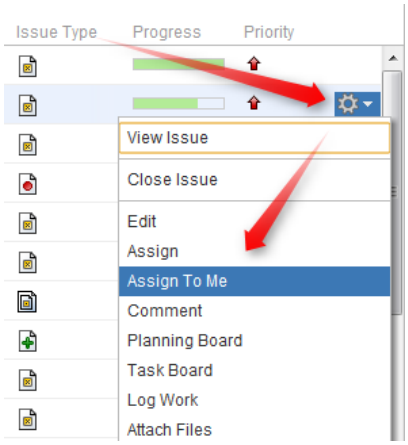
Structure widget lets you use JIRA actions available for the issues from the JIRA's Action & Operations Drop-Down and JIRA keyboard shortcuts for the most frequent actions.

Using Actions Drop-Down

Structure widget has drop-down menu with actions and operations available for the selected issue - just as the JIRA's Issue Navigator.

To use an action:

1. Click on the **Gear Button** at the right side of the widget in the issue's row, or select the issue with the keyboard and hit **Alt+Down**.
2. Select the action with the mouse or use **Up/Down Arrow** keys and then **Enter** to select the action with the keyboard.



Using JIRA Shortcuts

Most JIRA shortcuts that are available on the Issue Navigator page also work in the structure widget. Just select an issue and hit the shortcut.

The most useful shortcut is "." (dot) - available since JIRA 4.2 - which lets you type in the name of the action you need performed.

Calling an action usually brings up a dialog or moves the browser to another page. Please pay attention to the dialog title or the window title to see that you're applying the action to the correct issue.

On the Issue Page, keyboard shortcuts are always applied to the viewed issue - regardless of the selection in the structure!

No Page Reload

In many cases Structure is able to proceed without page reload after you have applied a JIRA action to an issue. The applied changes are immediately visible in the Structure widget, and that gives you a very smooth experience of working with a collection of issues.

Whether a page is reloaded after an action is applied depends on which page are you using to work with issues, and what action is being applied. On the Structure Board, most actions do not require page reload.

2.4.9 Secondary Issue Panels

Secondary Issue Panels are the auxiliary grids, which can open next to the structure main grid. Even though they serve slightly different purposes, they work very similarly. They display a set of issues, which you can add to the structure grid using the drag-and-drop or cut & paste.

Structure has the following secondary panels:

- Current Issue — The Current Issue secondary pane is shown in the structure widget on the Issue Page, if the displayed issue is not present in the currently selected structure.
- Issue Clipboard — Issue Clipboard secondary panel allows moving issues within the structure and between different structures.
- JIRA Search Results — JIRA Search Results panel is displayed when the Search is on, there are issues in the search result that are not in the structure, and More Issue button is turned on.
- Removed Issues — Removed Issues panel contains the issues that you have just removed from the current structure - just in case you need them back.

Configuring Secondary Panel View

You can configure the columns displayed on the secondary panel in the same way it's done on the primary structure panel - see Customizing Columns. The secondary panel configuration is the same for all secondary panels and is stored like the main panel's configuration.

Resizing Secondary Panel

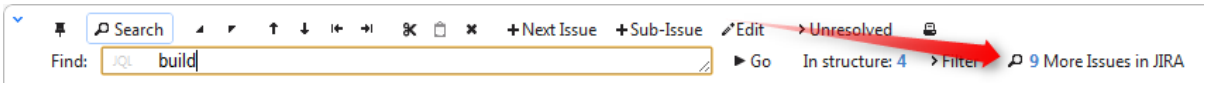
You can divide the horizontal space between a secondary panel and the main panel by dragging the separating border.

Secondary Panels Are Read-Only

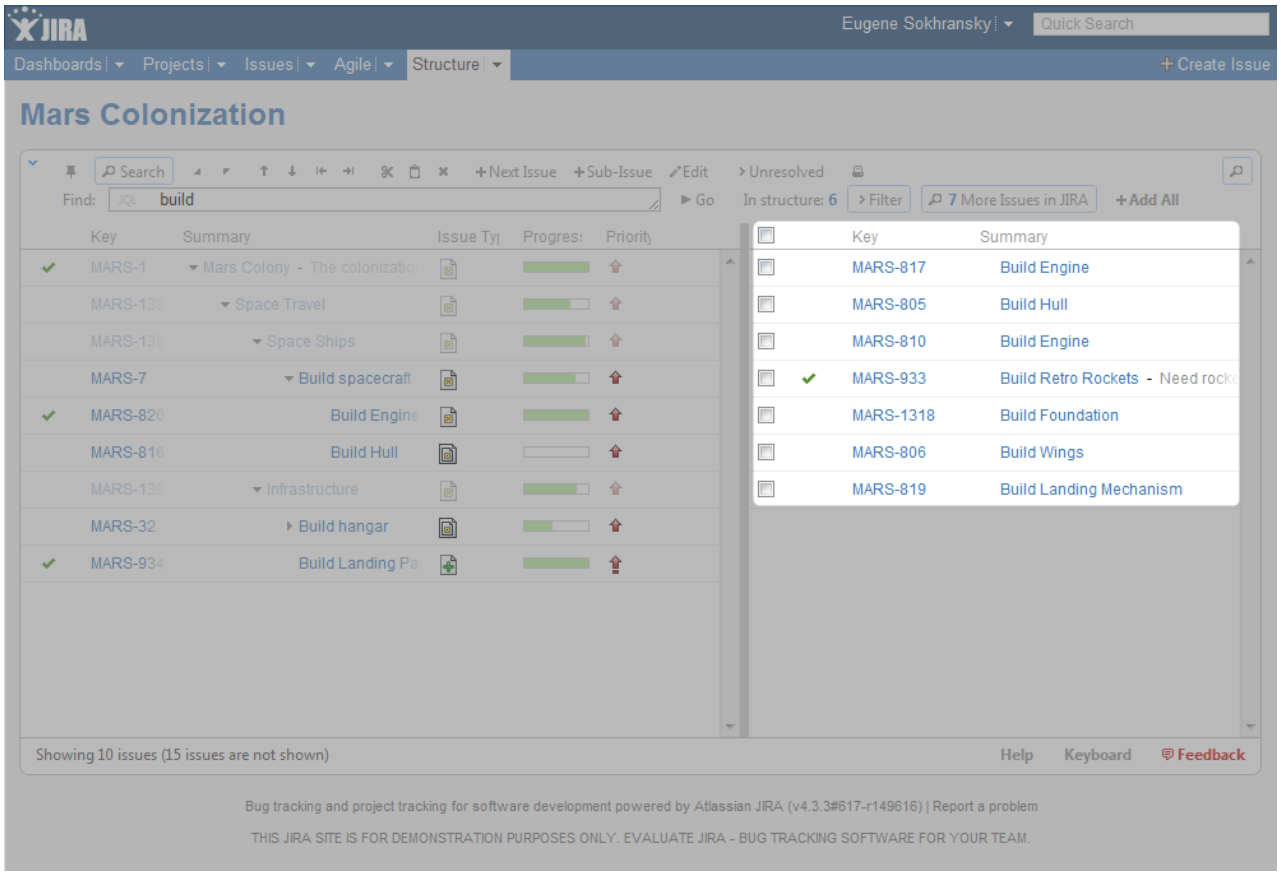
Although they seem to work in the same way the structure panel works, you cannot edit or create issues on the secondary panel, nor can you move issues around there.

JIRA Search Results

JIRA Search Results panel is displayed when the Search is on, there are issues in the search result that are not in the structure, and **More Issue** button is turned on.



As you start typing your search query, apart from the highlighted/filtered issues belonging to the current structure, you can also see those that match your search criteria, but are not included in the structure. They will be displayed in the JIRA Search Results panel, which opens to the right of the main grid.



Besides moving issues from the secondary panel with drag-and-drop or cut/paste, you can use **Add All** button on the search toolbar or keyboard shortcuts specific to the search field.

See Searching Outside Structure for the details.

The search is run through all projects for which the Structure plugin is enabled (see Selecting Structure-Enabled Projects).

Issue Clipboard

Issue Clipboard secondary panel allows moving issues within the structure and between different structures.

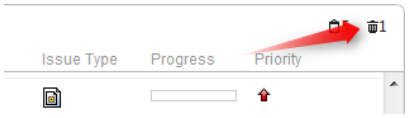
For details about using Issue Clipboard, see Using Copy and Paste.

Removed Issues

Removed Issues panel contains the issues that you have just removed from the current structure - just in case you need them back.

Once you remove an issue from a structure it is saved to the **Removed Issues** secondary panel. If you have removed an issue by mistake, you can open Removed Issues panel and reinsert it back into the structure.

The Trash icon in the top right corner of the structure widget indicates if there are any issues in the Removed Issues panel. To show or hide the panel, click the Trash icon.

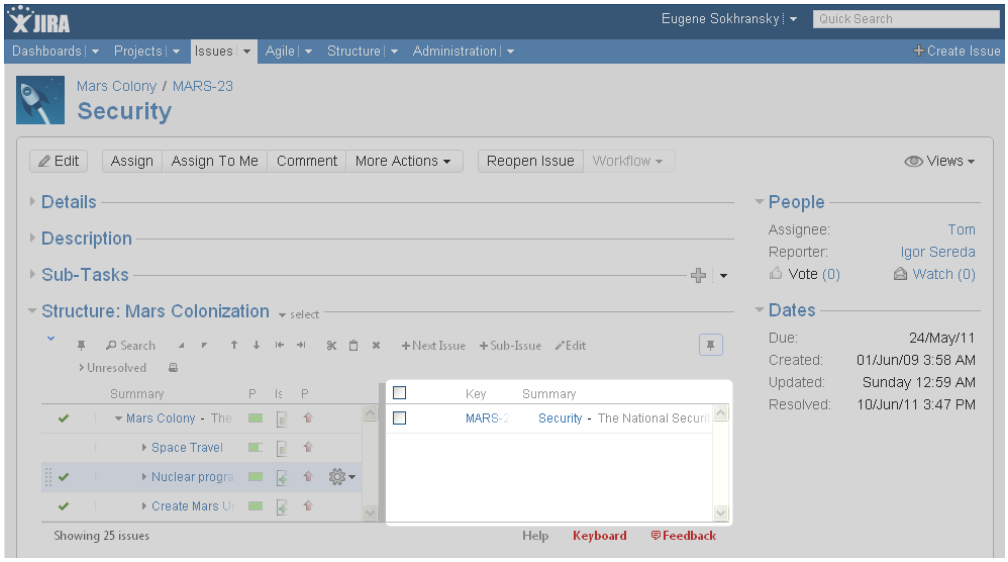


To add issues back to the structure, select the desired issues in the Removed Issues panel and move them to the desired location in the structure using drag-and-drop or cut and paste operations.

> Contrary to the Issue Clipboard, the contents of the Removed Issues panel does not survive page reload. So if you navigate to a different page, you will no longer be able to view the removed issues.

Current Issue

The **Current Issue** secondary pane is shown in the structure widget on the Issue Page, if the displayed issue is not present in the currently selected structure.



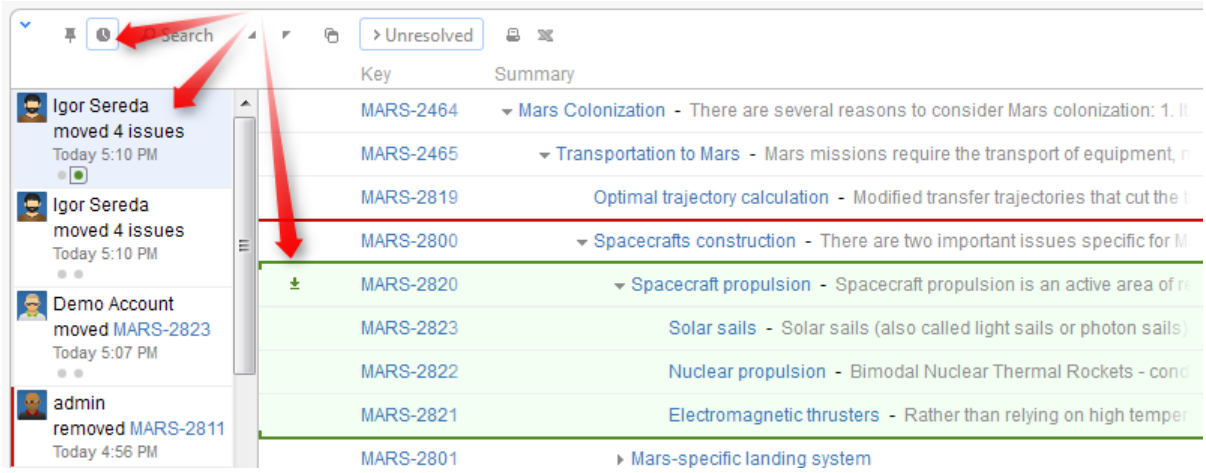
With this secondary pane you can immediately see that the issue is not in the structure and, if necessary, add it to the structure on the spot using drag-and-drop.

To hide/show current issue, click the Pin icon in the top right corner of the structure widget.

2.4.10 Viewing History of a Structure

Structure plugin records every change that you or other users make to a structure. The History View lets you see those changes and previous versions of your structures.

To turn on the History View, click the clock button on the toolbar. The list of recorded changes appears to the left of the structure grid, and the most recent change is selected.



Structure History has been introduced in Structure version 1.4. All changes made with earlier versions of Structure plugin have not been recorded.

Reading History View

By default, 20 most recent changes are loaded. If there are more, you can click the Show More button at the bottom of the list to load earlier changes. Newer changes are loaded and added to the top of the list as they happen.

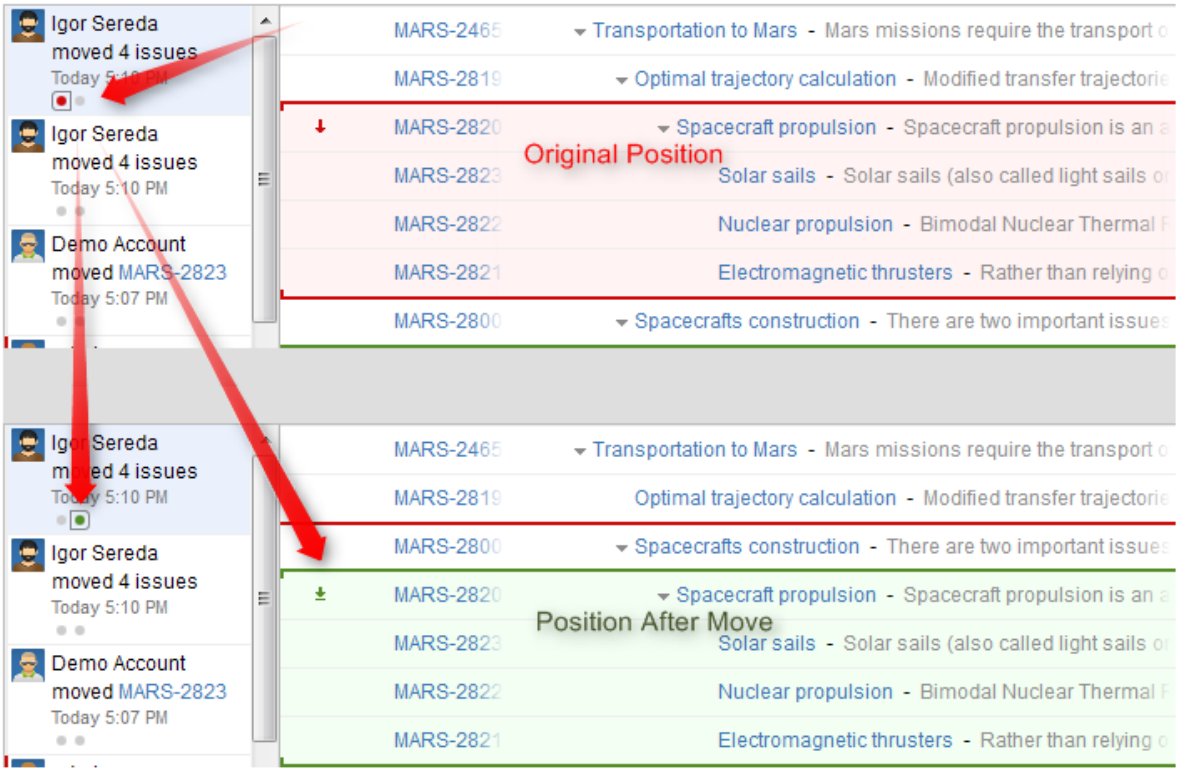
For each change, the following information is shown:

- The avatar and the name of the user who has made the change. In JIRA 4.4 and up you can hover your mouse over the user avatar to see the user details.
    - If the change has been made by a synchronizer, the synchronizer's name is shown. User avatar displays the user account that the synchronizer was running under.
- The nature of the change – how many issues were affected, were they added, removed or moved.
- The date and time when the change was made.

When you click a particular change, the main panel of the widget shows the structure as it was when that change was made. The affected issues are highlighted, and the structure expands and scrolls as needed to bring them into view.

Use the **Ctrl+]** and **Ctrl+[** keyboard shortcuts to navigate to an earlier or later change.

If issues were removed, they are shown in their position before the removal. Moved issues are shown in their new position by default, and their original position is marked by a red horizontal line. Use the small toggle button in the history section to show moved issues in their original position instead.



Limitations of the History View

History view is a powerful tool to track the history of a structure, however, it does not allow to change the history or to see the values of the issue fields as they were in the past. More specifically:

- All columns with issue fields (Summary, Description, Status and others) show current values – **not** the values that the issue had when the structure change was made.
- Aggregation columns, such as Progress or Total Time Spent do not show any values.
- You cannot edit issues, create new issues or change structure when viewing history.
- The history cannot be modified. (The administrator is able to clear whole Structure history.)

Printing a Previous Structure Version

You can Open Printable Page when viewing a previous version of the structure. The printable page will show the structure as it was after the selected change has been applied.

Note that all limitations apply: the current values of the fields will be displayed and Progress and other aggregate columns will not be displayed.

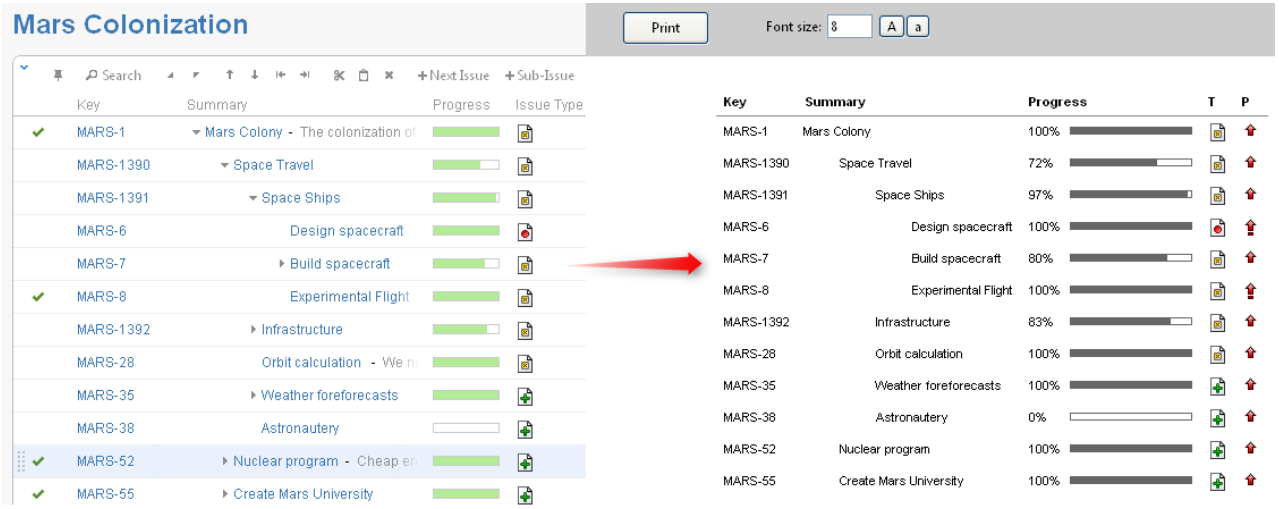
Exporting a Previous Structure Version to XLS Format

Like with the printable page, you can export structure to XLS (Excel). The XLS file will contain structure as it was after the selected change has been applied.

2.4.11 Printing Structure

Printable page lets you print the current structure from the browser.

Click the **Printer** button on the structure toolbar and the structure spreadsheet will open in a separate browser window or tab. The view fully copies the structure widget appearance - you can see the same issues as in the structure. For example, if some sub-issues are hidden, you will not see them on the printable page either.



The columns displayed on the printable page will be the same as in the structure widget, however, the widths of the columns will be set by the browser. To change the columns on the printable page, change them in the structure widget and click the Printer button again.

Summary column on the printable page displays only the summary field, without issue description. If you'd like to print description, add a separate Description column to the structure widget.

Depending on the number of columns, and the amount of the texts, it may be necessary to adjust font size before printing.

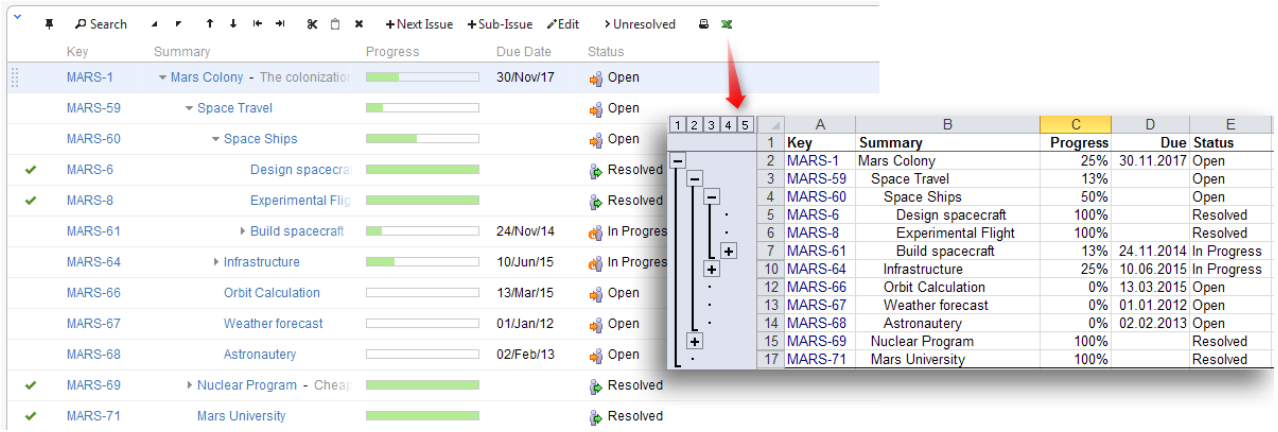
It's a good idea to print a single sample page to decide whether font size needs changing.

When ready to print, click Print button on the printable page or use your browser's Print menu.

2.4.12 Exporting Structure to XLS (Excel)

You can download the structure that you see on the screen as an XLS file and open it in Microsoft Excel or in other applications that support this format.

Click the **Excel** () button in the toolbar and the browser should download a new XLS file, which you can save or open. The XLS file will contain all issues that are displayed currently by the structure widget.



The XLS file has the same columns as the structure widget. Like with the printable page, the Summary column displays only the Summary field, without issue description, but indented to show how sub-issues are nested. If you'd like to export issue description, add a separate Description column to the structure widget.

Compatibility

The exported file is compatible with Microsoft Excel 2003, 2008 and 2010.

Mac OS X Users: On a Mac, **Excel** button is not present on the toolbar by default. To show the button, use **X, X** keyboard shortcut (press **X** twice).

Note that XLS format allows up to 65536 rows in the spreadsheet, so a larger structure wouldn't fit - use filtering or Fixed Structure View.

Likewise, if you have issues that are more than 15 levels deep in the hierarchy (meaning that they have 15 "parents" and "grand-parents"), they will be all shown with level 15 indentation in Excel, due to a technical limit.

Row Groups

The rows are grouped together using Excel grouping feature to form structure in the spreadsheet – you can expand and collapse sub-issues under a certain parent issue.

The maximum depth of grouping in XLS file is 8, so if you have a deeper structure, it still will be exported but the grouping will work only for the top 8 levels.

Not all spreadsheet applications that support XLS format also support row grouping feature. At the time of writing, Open Office does support it, but Google Docs don't.

Columns

The columns are formatted in the best way suitable for a spreadsheet.

Column Type	Notes
Issue Key	The cell with an issue key is a link to the actual issue.
Summary	Cells in the Summary column have indentation just like on the Structure Widget. Note that if you change the format of a cell there, you might lose the indentation level.
Progress	Progress field contains a fractional number from 0 to 1, formatted as a percent value.
Description, Environment and large text fields	The text might not fit in the column. You can increase column size or use **Format Cells | Alignment | Wrap Text** option in Excel to have a large text take more than one line, increasing the row height. Note that a cell might not accommodate a very large text and you might see only the first part of it.
Dates	Date values are displayed in your local date format.
Estimates, Time Worked	The duration fields contain actual numbers (fractional number of days), which you can sum or otherwise process. The display format is **HH:MM**, where HH is the number of hours and MM is the number of minutes. So an estimation of 5 days will be displayed as **40:00** (if you have 8-hour days).
Standard custom fields	Standard custom fields are rendered according to their type.
Plugin-provided custom fields	Custom fields from other plugins are displayed as they are rendered.

Note for Plugin Developers

If your plugin provides a new custom field type, please ensure that the field is displayed with the best compatibility with the other plugins, including Structure. In your column view velocity template, check for `$displayParams.textOnly` and/or `$displayParams.excel_view` and/or `$displayParams.nolink` – all those parameters will be set to `true` by Structure and may also be used by other plugins. See `CommonVelocityKeys.java` and JIRA sources for examples.

Printing

The XLS file is set up for a standard printing configuration:

- Page orientation is Landscape.
- The content is fit horizontally on the page (you might need to change that if you have too many columns or large content).
- Paper size is set to *Letter* if your account locale is US or Canada, otherwise it is set to *A4*.

Make sure you see Print Preview before sending the document for printing. If you don't like how it looks, consider using Printable page.

2.4.13 Real-time collaboration

Structure widget is a real-time collaboration tool.

The issue hierarchy displayed in the widget is kept up-to-date with the JIRA server, so if someone else changes the structure on the server, you will see the web page update within several seconds. Issues that have been added, removed, or moved are highlighted for a second with a flashing yellow background.

In the same fashion, the values of the issue are maintained up-to-date: if someone edits an issue or otherwise changes it, the structure widget will update the displayed fields within a few seconds. A value that has been changed is highlighted for a second with a flashing yellow background.

This feature lets you collaborate with other people who may work with the same structure on different computers.

Structure keeps data up-to-date by polling the server with short requests every few seconds when the application is ready. If structure widget detects that the browser is inactive it will reduce polling frequency to conserve network traffic.

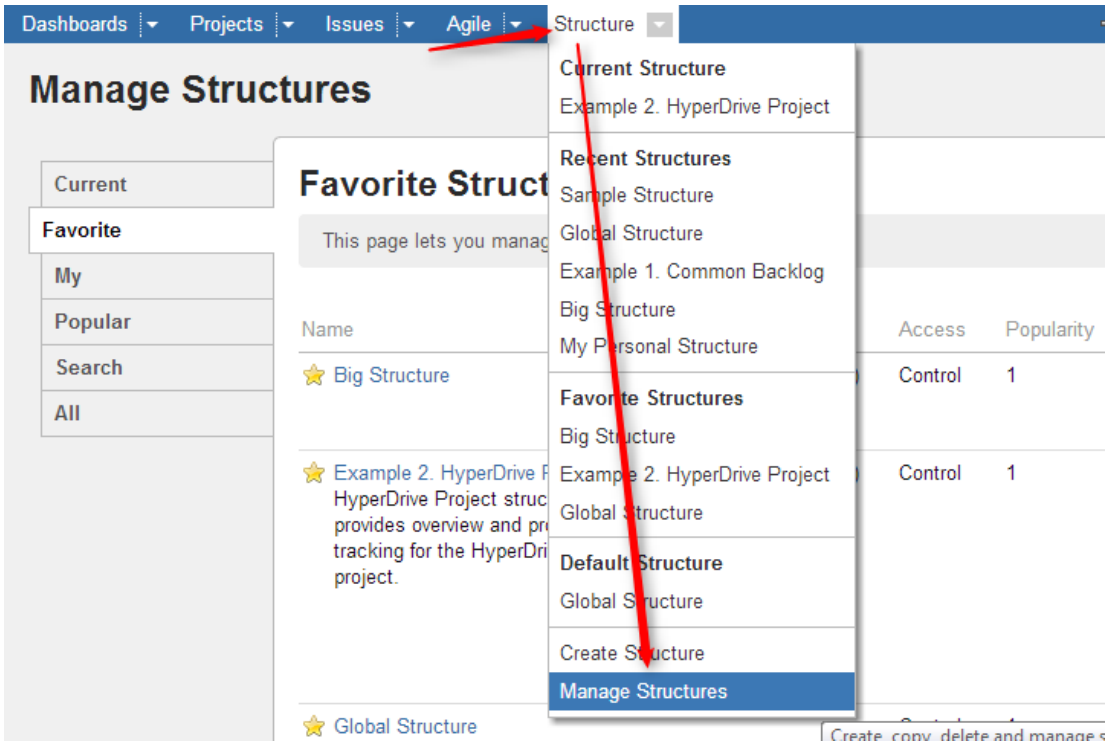
On all pages with the Structure widget, except for the Dashboard, when user inactivity is detected (there's no using of keyboard or mouse for at least 5 minutes), the polling of the server stops. It resumes as soon as the user moves the mouse or touches the keyboard (when the browser's window is active). This allows session cookies to properly expire after some inactivity period.

In some cases you would like to open a Structure Board and have it continuously display up-to-date information without any user activity – for example, to show a structure on a heads-up display. To have Structure always poll the server, press **xx** ("x" two times) to display additional actions on the Structure Toolbar and turn on Continuous Polling with the button

2.5 Managing Structures

Structure Plugin lets you have several independent structures in JIRA. Manage Structures page lets you view, search for, create, and delete structures as well as change their settings.

To open Manage Structures page, go to **Structure** menu in the top navigation bar, and then select **Manage Structures**.



Manage Structure page contains the following tabs:

- **Current** – shows the only the structure you've currently working with.
- **Favorite** – lists structures that you have marked as your favorite.
- **My** – lists structures created by you.
- **Popular** – lists structures that are marked as favorite by at least 2 users, ordered by their popularity.
- **Search** – allows you to find structure by name, owner or ID.
- **All** – this tab lists all structures visible to you.

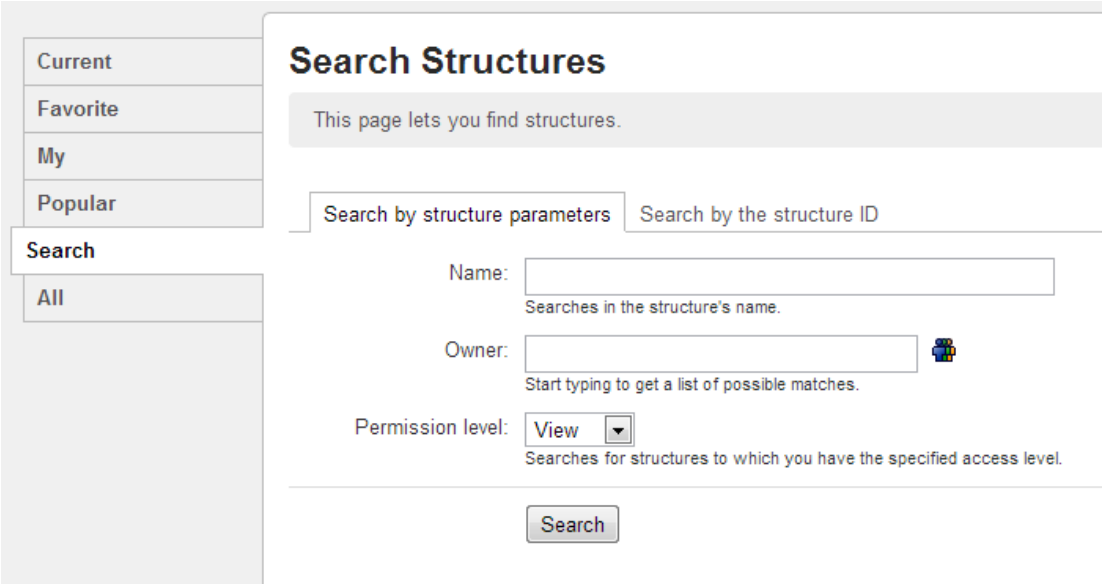
Since anonymous users can't create structures and can't mark structures as their favorites, **Favorite** and **My** tabs are not shown when you are not logged in.

More about managing structures:

- Locating a Structure
- Structure Details
- Creating New Structures
- Structure Permissions
- Customizing View Settings
- Copying a Structure
- Deleting a Structure

2.5.1 Locating a Structure

To find a specific structure, use **Structure | Manage Structures** menu and select **Search** tab.



Finding Structures by Name, Access Level or Owner

To search for structures by their properties:

1. Enter any of the search parameters. Parameters are:

Name	Only structures that contain the specified text in their name will be shown. You can use a part of the word that you know should be in the structure's name.
Owner	Only structures that are owned by the specified user will be shown.
Permission Level	Lets you select the structures that you can Edit or Control, according to the selected permissions level. (For example, if you select **Edit** permission level, you will see all structures that you can edit and control, but you will not see structures that you can only view.)

2. Click **Search**. If no parameters were specified, all structures visible to you will be shown.


You can search by structure owner only if you have the permission to browse users.

Finding a Structure by Its ID

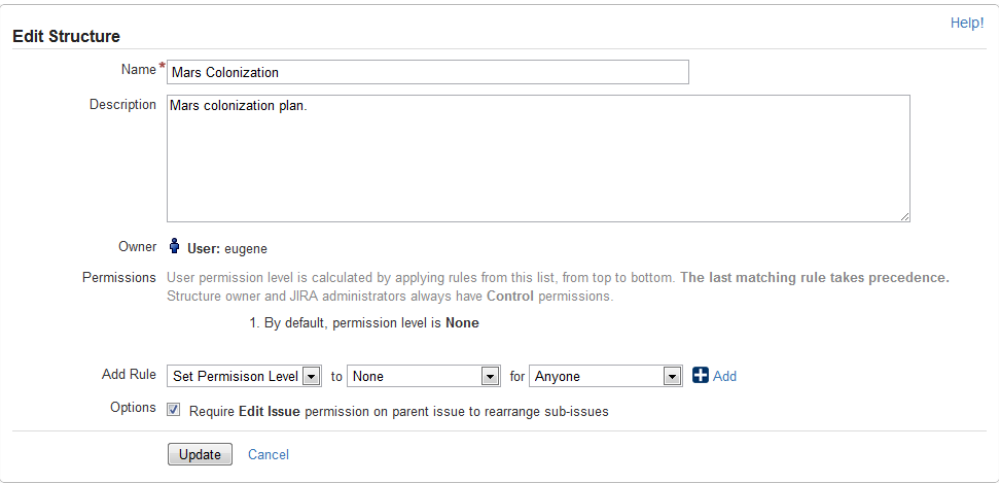
To perform a search by structure's numeric ID:

- Click **Search by the structure ID** tab.
- Enter the structure ID. (It must be a number.)
- Click **Search**. If there's a structure which has the specified ID and you have the permission to view it, it will be shown.

2.5.2 Structure Details

Every structure has the following parameters:

Name (*required*)	Name is used to identify the structure in the drop-downs like the *Structure* menu in the top navigation bar.
Description	Used to describe the meaning of the structure to the users.
Owner	The user who has created the structure. You can't change the owner.
Permissions	Define who can view, edit or configure the structure. See Structure Permissions for details.
Require Edit Issue Permission flag	When *Require Edit Issue permission on parent issue to rearrange sub-issues* flag is set, additional permission constraints are applied to figure out what changes is the user allowed to make. See Structure Permissions for details.



You can specify structure details when Creating New Structures and when Editing Structure Details.

Editing Structure Details

To edit details of a structure:

1. Open Manage Structure page by using **Structure | Manage Structures** menu.
2. Locate the structure you need to change and click on **Configure** link in the **Operations** column.

If you do not see **Configure** link, then you probably do not have Control permission on that structure.

2.5.3 Creating New Structures

To create a new structure, use **Structure | Create Structure** menu or click **Create Structure** button on the Manage Structures page.

You need to specify at least the name of the new structure, and optionally description, permissions and other parameters. See Structure Details for description of the structure parameters.

When you create a new structure, you become the owner of the structure. Structure owner always has full access to the structure - see Structure Permissions.

> Only logged in users who have access to the Structure Plugin are allowed to create new structures. See Who Has Access to the Structure.

2.5.4 Structure Permissions

Every structure has a list of permission rules, which defines who is allowed to see, edit or configure the structure.

Access Levels

Each user has one of the following access levels to a structure:

None	The user does not see the structure at all and does not know that it exists.
View	The user can view the structure but cannot make changes.
Edit	The user can view the structure and can rearrange issues in the structure, add issues to the structure and remove issues from the structure.
Control	The user can view, edit and configure the structure - including changing structure permission rules and configuring synchronizers.

Default Access

By default, all users have **None** access level.

The structure's owner and JIRA administrators always have **Control** access level.

Therefore, if you create a new structure and do not specify any permission rules, it will be a private structure that only you and JIRA administrators will be able to see and modify.

Permission Rules

Users who have **Control** permission on a structure can define permission rules by Editing Structure Details.

Permission rules list is an ordered list that's used to calculate the access level for a given user. Each rule has a **condition** that is matched against the user, and **access level** which is set if the condition matches. The conditions are applied from top to bottom, and the **last matching rule has precedence**.

The following conditions are supported by permission rules:

Anyone	Matches any user, including anonymous (not logged in). This condition can be used to set a default permission for everyone.
Group(G)	Matches users that belong to the group G.
Project Role(R,P)	Matches users that have role R in project P.

Additionally, there is a special rule type **Apply Permissions From**, which works by going through the permission rules from a different structure. You can apply permission rules only from structures with Control access level for you.

Examples

- Anyone can view, developers can edit, only the owner and admins can control:

  1. View for Anyone
  2. Edit for jira-developers (Group)

- Any logged in user can edit, except for the users from structure-noaccess group, who can't even view the structure. Project administrators are allowed to control the structure:

  1. Edit for jira-users (Group)
  2. None for structure-noaccess (Group)
  3. Control for Administrators of Mars Colony (Project Role)

- Incorrect configuration: everyone is given View access level

  1. Control for jira-developers (Group)
  2. Edit for jira-users (Group)
  3. View for Anyone

  Although the configuration looks ok at first glance, remember that **the last matching rule has precedence**. So regardless of whether the user is part of jira-developers or jira-users group, their access level will be set to View by the last rule.

Edit Issue JIRA Permission and Editing Structure

If you set *Require Edit Issue Permission on Parent Issue* flag on the Structure Details page, additional per-issue permissions checks will be performed to decide whether the user is allowed to change the structure.

If the flag is on, the user must have Edit Issue permission on a parent issue to adjust its sub-issues. In other words, direct sub-issues (or children issues) are treated as if they are part of the parent issue, and therefore adding sub-issues, removing sub-issues and rearranging sub-issues is actually changing the parent issue - for which the Edit Issue permission is required.

The user must also have **Edit** access level to the structure to be able to make changes at all.

Note the following:

- Top-level issues do not have a parent issue, and therefore are not affected by this flag: the user can add/rearrange issues at the top level of the structure if they have Edit access level.
- If issue A has sub-issue B, and B has sub-issue C, then to be able to move or remove C from the structure, the user needs Edit Issue permission on B - not on A. In other words, the Edit Issue permission is required only for the direct parent issue.

Permissions Caching

Structure plugin maintains a cache of users permissions with regards to each structure. In most cases, the cache is recalculated automatically, but in some cases Structure plugin may miss a change in a user's groups or roles. The result could be that the changed permissions take effect several minutes later (but only with regards to Structure Permissions). A user can force the cache to be recalculated by doing **hard refresh** from the browser. Typically, it's done by holding **Ctrl** or **Shift** or both and clicking the **Refresh** button.

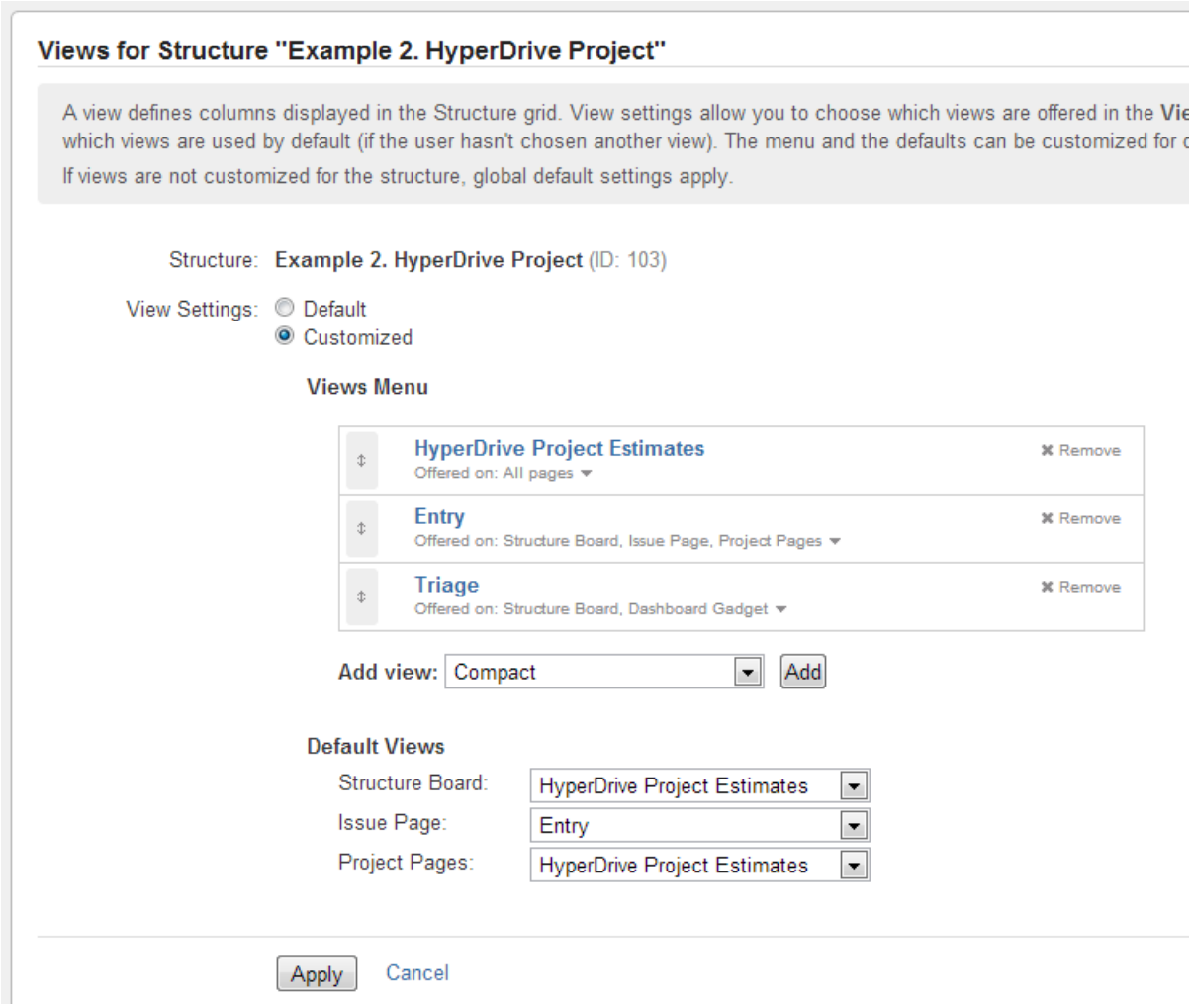
2.5.5 Customizing View Settings

A structure's view settings determine which views are offered to the users in the Views Menu when they are using that structure, and which view is the default. Initially, each structure has default view settings, defined globally for all structures.

A view is called **associated** with a structure if it is part of the Views Menu, as defined by the structure's view settings.

You can customize view settings if you have **Control** access level to the structure – open **Manage Structures** page and locate the structure, then click **Views** link.

You can change the default global view settings if you are a JIRA administrator – open **Administration | Structure | Defaults** tab and click **Change** in Default View Settings section.



Switching Between Default and Customized View Settings

To customize view settings for the structure, select **Customized** radio button. The default settings are copied and you can adjust them up to your needs.

To revert to default view settings, select **Default** radio button.

Configuring Views Menu

Views Menu section on the view settings page lets you configure Views Menu for each type of JIRA pages where Structure widget is present.

- To add a view to the menu, select the view in the **Add view** drop-down and click **Add**.
- To remove a view from the menu, click **Remove** button.
- To change a view's position in the menu, drag the view by the drag handle at the left of the view bar.
- To restrict a view's appearance in the menu to some specific pages, click **Offered on:** line and select the pages where you'd like this view to be used.

Configuring Default View

In the **Default Views** section, you can select which view from those added to the menu is the default for a given JIRA page (Structure Board, Issue Page and Project Page). Pick one view from those offered in the drop-down.

If views menu configured above does not have any views for a specific page (for example, no views for Issue Page), you won't be able to configure the default view for it.

Changes take effect when you press **Apply** button.

2.5.6 Copying a Structure

With the **Copy** action, you can create a full copy of a structure, and, optionally, clone every issue in the structure.

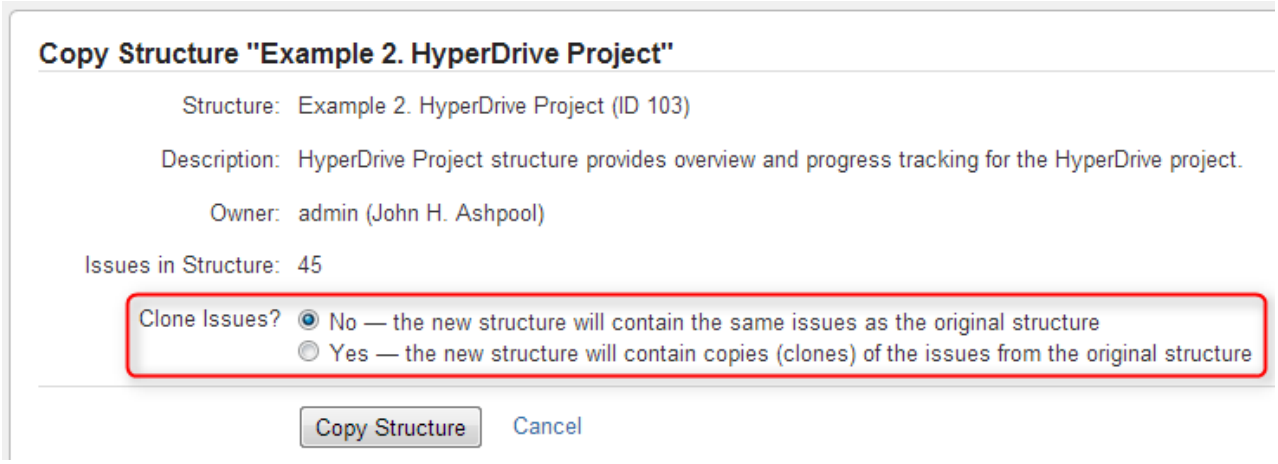
If you need to copy only part of a structure, create a new empty structure and use Issue Clipboard to copy a part of the structure.

To create structure copy:

1. Open **Manage Structures** page using top navigation **Structure** menu.
2. Find the structure you'd like to copy and click **Copy** link in the Operations column.

If you don't see **Copy** in the Operations column, then you probably don't have permissions to create new structures.

3. Choose if you'd like to clone issues.
4. When cloning issues, enter additional parameters for the cloning process.
5. Press **Copy Structure** or **Start Cloning**.



Copying Structure As-Is vs. Cloning Issues

	Copy Structure	Copy Structure & Clone Issues
Selected answer for **Clone Issues?**	**No**	**Yes**
New structure created?	Yes	Yes
New structure contains:	same JIRA issues as the original structure	clones (copies) of the issues from original structure
Quick?	Yes	No, background process is launched to do issue cloning
Permissions required:	**View** access to the original structure **Create Structure** permission	**View** access to the original structure **Create Structure** permission **Bulk Change** global JIRA permission A number of project-level permissions

For details about configuring and running cloning, see Copying Structure and Cloning Issues.

New Structure

The new structure is created with the following properties:

- Structure name is automatically set to "Copy of *<old structure name>* (*<date of copy>*)".
- Structure description is copied.
- View settings are copied.
- You become the owner of the copied structure.
- If you have **Control** access level to the original structure, permission rules are copied. Otherwise, permission rules for the new structure are empty (it is a private structure). To share the new structure, add permission rules.

You can immediately edit new structure's properties on the screen with the copy result.

Copying Structure and Cloning Issues

When copying a structure, you can turn on **Clone Issues** parameter and have Structure plugin create a copy (clone) of every issue in the original structure.

How Issue Cloning Works

Each issue in the original structure is cloned by creating a new issue with the same summary, description, and the same value for every other field, including custom fields. There are a few exceptions to this rule, however:

- The **Status** field is not copied. The cloned issues are always created in the initial status, according to each issue's project and workflow scheme.
- If a field is not present on the Create Issue screen, its value is not copied. The cloned issues will have the default value for that field instead.
- Archived versions are removed from **Affects Versions**, **Fix Versions**, and custom fields that have versions as values.
- If you clone issues to a different project, and some custom fields of the original issues are not available in that project, the values of those custom fields are not copied.
- If you clone issues to a different project, and some field values of the original issues are not available in that project, those values are removed. For example, this may happen to the **Components** field, or to the fields that take versions as values.

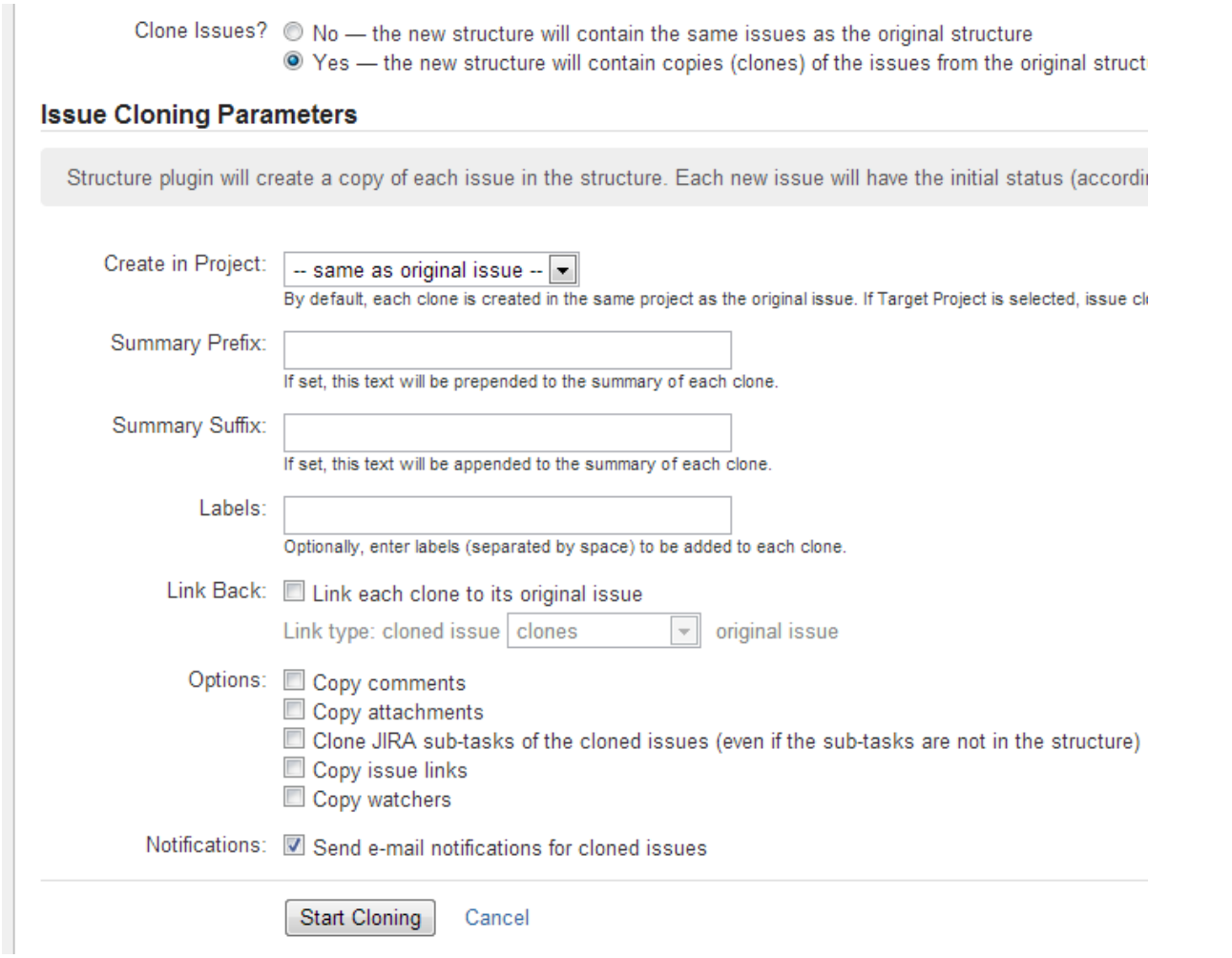
Cloning issues to a different project may even be impossible, for example, when a certain field is required in the target project, but absent (or not required) in the source project. If this is the case, you will need to either change the target project restrictions or make sure that every issue in the copied structure satisfies them.

In any case, Structure does its best to verify that it can indeed clone each issue in the original structure **before** it begins the actual cloning. If Structure detects a potential data loss, for example, because one of the custom fields is absent in the target project, it warns you and lets you decide whether you want to continue. If even a single issue cannot be cloned (for example, if you do not have the **Create Issues** permission for a certain project), then the operation stops before creating any clones.

On a rare occasion when permissions or other restrictions are changed while the cloning operation is in progress, the operation may still fail after the initial checks.

The **Status** field is not copied. The cloned issues are always created in the initial status, according to each issue's project and workflow scheme.

Cloning Parameters



Additional parameters may be specified for the cloning process:

Create in Project	Lets you specify a project for the new issues, different from the project the issues currently belong to. If not specified, every new issue is created in the same project as the original issue.
Summary Prefix **Summary Suffix**	Let you modify the summary of the clones. If the resulting summary gets longer than the JIRA limit (255 characters), it will be truncated.
Labels	Space-delimited labels to be added to the cloned issues. (Already existing labels are preserved.)
Link Back	If specified, every new issue will be linked with its original issue.
Copy comments	If selected, all comments are also copied. If not selected, new issues will have no comments.
Copy attachments	If selected, attachments are copied (the actual files are copied on JIRA server).

Clone JIRA sub-tasks of the cloned issues	Let's say the copied structure contains issue A-1. In JIRA, A-1 has a subtask A-2. But this subtask is not in the structure. If this option is selected, A-2 will also be cloned. If the option is not selected, A-1 will be cloned without the subtask.
Copy issue links	If selected, all issue links and remote issue links will be copied. If a link exists between two issues, which both are cloned, then the new link will be created between clones of the original issues. ⚠ If you use **Link Back** option, then the links of the type selected for linking back to original issues will **not be copied**. If you have JIRA Agile (GreenHopper) 6.1 or later installed, its Scrum board Epic-Story relationships are also copied when you select this option. The rule is the same as for issue links: • If you clone an epic together with its stories, the cloned stories will be added to the cloned epic. • If you clone the stories alone, the clones will be added to the original epic.
Copy watchers	If selected, the users watching an original issue will be added to the watcher list of the clone.
Notifications	If selected, an email may be issued for every created issue, depending on the JIRA notification scheme for the issue's project.

Required Permissions

- To be able to clone structured issues you need **Bulk Change** global permission.
- Because the result of cloning is a new structure, you also need to be allowed to create new structures. (Configured by JIRA administrator - see Administrator's Guide.)
- You need to have **Create Issue** permission in the projects where clones are created. If you specify **Create in Project** option, the issues will be created only in the specified project. Otherwise, clones are created in the same projects as their respective original issues.
- Users in the **Assignee** field of the original issues will have to have **Assignable User** permission in the target project – otherwise, issue clone cannot be assigned to that user and will be assigned by default.
- If you don't have **Modify Reporter** permission, you won't be able to set the value of **Reporter** field in the cloned issues. Instead of the original reporter, you will be the reporter of the issue clones.
- You need to have **Add Comments** permission to copy comments, **Link Issues** permission to copy issue links or use **Link Back**, **Create Attachments** permission to copy attachments, **Manage Watchers** permission to copy watcher lists, and **Edit Issue** permission to copy GreenHopper's Epic-Story relationships.

Executing Bulk Cloning

When you press **Start Cloning** button, a background process starts on JIRA server, which performs the following:

1. Copy original structure's hierarchy and store it in memory.
2. Check all necessary permissions required for cloning.
3. Clone all issues.
4. Create a structure and fill it up with the cloned issues.

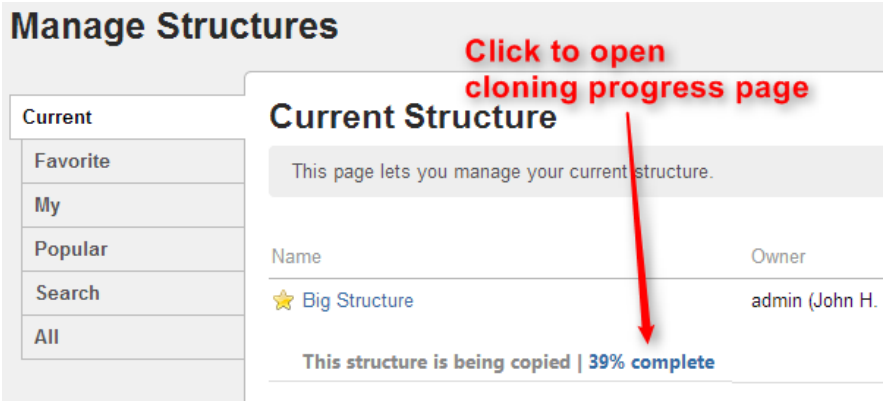
At step 2 the cloner process might discover some problems. If critical problems are discovered, an error message is shown and process is aborted. If non-critical problems are discovered, then warnings are shown and user input is required. The warnings may suggest that cloning may continue, but the resulting issues might not be exact copies. After your confirmation, the process continues.

Cloning issues is potentially a long operation. Cloning a structure with tens of thousands of issues may take an hour or more. Cloning smaller structures usually takes reasonable time.

As cloning proceeds, progress bar is shown on the screen. When cloning is done, the resulting structure is opened for modification of its name and permissions.

Checking Clone Progress

When cloning has started, you can navigate away from the cloning progress page. To see the progress and get back to the progress screen, open **Manage Structures** page and locate your structure. It should show that the structure is being copied.



When cloning is completed, or if there are warnings or questions from the cloning process, the link will read "Waiting for input". Click the link to open cloning progress page.

Cancelling Cloning

You can cancel cloning process from the cloning progress page by pressing **Cancel** link.

Issues that have already been created by the cloning process will be assembled into a special structure marked "*[Cancelled Cloning Result]*" in the structure name. You can use Bulk Change to quickly delete the unwanted issues.

Cloning Queue

Cloning issues can place considerable load on JIRA server. To avoid overloading server with cloning jobs, there is a limit to the number of cloning processes that can happen simultaneously. If this limit is exceeded, your cloning process will initially be in "waiting" state, pending for other cloning processes to finish.

2.5.7 Deleting a Structure

When you delete a structure, the following information is deleted:

- The hierarchical list of issues from the structure
- Structure details - name, description, permissions
- Synchronizers installed into structure

The issues that the structure contains are not affected in any way. They remain in JIRA and still can be part of another structure.

If there's any synchronizer installed for the structure you delete, it will not have a chance to react. So, if removing an issue from the structure should cause synchronization (such as removal of the links, done by the Links Synchronizer), you might need to first manually delete issues from the Structure to let the synchronizer do its job, and then delete the structure itself.

To delete a structure:

1. Open **Manage Structures** page using top navigation **Structure** menu.
2. Find the structure you'd like to delete and click **Delete** link in the Operations column.

You need **Control** access level to be able to delete a structure. See Structure Permissions.

3. Review the structure you are about to delete and confirm the operation. **There's no Undo!**

2.6 Template Structures and Projects

Template structure is a structure that you copy & clone to get the real, "workable" structures.

Technically, template structures are ordinary structures, containing ordinary issues. It is up to you to designate a structure to be a template and configure it accordingly.

2.6.1 Configuring Template Structures

Here are some suggestions about configuring template structures:

1. Clearly designate them as a template - for example, have "[Template]" marker as a part of the structure's name.
2. Give permissions to change the template structure only to those users who really need it. If needed, create another JIRA group for them (or ask JIRA administrator to do so).
3. Do not install any synchronizers on the template structure (unless you want the template to change, of course... which would be a quite unusual case).
4. Do not mark template issues as template in the issue summary. If you need to mark template issues somehow, use a label, which you will be able to remove from cloned templates via Bulk Change.

If you need to remove template issues from a JQL search, you can add to JQL: `AND NOT (issue in structure('template structure name'))`. See structure() JQL function.

2.6.2 Creating Issues and a Structure from Template

Once you have a template structure, you can use Copy action from the **Manage Structures** page and turn on Clone Issues option. For details about configuring and running cloning operation, refer to Copying Structure and Cloning Issues article.

After you have created a new structure with new issues from template, you might want to:

- Rename the new structure and give it a meaningful name.
- Assign permissions for the new structure, if they are different from template structure permissions.
- Open the new structure to make sure it looks good.
- Do a Bulk Change on all issues - for example, to remove a template marker.

2.6.3 Template Projects

In the same manner, you can create a template project with template issues, and put them all into a template structure.

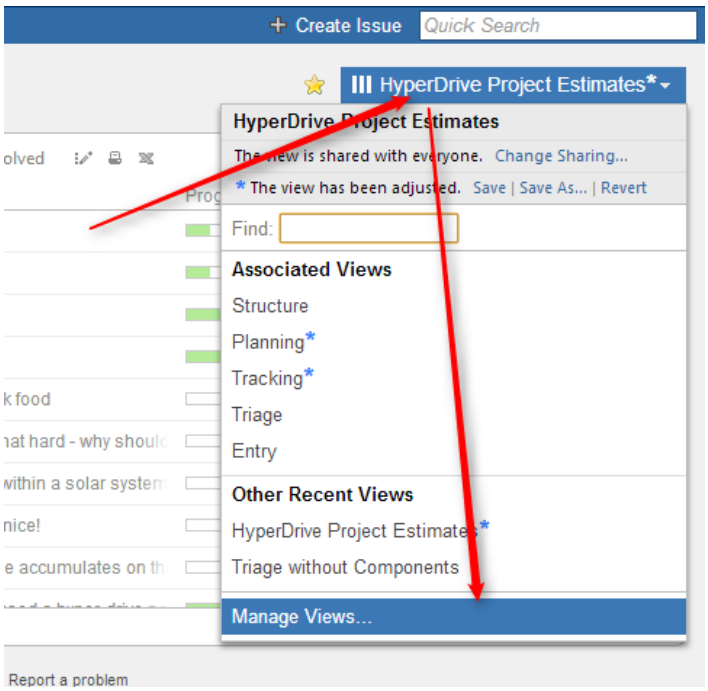
When you need to create a new project based on the template project, do the following:

1. Manually create an empty new project.
2. Create new structure and issues from template structure, as advised above. When configuring cloning parameters, specify the new project in the **Create in Project** parameter.

2.7 Managing Views

A view is a named configuration of the columns in the Structure widget. There are a number of pre-installed views that come with the Structure plugin, and the users may create and share more views.

You can find and select a View via Views Menu. You can also save changes, create a new view and share a view in the same Views drop-down. For other operations with the views, you need to open **Manage Views** dialog:



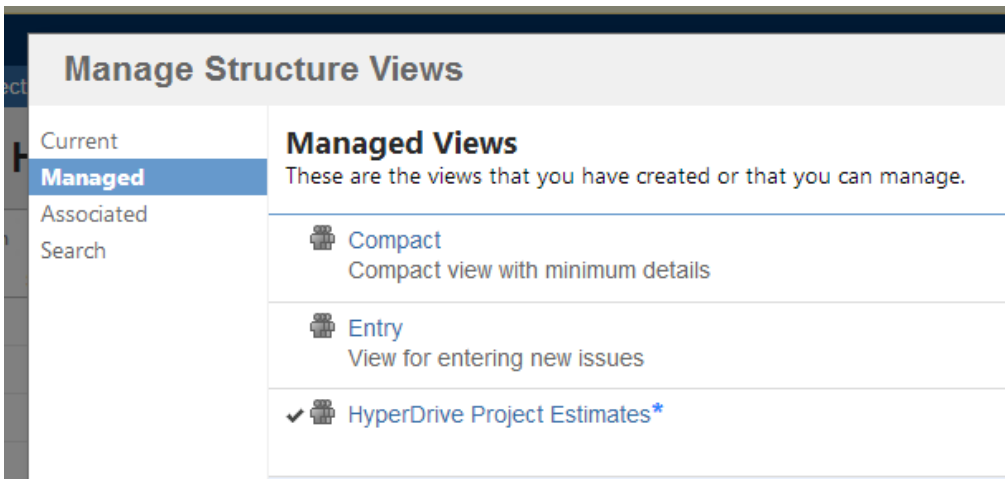
See the following sections for details on view management:

- Locating a View
- Changing View Settings
- View Sharing and Permissions
- Associating Views with Structures
- Copying a View
- Deleting a View

2.7.1 Locating a View

To do anything with a view, first you need to find it.

You can use search box in the Views drop-down, and find a view by name. You can also open **Manage Views** dialog and find the view on one of its tabs:

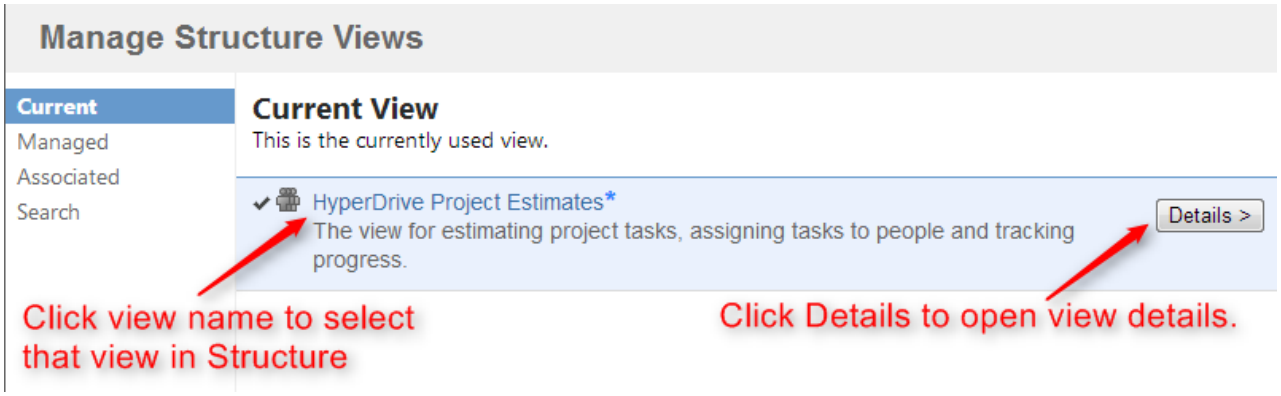


The following tabs are present:

- **Current** tab displays only the view that is currently selected in the Views menu. You can quickly go to the current view's details from this tab.
- **Managed** tab displays all views that you can **Manage** – that is, you have full administrative access to those views.
- **Associated** tab displays all views that are associated with the currently viewed structure (by the structure administrator).
- **Search** tab allows you to find views or list all views.

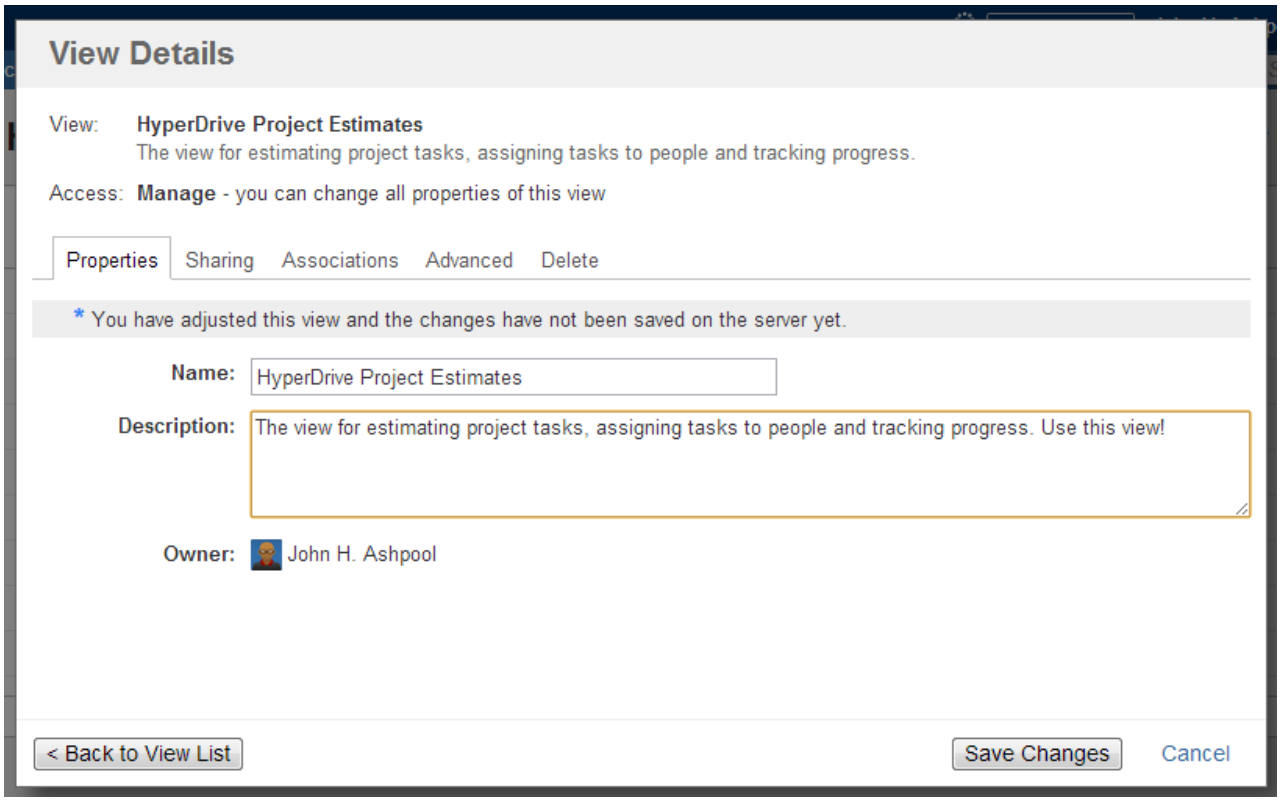
When you have located the view, you can click its name to switch to that view in the Structure widget. The view will also appear in **Also Recently Used** section of the Views Menu.

To see and edit View details, click **Detalis** button that appears when you move mouse pointer over the view record.



2.7.2 Changing View Settings

When you have located a view in the Manage Views dialog, click **Details** button to open View Details page in the same dialog:



View Details page shows a number of tabs:

- **Properties** tab lets you change the name and the description of the view.
- **Sharing** tab lets you view and modify sharing permissions for the view – see View Sharing and Permissions.
- **Associations** tab shows structures, which are associated with the view (have this view in their Views drop-down). See Associating Views with Structures.
- **Advanced** tab shows some technical information about the view.
- **Delete** tab lets you delete this view.

The tabs and the scope of functionality available may be limited, depending on your access level to the view.

Renaming a View and Changing Other Properties

When you change view name, description, sharing permissions, or anything on Advanced tab, the changes are not save until you hit **Save Changes** button. After you have saved the changes, they take effect for you and anyone else who has access to the view.

Associations tab is different – it contains only links to structures. The associations between structures and views are managed by the structure administrator on the Manage Structure page.

2.7.3 View Sharing and Permissions

Like structures, views can be shared with different levels of access for each group of users.

There are four levels of access to a view:

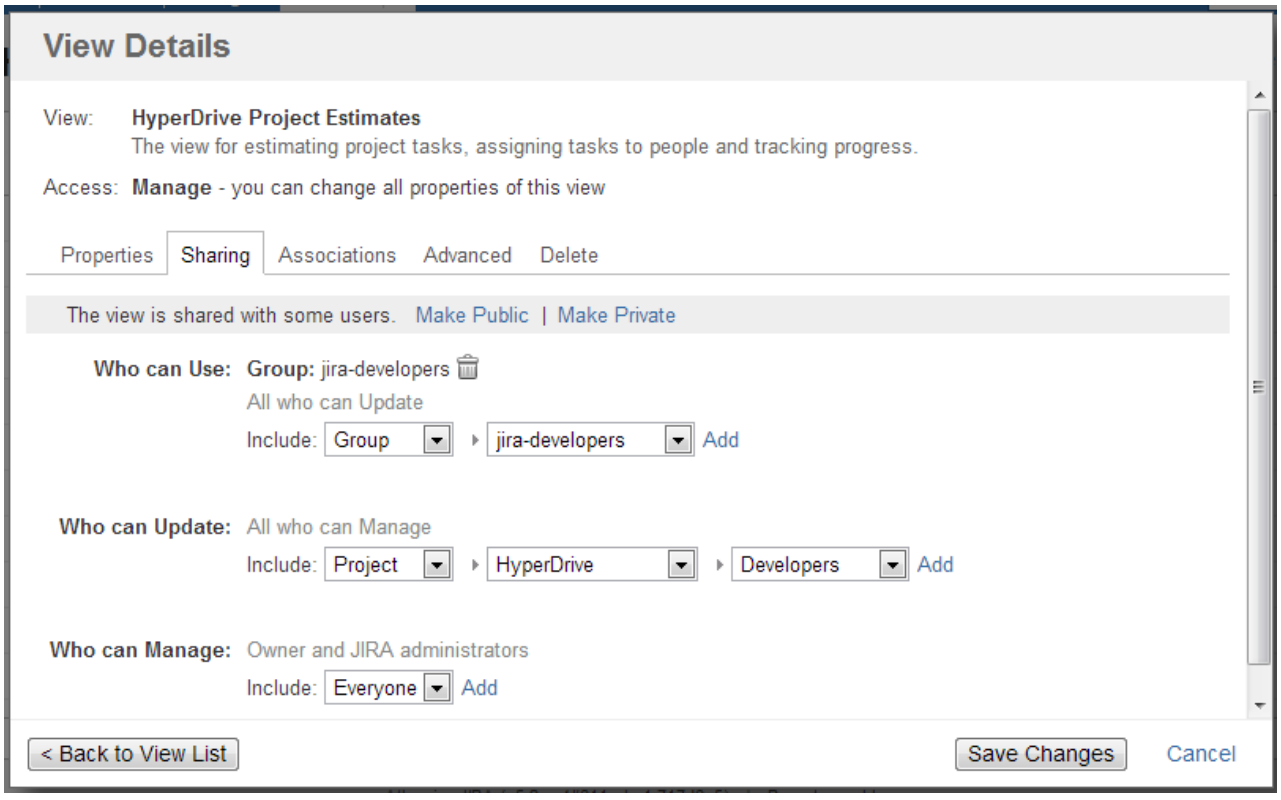
None	The view is not visible nor usable by the user.
Use	Read-only access: the user can use the view, but cannot modify it.
Update	The user can use the view, and also save view adjustments as the new version of the view. The user cannot modify view name or sharing permissions.
Manage	The user can change any of the view's properties and also can delete it.

View owner and JIRA administrators always have **Manage** access to a view.

People who have only **Use** permission for a view still **can** add, remove or rearrange columns, but they won't be able to save the modified configuration as a new version of the view. They will be able, however to use **Save As** link to create a new view with the modified configuration.

Changing permissions

If you have **Manage** access to a view, you can modify its permissions on the **Sharing** tab of the view details dialog.



For each level of access, you can define categories of users who have this type of access:

- Nobody
- Specific user groups

- Specific roles in specific projects
- Everyone (including anonymous users)

Note that higher-level access implies all lower-level access. So everyone who can **Manage** a view, can also **Update** and **Use** it - no need to add those users at all three levels!

Private and Public Views

When a view is not shared with anyone, it's called **private view**. You can quickly make a view private by clicking **Make Private** link – this will have the effect of removing all permission assignments.

When **everyone** is given at least **Use** permission for a view, it is called **public view**. You can quickly make view public by clicking **Make Public** link on the the **Sharing** tab and also in the Views Menu – this will give **Use** permission on that view to everyone.

You need to have global **Create Shared Objects** permission to be able to share views.

2.7.4 Associating Views with Structures

Views, which are associated with a structure, can be easily accessed from the Views Menu when that structure is used – they are always located in the **Associated Views** section of the menu.

Views are associated with a structure by a structure administrator (someone who has **Control** access level to it) on the on the **Manage Structures** page – see Customizing View Settings for details. Also, JIRA administrator may specify global default view settings, which define associated views for structures that don't have customized view settings.

When you look at a view in the **View Details** dialog, **Associations** tab lists all structures that have this view in their view settings. If you have **Manage** access level to some of those structures, you can quickly jump to **View Settings** page for those structures to change the settings.

View settings (associations between a view and a structure) are a property of the structure, not the view. The **Associations** tab on the View Details dialog is provided for convenience.

2.7.5 Copying a View

There's currently no way to directly copy a view, but you can use **Save As** function to create a new view based on the existing view configuration:

1. On the Structure Board, select a view you'd like to copy so it is your current view. You can use Views Menu or Manage Views dialog to find the view you need.

2. If you don't have local adjustments to the view, make some – for example, add a column, or change column order. (Note that just resizing a column does not change view configuration.)
3. Open views drop-down menu and use Save As link to create a new view.
4. Use **Manage Views** dialog to review the new view's description and sharing permissions.

2.7.6 Deleting a View

To delete a view:

1. Open **Manage Views** dialog.
2. Locate a view you'd like to delete and click the view record or **Details** button.
3. Open **Delete** tab and use Delete button to delete the view.

Deleting a view cannot be reverted.

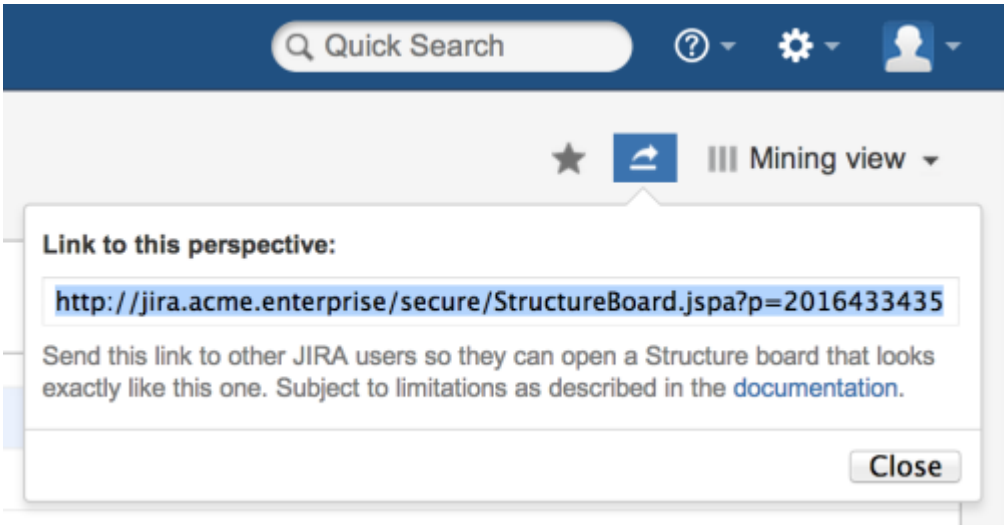
Open **Associations** tab and take a look at the list of structures that are associated with this view. The associations won't stop you from deleting the view, but you might want to discuss the matter with administrators of those structures.

2.8 Sharing a Perspective

Structure Perspectives is a feature that lets you store the way you see a structure in the form of a permanent link which can be published or sent to another person.

To create a perspective:

1. Open Structure Board
2. Click the "Share perspective" button. A message with the link will appear:



3. Copy the link and save it, publish it, or send it to persons you want to share it with.

To use a perspective:

1. Follow the link you've received. This will open a Structure Board in Perspective mode, that is, the Structure Board will look mostly the way it looked when the perspective was created.

   > A special **Perspective** View will be automatically selected. It represents the column configuration that was in use when the perspective was created. It is temporary and read-only. You can modify it, make it permanent by saving it under a new name, or switch back to some other view.

When creating a perspective, please consider the following:

1. If you send the link to someone who has no access to the Structure Plugin in general, or to the individual structure for which the perspective was created, they won't be able to use your link.
2. If the structure contains issues accessible to you but not to the recipient, they will not see them in the structure, even in Perspective mode.
3. Issue hierarchy is not stored in the perspective. A structure opened in Perspective mode will always show the current issue hierarchy (it will contain all the changes that were made after the perspective was created).

   > You can open structure history and select the latest change before creating a perspective. This way, users will always see structure in history mode when opening your perspective.

4. If a perspective is not accessed for some time, it may be automatically removed from the system. This behavior can be configured or disabled by JIRA administrators via Structure Maintenance.
5. Once created, a perspective becomes accessible to any person who has access to the structure for which that perspective was created.

2.9 Synchronization

Synchronization lets you keep Structure issue hierarchy in sync with some other issue properties. For example, you can enforce the rule that JIRA sub-tasks should always be placed under their parent in the structure, or that there should be an issue link from parent issue to each sub-issue.

Synchronization can also be run once to perform a one-time update of the structure (Import) or one-time update of the issues based on the structure (Export).

Synchronization is extendable system that allows JIRA plugins provide their own synchronizers. The following synchronizers are supplied with the Structure plugin:

- Sub-Tasks Synchronizer places JIRA sub-tasks under their parent issues in the structure
- Links Synchronizer makes sure that sub-issues are linked to their parent issues with a specific link type, and it also can be used to reconstruct structure from links
- Filter Synchronizer populates structure with issues that pass a saved filter, it also can be used to remove issues from structure when they no longer pass the filter

- JIRA Agile (GreenHopper) Synchronizer works to sync JIRA Agile ranking of issues with their position in the structure and to make it easier to assign stories to epics by using Structure
- Status Rollup Synchronizer allows to make propagate issue status upwards, for example, make parent issue Resolved if all sub-issue are resolved.

One-time synchronization works when you run Export or Import, or when you run a Resync. Automatic synchronization runs in the background and listens for the updates in the structure and beyond.

Please be careful: synchronization may cause massive changes on the issues. For example, if you install JIRA Agile (GreenHopper) synchronizer and then add issues to the structure in some random order, those issues' ranks will be changed according to that order almost immediately! We plan to add "synchronization undo" in the future version to reverse possibly uncalled bulk changes. But for now, please make sure you have daily backups and carefully read how a synchronizer works before installing it

In order to install a synchronizer you need to have **Control** permissions on a structure and have necessary permissions on the JIRA issues.

Note that you also need to have special permission to **Control Synchronizers**. If there is a warning message above the feature description please contact your *JIRA Administrator* for assistance.

Anonymous user cannot install synchronizers or use Export/Import even they are granted Control permissions.

2.9.1 Importing Structure

When you **Import** a structure, you get a set of issues which should be in the structure and/or information on how they should be arranged in a hierarchical list. Then this information is applied to an existing structure.

For example, you can use a Filter Synchronizer to add All Open Issues to a structure (or the results of whatever Saved Filter you have), or JIRA Agile (GreenHopper) Synchronizer to rearrange issues in the structure according to their rank and epic in JIRA Agile.

To run Import, you must have **Control** permissions on the structure and additional permissions may be required by a specific synchronizer.

To import hierarchy into a structure:

1. Open **Manage Structures** page using top navigation Structure menu.
2. Select the structure you'd like to import into and click **Import** link.

If you don't see **Import** link in the Operations column, then you possibly don't have Control permissions for this structure.

3. Select a synchronizer from the drop-down list and proceed to configure import parameters.

   If there are no synchronizers in the drop-down list, then either none are currently installed or none of the installed synchronizers support import into a structure.

4. Enter synchronizer parameters. Each synchronizer has its own parameters, so please refer to specific synchronizer documentation. If you're not yet acquainted with how this specific synchronizer works, please read the Rules section on the parameters page.
5. Click **Run Import**. When you start import, synchronizer will analyze data and possible update the whole structure.
6. After you click Run Import and confirm the operation, a job status page is shown. When the job is marked as Finished, the synchronization is done and you can view the results by opening the affected structure.

When import is run, it runs under your user name and with your permissions. So if you don't have enough permissions to read values somewhere else or to view issues you'd like to import, you may not see the expected result.

Import and Export are actually a one-time Resync. Export is resync from Structure and import is resync into Structure. If you need to run export or import periodically, you can set up a synchronizer with all the parameters but without enabling it - so no synchronization happens in the background. When you need to export or import, you can open Synchronization Settings page for the structure and run Resync. Just make sure you've selected the correct Resync direction!

2.9.2 Exporting Structure

When you **Export** a structure, you use the hierarchy from the structure to create, update or delete other issue attributes or make any other changes based on the hierarchy that a specific synchronizer provides.

For example, you can use Links Synchronizer to create issue links between sub-issues and their parents.

To run Export, you must have **Control** permissions on the Structure, and you will likely need some additional permissions, depending on which synchronizer you're going to use. For example, you have to have *Link Issues* permission when working with the Links synchronizer.

To export hierarchy from a structure:

1. Open **Manage Structure** page using top navigation Structure menu.
2. Select the structure you'd like to export from and click **Export** link.

> If you don't see **Export** link in the **Operations** column, then you possibly don't have **Control** permissions on this structure.

3. Select a synchronizer from the drop-down list and proceed to configure export parameters.

> If there are no synchronizers in the drop-down list, then either none are currently installed or none of the installed synchronizers support exporting from a structure.

4. Enter synchronizer parameters. Each synchronizer has its own parameters, so please refer to specific synchronizer documentation. If you're not yet acquainted with how this specific synchronizer works, please read the *Rules* section on the parameters page.
5. Click **Run Export**. When you start export, the synchronizer will read the current structure and apply it to whatever it syncs with.
6. After you have clicked Run Export and confirmed the operation, a job status page will be present. When the job is marked *Finished*, the synchronization is done and you can inspect the results.

> When export is run, it runs under your user name and with your permissions. So if you don't have enough permissions to make a certain change in JIRA, the synchronizer will skip that change (a warning will be printed out in the server logs).

> Import and Export are actually a one-time Resync. Export is resync from Structure and import is resync into Structure. If you need to run export or import periodically, you can set up a synchronizer with all the parameters but without enabling it - so no synchronization happens in the background. When you need to export or import, you can open Synchronization Settings page for the structure and run Resync. Just make sure you've selected the correct Resync direction!

2.9.3 Installing Synchronizer

When you install a synchronizer on a structure, you make this structure automatically sync with something else. For example, after you have installed and enabled a Links Synchronizer, any changes someone makes to the structure will cause issue links to be created or deleted to match those changes. Or when you have installed and enabled Filter Synchronizer in *Add* mode, creating or changing an issue that causes it to pass the selected saved filter will cause this issue to be added to the structure.

When you install a synchronizer, you define its parameters. Those parameters can be edited later.

Please note that after a synchronizer is installed, it's not working yet - it must be **Enabled** to start monitoring the changes.

To install a new synchronizer:

1. Open **Manage Structure** page using top navigation Structure menu.
2. Find the structure you'd like to sync. The **Sync With** column shows currently installed synchronizers. Click on the **Settings** link in that column.

   > If you don't see **Settings** link in the **Sync With** column, then you possibly don't have **Control** permissions on this structure.

3. Synchronization settings page shows detailed information about each installed synchronizer and lets you work with them. To proceed with the installation of a new synchronizer, select the type of the synchronization and click **Configure and Install Synchronizer**.
4. Enter synchronization parameters. Each synchronizer has its own parameters, please refer to the specific synchronizer documentation.

   > If you're not acquainted with how this synchronizer works, please make sure to read the *Rules* section at the top of the page. Especially text in red.

5. Press **Create** button and the synchronizer gets installed. However, it's not enabled yet.
   a. Before synchronization is enabled, you might want to run Resync to bring the current state of the structure and JIRA to the same page. In that case, press **Resync and Enable** button after the synchronization is installed, or later use the same link on the synchronization settings page.
   b. If you need to enable synchronization without resyncing first, press **Enable without Resyncing**.
   c. You can enable and resync the synchronizer later from the synchronization settings page. Press **Done** if you don't need to enable the newly installed synchronizer now.

> Import and Export are actually a one-time Resync. Export is resync from Structure and import is resync into Structure. If you need to run export or import periodically, you can set up a synchronizer with all the parameters but without enabling it - so no synchronization happens in the background. When you need to export or import, you can open Synchronization Settings page for the structure and run Resync. Just make sure you've selected the correct Resync direction!

2.9.4 Modifying Synchronizer

You can change the parameters of a synchronizer to alter how it works. If the synchronizer is enabled (and so, working in the background), it first needs to be stopped.

Changing synchronizer's parameters may completely change the result of synchronization. That's why the synchronizer first needs to be stopped, and after the parameters are changed, Resync is recommended.

To edit a synchronizer:

1. Open **Manage Structure** page using top navigation Structure menu.
2. Find the structure of the synchronizer you'd like to edit. Click on the **Settings** link in the **Sync With** column.

   If you don't see **Settings** link in the **Sync With** column, then you possibly don't have **Control** permissions on this structure.

3. Find the synchronizer you'd like to edit. Click on the **Edit** link or the **Disable and Edit** link in the **Operations** column. The synchronizer will be automatically disabled.

   If you don't see neither the **Edit** link nor the **Disable and Edit** link in the **Operations** column, then the synchronizer is probably provided by the third-party plugin and does not support editing.

4. Set the new synchronizer parameters. Also, if you are a **JIRA Administrator** and not the synchronizer owner, choose if you want to become the new synchronizer owner.
5. Press **Apply** button. This will update the synchronizer parameters and make you the new owner of the synchronizer (unless you chose not to do so in the previous step). However, the synchronizer is not enabled yet:
   a. Before synchronization is enabled, you might want to run Resync to bring the current state of the structure and JIRA to the same page. In that case, press **Resync and Enable** button, or later use the same link on the synchronization settings page.
   b. If you need to enable synchronization without resyncing first, press **Enable without Resyncing**.
   c. You can enable and resync the synchronizer later from the synchronization settings page. Press **Done** if you don't need to enable the updated synchronizer now.

2.9.5 Removing Synchronizer

You can remove an installed synchronizer at any time if you have **Control** permissions on the structure.

1. Open **Manage Structure** page using top navigation Structure menu.
2. Find the structure you need to remove a synchronizer from. You can look at *Sync With* column to see which synchronizers are installed in a structure. Click **Settings** link for the selected structure.
3. Find the synchronizer in the list and use **Delete** link to remove it.

Note that if the synchronizer is currently performing an incremental sync or resync, it will be allowed to finish.

2.9.6 Turning Synchronizer On and Off

A synchronizer is disabled by default and it's usually explicitly enabled after it is installed into a structure, probably with a resync. The following list summarizes the possible states of a synchronizer:

- **Disabled** - the background incremental synchronization is not running. You can run resync to do a one-time full sync.
- **Enabled** - the background incremental synchronization is running. When a change is detected, synchronization applies the change to the other part of the synchronous link as soon as possible, typically within several seconds.
- **Not Available** - the synchronizer is installed but it cannot run any synchronization. The possible reasons are changes in JIRA configuration or lack of permissions from the user.

To **disable** an active synchronizer:

1. Open synchronization settings page for the structure.
2. Find the synchronizer and click **Disable** link.

> If the synchronizer is currently running a sync, it will be allowed to finish.

To **enable** an inactive synchronizer:

1. Open synchronization settings page for the structure.
2. Find the synchronizer and click **Enable** link.
3. Alternatively, you can click **Resync and Enable** to resync and enable immediately after resync finishes.

2.9.7 Running Resync

A resync, or full resynchronization, is a one-time process activated manually by the user to bring Structure and some other aspect of issues to the same page. Resync typically scans all issues that may be affected - contrary to the incremental synchronization, which inspects only issues that have been changed.

For example, running resync on a Saved Filter synchronizer (in *Add* mode) runs the related Saved Filter and makes sure all issues from the result set are in the structure. When the same synchronizer is working in the background, it checks only those issues that have been changed.

Resync Directions

Resync is also different from incremental synchronization in that it has a direction. The incremental sync tries to apply changes on *both* sides to the other side, if possible, depending on where the change has happened: with JIRA Agile (GreenHopper) synchronizer, if you change the rank (issue position in backlog on the Planning Board), its position in the structure is also changed; and if you change the position in the structure,

GreenHopper's rank is changed. However, when applying Resync, you need to choose which side of the data is to be taken as the final version and which is to be updated.

Resync can be run:

- *from Structure*, which means that the issue hierarchy in the Structure is the final data and the synchronizer should update whatever it syncs with. This is what happens when you export from a structure.
- *into Structure*, which means that the issue hierarchy is going to be updated (or issues possibly added or removed), and whatever the synchronizer syncs with has the final say. This is what happens when you import into a structure.

A synchronizer may support resyncing in only one direction. For example, Saved Filter synchronizer, which adds issues from a saved filter result, can only sync *into* Structure.

Running Resync

To run a resync:

1. Open **Manage Structure** page using top navigation Structure
2. Click **Settings** link in the *Sync With* column for your structure
3. On the synchronization settings page, find the synchronizer you'd like to Resync, and either
    a. Click **Resync**
    b. Click **Resync and Enable** if the synchronizer is disabled and you'd like to enable it immediately after Resync finishes
4. Select a direction for the Resync. For example, *JIRA Agile (GreenHopper) Structure* means that the data will be taken from JIRA Agile and the structure will be rearranged. If a direction is not supported by the synchronizer, it will be disabled.
5. Click **Start Resync**.

    Resyncing in a wrong direction may lead to data loss! Please make sure you understand that you're doing the correct thing and confirm running the resync when a confirmation dialog appears.

6. The job status page that appears will tell you when the Resync has finished.

If the synchronizer is currently running an incremental synchronization, the resync will wait until it finishes.

2.9.8 Synchronization and Permissions

IMPORTANT! Please read.

When synchronizer makes changes to bring structure and some other aspect of issues to the same page, it should do that on behalf of a certain user - for the sake of permissions and logging. This user is the one who has created the synchronizer, and it is displayed in *Run as User* column on the synchronization settings page.

Due to asynchronous nature of the synchronization, the changes that cause sync may be effected by a different user or users. **However, when sync runs, the updates will be made on behalf of the user who installed the synchronizer!**

This is really important to understand. Consider the following settings:

- You create a Structure and you set up structure permissions so that anyone can edit the structure.
- You have Link Issues permissions on a project and you install Links synchronizer to have children issues linked to their parent issue.

Now, anyone can edit the structure - add issues there, remove issues from there and rearrange the issues in the structure. **Every change of the structure will lead to adding and removing links between the affected issues on your behalf - even if the user who changes the structure does not have Link Issues permission!**

So when using synchronizer, Structure edit permissions implicitly grant limited permissions to make changes according to the synchronizer's algorithm, as well as issue permissions implicitly grant limited permissions to edit the structure.

2.9.9 Protection from Synchronizer Cycles

It is possible to accidentally create a pair of synchronizers that would contradict each other. For example, a sub-tasks synchronizer can be configured to put a sub-task under an issue, while a links synchronizer with the "links primacy" option would have to move it to the top level of the structure.

If both such synchronizers are enabled (i.e. perform automatic synchronization), they will end up in an endless cycle, processing and overriding one another's changes, forever. These situations are undesirable, because they put unnecessary load on the server and quickly fill up issue and structure histories with meaningless change records.

Structure is designed to detect and stop such infinite cycles. In the background, Structure keeps track of how many times each of the enabled synchronizers has been invoked to process the changes generated by another synchronizer. If this number passes a certain threshold (10 by default), and there were no user-generated changes between the invocations, Structure will flag this as a probable conflict, and perform one or more of the following actions, depending on the configuration:

- Prevent one of the synchronizers from running this particular time, but keep both synchronizers enabled.
- Disable both of the synchronizers involved in the cycle.
- Send e-mail notifications to the user (or users) who created the conflicting synchronizers.

- If the synchronizers are not automatically disabled and keep cycling, send e-mail notification to the JIRA administrators (all users having the "System Administrator" global permission).

The default behavior is to disable the conflicting synchronizers and send e-mail notifications to the users who installed them.

> **How do I respond to a cycle warning?**
>
> If you've installed a Structure synchronizer and then receive a cycle warning e-mail from Structure, please take appropriate measures to reconcile the synchronizers – disable or reconfigure one or both of them. If the second synchronizer was created by a different user, you may need to cooperate with them to solve the problem. If you're not sure what to do, contact your JIRA administrators or ALM Works support team.

JIRA administrators can configure the cycle guard as described in the Administrator's Guide.

2.9.10 Bundled Synchronizers

There are several bundled synchronizers coming with the Structure. Other synchronizers can be provided by other JIRA plugins.

- Sub-Tasks Synchronizer — Sub-Tasks Synchronizer lets you have sub-tasks automatically placed under their respective parent issues in the structure.
- Filter Synchronizer — Filter Synchronizer lets you automatically add issues to structure or remove issues from structure based on a Saved Filter or a JQL query.
- Links Synchronizer — Links Synchronizer maintains issue links between parent issue and children issues.
- JIRA Agile (GreenHopper) Synchronizer — JIRA Agile (GreenHopper) Synchronizer lets you synchronize the position of issues in the structure and on an Agile board (such as a Scrum or Kanban board) using Rank synchronization, and synchronize an Epic field with the position of stories under epics in the structure.
- Status Rollup Synchronizer — Status Rollup synchronizer automatically aggregates statuses of the sub-issues and updates the status of the parent issue. For example, it can make parent issue Resolved if all sub-issues are Resolved.

Sub-Tasks Synchronizer

Sub-Tasks Synchronizer lets you have sub-tasks automatically placed under their respective parent issues in the structure.

> This synchronizer is available only when Sub-Tasks are enabled in your JIRA and you have at least one Sub-task issue type defined.

Sub-Tasks Synchronizer Parameters

You can select which sub-task issue types the synchronizer works with. Issues of other issue types will not be affected.

This synchronizer supports only Import / Resync into Structure (*more about resync*).

Sub-Tasks Synchronizer Rules

- When there's a sub-task (of one of the selected types) and its parent issue is in the structure, the sub-task is also added to the structure and placed under its parent task.
- The parent issue must be in the structure already - the synchronizer does not add parent AND sub-task, neither does it add parent for the sub-tasks already added.

> You can add parent issues to structure manually, or use Saved Filter synchronization to add parent issues (and probably sub-tasks) automatically.

- If a sub-task is already in the structure, and is located under a different parent (or at the top level), it will be moved under its *subtask parent* (with all sub-issues that it may have).
- Changes in structure are not synced back to sub-tasks: if you place an issue under another issue, it will not become a sub-task.
    - If you move a sub-task away from its parent task, it will soon be moved back by the synchronizer.

Filter Synchronizer

Filter Synchronizer lets you automatically add issues to structure or remove issues from structure based on a Saved Filter or a JQL query.

This powerful synchronizer lets you control the contents of the structure with an issue filter (either a **Saved Filter** or an arbitrary **JQL Query**). You can either add issues to structure automatically, or remove issues from structure automatically, or do both.

Filter Synchronizer Parameters

Filter	A Saved Filter or a JQL Query to sync with. Click **Select** to choose a saved filter or switch to **JQL query** and enter the JQL. When this synchronizer is enabled and runs in background, it "listens" to JIRA events about issues being changed. That means that if the result of a query may change without an issue being actually changed, the synchronizer will not detect the change and will not updated the structure. For example, if you use JQL query `updatedDate > startOfMonth()`, the synchronizer will not update the structure at the beginning of a month, when the result of the query changes. You will need to do a Resync or use scheduled synchronization.
Add	Turns on **Add Mode**: the synchronizer will make sure that all issues from the filter's result are present in the Structure.
Place added issue at the top level	The newly discovered issues from the filter result are placed at the top level, at the end of the structure.
Place added issue as a sub-issue of ...	You can enter issue key (like PROJECT-123) of an issue that will serve as the parent for the newly discovered issues from the filter. They will be placed as children of this issue, at the end of the current children list. Note that if this issue is not present in the structure, the issues won't be added at all.
Allow move	This option is available if you have specified the parent issue for adding matching issues. This option tells the synchronizer what to do if a discovered matching issue is already added to the structure, but is located somewhere else, not under the designated parent issue. If the option is on, the synchronizer will move the issue (with all its possible sub-issues) under the parent issue. If the option is off, the issue will be left alone where it now resides.
Remove	Turns on **Remove Mode**: the synchronizer will remove issues from the structure when they no longer are present in the filter result. However, if an issue to be removed contains sub-issues that should stay in the structure, it will not be removed.
Remove only from where added	Additional flag to remove issues only if they are either at the top level or under the issue where they were initially placed by the synchronizer. So if you move an automatically added issue somewhere else, it will not be removed even if it is no longer present in the search result.

issues are placed	

This synchronizer supports only Import / Resync into Structure (*more about resync*).

If the Saved Filter used in configuration is deleted later, or if you lose permissions to run it, the synchronizer will not work.

No matter how synchronizers are configured, they will only affect issues from the projects that are enabled for synchronization.

CAREFUL! Please be careful when turning on Remove mode and installing another synchronizer into the same structure. It is possible to set up the structure synchronizers in a way to make them cycle: some other synchronizer, like Sub-tasks synchronizer, would add an issue to the structure and then Saved Filter synchronizer in Remove or Add/Remove mode would remove that issue, and so forth.

Filter Synchronizer Rules

- Synchronizer adds issues from its filter's result to structure and/or removes issues that no longer are in the filter's result.
- Whenever an issue changes, a query is run to see if it matches the filter. On resync, all issues are checked.
- With **Add** mode on, an issue will be added to the structure if it matches the filter - even if the user has manually removed it from there. If the issue is already in the structure, it will not be affected, unless **Allow Move** is on - in which case it will be relocated under the specified parent issue.
- With **Remove** mode on, an issue will be removed from the structure if it does not match the filter - even if the user has manually added it before.
- When adding issues on Resync or Import, synchronizer places them at the end of the structure (or at the end under the specified parent issue), in the order that corresponds to the filter's order. However, if only part of the filter result is added (for example, because other issues are already in the structure), the final sequence of issues may be different from the filter result.

Links Synchronizer

Links Synchronizer maintains issue links between parent issue and children issues. You can use this synchronizer to replicate the hierarchy in the structure with issue links, or to import a hierarchy that was previously created with links.

Links synchronizer is available only when Links are enabled and there's at least one link type.

This synchronizer supports Resync in both directions (Import and Export) (*more about resync*). Incremental synchronization watches both structure changes and issue link changes and applies the change to the other side (unless **Reverse contradicting changes** option is specified, see below).

No matter how synchronizers are configured, they will only affect issues from the projects that are enabled for synchronization.

When synchronizer adds or removes JIRA issue links, it has the same permissions as the user that installed the synchronizer.

Links Synchronizer Parameters

Link Type

The type of the link to sync with. Links of other types will be ignored.

Link Direction

Defines which side of the link is the parent issue and which is the sub-issue.

Parent Issue Filter and Sub-Issue Filter

If set, these filters determine which issues and links can be affected by the synchronizer:

- If a link's parent issue or sub-issue (as determined by **Link Direction**) doesn't pass the corresponding filter, then the link is ignored by the synchronizer, as if it didn't exist.
  In particular, if there are two issues that belong to the structure and pass the corresponding filters, and one of them falls out of its corresponding filter, the link will not be deleted.
- If there is a parent issue and a sub-issue in the structure, and either of them doesn't pass the corresponding filter, the synchronizer will not create a link between them.

You can use saved filters or JQL queries.

Scope

Defines which issues are affected by the synchronizer, based on whether they are in the structure or not.

- **Synchronize issues that are already in the structure** means that the synchronizer will affect only those issues that are already in the structure or reachable from it via issue links. Use this option when you need manual control over which of the linked issues appear in the structure.
  - If **Expand to sub-issues** is selected, the synchronizer will add sub-issues to the structure if their parent issue is in the structure.
  - If **Expand to parent issues** is selected, the synchronizer will add a parent issue to the structure if any of its sub-issues is in the structure.
- **Synchronize all issues that have links of selected type** means that the synchronizer will affect all issues that have matching issue links and pass the **Issue Filters**. For example, you can use this option to import all issue relationships represented by links into an empty structure.

This setting also controls which issue links can be deleted during export, manual resync *from* structure, or incremental synchronization. For example, when you remove a sub-issue from the structure, the synchronizer will remove the corresponding link only if it could have added this sub-issue back, that is, when either **Expand to sub-issues** or **Synchronize all issues** is selected.

CAREFUL! Please be careful when using this synchronizer with **Synchronize all issues** option selected, because Exporting or Resyncing *from* Structure would delete all the existing links of the selected type between issues that are not in the corresponding positions in the structure.

Removal

Defines how the synchronizer treats a sub-issue that doesn't have a link to justify its position in the structure (for example, when a user deletes the link from its parent issue):

- When **Move upwards** is selected, the synchronizer will move such an issue up the hierarchy until it's either at the top level of the structure or in a position that doesn't contradict the settings (for example, under an issue that does not pass the **Parent Issue Filter**).
- When **Remove** is selected, the synchronizer will remove such an issue from the structure, together with all its sub-issues.

Primacy

By default, when a synchronizer is installed and enabled, it tracks changes made by users and applies them to the "other side":

- When a user creates or deletes issue links, the synchronizer adjusts the structure accordingly.
- When a user changes the structure, the synchronizer creates or removes the corresponding links.

You can use the **Reverse contradicting changes** option to override this behavior and specify the primary place where issue relationships are stored:

- With **Structure primacy**, when a user creates or deletes a link that is within the scope of the synchronizer, but contradicts the structure, that change will be reverted. One needs to change the structure to adjust issue relationships.
- With **Links primacy**, the synchronizer reverts changes to the structure that contradict issue links. One needs to change the links to adjust an issue's position within the structure. Note that this does not apply to reordering issues without changing their parents.

Please note that this option does not apply during Export, Import or manual Resync.

Links Synchronizer Preserves Links Between Added List of Issues

There is a special case: when a list of 2 or more issues is added to the structure, links between these issues are preserved, and they form a hierarchy according to these links. Such a situation may arise, for example, when searching outside the structure and moving a bunch of issues into the structure.

This differs from the default behaviour when **Reverse contradicting changes** option is not selected: normally, if an issue *A* is added to the structure as a sub-issue *B*, and both of them pass the Issue Filters, Links synchronizer would establish a link between *A* and *B* and remove all other links of the corresponding type where *B* is on the sub-issue end of the link. When a list of issues is added, however, the synchronizer behaves as if **Links primacy** was selected.

Links Synchronizer Rules

- When synchronizer is enabled:
  - Changes in the structure will be reflected by creating and removing links of the selected type.
  - Links created or removed by the user will be automatically reflected in the structure.
- Links created and removed by the synchronizer are not recorded in the issue history, and issue update time is not changed (due to performance reasons).
- Use Resync (*from* Structure to Links) or Export to update the links according to the structure.
  - If **Synchronize all issues** is selected, all other links of the selected type will be deleted.
  - Otherwise, the links that are reachable from the structure considering **Expand to...** options, but not represented in the structure, will be deleted.
- Use Resync (from Links *into* Structure) or Import to add and rearrange the issues in the structure according to the existing links.
  - If **Synchronize all issues** is selected, all issues with matching issue links will be added to the structure.
  - Otherwise, only the issues reachable from the structure considering **Expand to...** options will be added.
- Links that violate hierarchy restrictions are treated as follows:
  - If a sub issue has more than one parent issue, the most recent issue link is used.
  - If there is a sub-issue cycle, the oldest issue link is not used.
  - There is an exception to the two preceding rules: Links synchronizer prefers to use links between added list of issues, even if they are older than others.
  - Unused links are deleted during incremental synchronization, and ignored during Import or manual Resync.

JIRA Agile (GreenHopper) Synchronizer

JIRA Agile (GreenHopper) Synchronizer lets you synchronize the position of issues in the structure and on an Agile board (such as a Scrum or Kanban board) using Rank synchronization, and synchronize an Epic field with the position of stories under epics in the structure.

JIRA Agile Synchronizer Parameters

	Agile Board mode (GreenHopper/JIRA Agile 6.1+ only) parameters:
Synchronize	Choose mode of operation *(GreenHopper/JIRA Agile 6.1+ only)*: • Use Agile Board configuration (this feature is available only with JIRA Agile/Greenhopper 6.1+) • Use custom projects and fields configuration
Agile Board	JIRA Agile board to synchronize with. The issues matching Board query will be synchronized. The structure may contain other issues, they will not be affected. If Ranking is turned on by ORDER BY clause in the query, it can be used for synchronization.
Synchronize Epics	If checked, epics will be synchronized with JIRA Agile epics.
Synchronize Rank	If checked, and Ranking is enabled for Agile Board, it will be synchronized with Structure.
	Custom issue set mode and GreenHopper 6.0 and earlier parameters:
Project	A project that JIRA Agile is used in. The structure may contain issues from other projects, they will not be affected. **GreenHopper 5.8 or later:** Multiple projects may be selected. The issues from all selected projects will be synchronized using the same Global Rank field.
Rank Field	The field of type "Rank" (managed by JIRA Agile) that holds the rank (backlog order) for the selected Project. If you do not wish to synchronize rank, select *Don't synchronize*.
Epic Field	The field holding the Epic that the story belongs to. • If you use epics on the Scrum boards in GreenHopper 6.1 and up, select "Scrum Board Epics" as the Epic field to synchronize them. • If you use the Classic Planning Board, pick the appropriate custom field of type "Labels", which is typically named "Epic/Theme". ⓘ The synchronizer allows to select an Epic/Theme field even if it is applicable only to some of the available issue types. When the synchronizer should set a value to an Epic/Theme field, it will not make a change if the field is not applicable to the issue type of the changed issue. • If you do not wish to synchronize Epics content, select *Don't synchronize*.
Epic Type	

	Agile Board mode (GreenHopper/JIRA Agile 6.1+ only) parameters:
	Relevant only if an Epic Field is selected. Defines an issue type that is treated as Epic - typically named "Epic". All issues placed under an issue of this type in the structure will be updated to have Epic Field point to that issue.
Auto-add Subtasks	When turned on, sub-tasks will be automatically added to the structure and forced to stay under their respective parent issues. This works similarly to Sub-Tasks Synchronizer.

This synchronizer supports both Import and Export / Resync into/from Structure (*more about resync*). Incremental synchronization watches both structure changes and JIRA Agile's changes and applies the change to the other side.

CAREFUL! Please be careful when using this synchronizer, especially when you add multiple issues to the Structure, as this may lead to massive updates in the Agile ranks without undo.

On Fix Versions

Earlier GreenHopper versions relied on values in the **Fix in Version/s** field - if a version has been released, the issues assigned to that version won't appear on the Classic GreenHopper boards. GreenHopper synchronizer in Structure reflected that behavior and ignored such issues.

With the introduction of new Boards (known initially as Rapid Boards, then as Agile Boards), this dependency on Fix Version field has become optional. In some cases, Fix Version field is completely disabled and the teams use Agile Sprints. To address that, the JIRA Agile synchronizer no longer filters issues by Fix Version, unless you're using an old GreenHopper version.

JIRA Agile Synchronizer Rules

Common Rules:

- Issues that do not belong to the synchronized project(s) are not affected. If you've got GreenHopper earlier than 5.8 and not using Global Rank field, then issues that are assigned to Fix Versions that have been released are also not affected.
- This synchronizer does not add issues to the structure (with two exceptions, explained below). You can use Saved Filter synchronizer together with JIRA Agile synchronizer to automatically add and position issues.

Sub-Tasks Synchronization:

- With **Auto-Add Subtasks** mode on, sub-tasks are added to the structure if their parent is there in the structure.
- The sub-tasks are forced to stay under their parent, so if you move a subtask somewhere else, it will jump back under the parent again. You can rearrange the order of the sub-tasks, which will be sync'ed to the Agile Rank if the Rank Field is configured.

Rank Synchronization:

- Repositioning issues in the structure causes Rank change and the repositioning issues on the Planning Board.
- Rearranging issues on the GreenHopper's Planning Board causes the issues to be rearranged in the structure.
- When issues are repositioned in the structure according to Rank, they are never moved under a different parent issue.

> ⚠ This restricts the possible rank changes in JIRA Agile - you can only move an issue to the position of another issue that is under the same parent issue in the structure, otherwise the issue will "jump back" later.

Epic Synchronization:

- Placing an issue under an Epic in the structure will cause its Epic field to change to that Epic.
  - It does not matter at what level of depth is the sub-issue. A sub-sub-sub-issue of an Epic issue will also have its Epic field updated.
- If you move an issue in the structure so that it's not under any epic, its Epic field will be cleared.
- If you manually change Epic field (using JIRA Agile UI or otherwise) to point to a different Epic, the issue will be repositioned under that Epic in the structure.
  - An issue that has the Epic field pointing to an Epic in the structure will be automatically added to the structure.
- If you clear Epic field or change it to point to an epic that is not in the structure, the issue will be moved up in the structure until it is no longer under any epic.

How to Add Issues to Structure Sync'ed with JIRA Agile

When JIRA Agile synchronizer is enabled, it automatically updates Agile order in background when any Structure change happens. So if you carelessly add issues from the sync'ed project to the structure in some random order, their ranks will be updated according to that order.

To add issues to the structure without breaking the existing backlog order:

- If adding manually on the Structure Widget, use JQL search and add *order by Rank* clause at the end of the query. Use the rank field that is used by the synchronizer.
- Select the position of the added issues carefully (best with drag-and-drop or copy/paste) - the order is likely to change unless you place issues under another issue without any other sub-issues (see *Syncing Partial Orders* below).
- If using Saved Filter synchronizer to add issues, add *order by Rank* clause to the Saved Filter's query. However, the new issues that are added with the Saved Filter synchronizer will appear at the end of the structure and so will have the latemost ranking.

Syncing Partial Orders

JIRA Agile's Board is flat (except for sub-tasks), and the Structure is hierarchical - so it is not possible to precisely rearrange Structure to have all issues come in the same order as they do on the Planning Board, without changing issue parents or making the Structure also flat.

Henceforth, the Structure syncs subsets of the issues in the hierarchy with Agile Rank. For example, consider the following Structure:

A	
	B
	C
D	
	E
	F

It is not possible to rearrange the sub-issues so that they come in the following order: B, E, C, F - although this is possible on the Planning Board. Instead, the structure will synchronize sub-sets of the issues in the Structure with JIRA Agile. The following sub-sets will be synchronized separately:

- A, D - top-level issues: A must come before D on the Planning Board
- B, C - sub-issues of A are sync'ed separately, so B must come before C on the Planning Board
- E, F - ditto for the sub-issues of D

In JIRA Agile version 6.1 and later, the Epics are treated by JIRA Agile as a separate set of issues, different from Stories and other non-Epics. To accommodate this change, Structure updates the rank of issues also using "partial order" approach, syncing Epics and non-Epics separately. This means that, starting with JIRA Agile 6.1, if an Epic comes before a Story on the Structure Board, it is not required that they come in the same order on the Scrum Board.

Status Rollup Synchronizer

Status Rollup synchronizer automatically aggregates statuses of the sub-issues and updates the status of the parent issue. For example, it can make parent issue *Resolved* if all sub-issues are *Resolved*.

Status Rollup Synchronizer Parameters

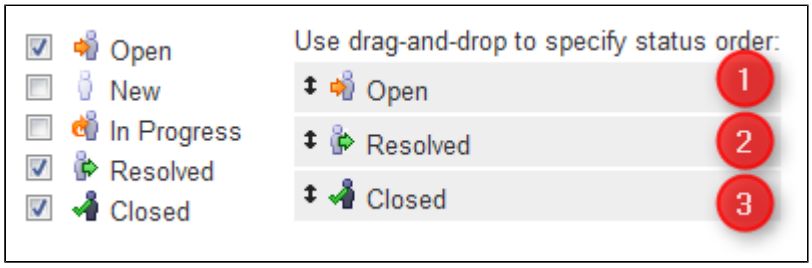
Enabled Projects	Only issues belonging to the selected projects are **changed**. It does not matter what project sub-issues belong to, as long as their parent belongs to the enabled project — every sub-issue counts with its status.
Enabled Issue Types	Same with types — you can select issues of which types may be changed by the synchronizer, and like with the enabled projects, only the parent issue type is checked.
Statuses Rolled Up	The selection and order of statuses that are used to calculate parent issue status. Parent issue status is set to the *earliest* status among its sub-issues. If a sub-issue has a status not selected in this parameter, the parent issue is not changed.
Allowed Transitions	For every status, you can select which transitions the synchronizer can make to move an issue to that status.
Resolution	Value to set to the *Resolution* field when workflow transition requires it. By default, a current or default value for Resolution is used.

The synchronizer is normally installed, resynced and used in the Incremental mode, tracking changes to issues and structure and updating issues. The synchronizer supports Exporting from Structure, changing statuses of the issues in the structure on one-time basis.

How Status Rollup Synchronizer Works

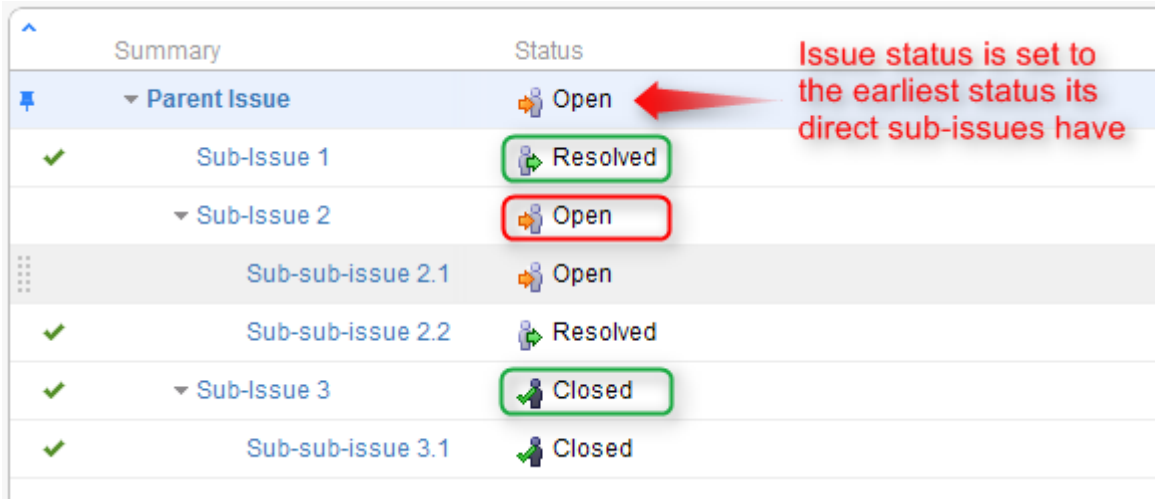
The synchronizer tracks updates to issues and to structure, and tries to make sure that the status of the parent issue corresponds to the aggregate status of its direct children.

When you configure Status Rollup, the most important parameter is the selected Statuses and their order:



Statuses that are not selected in the parameters are not recognized by the synchronizer. If a sub-issue has one of the unselected statuses, the synchronizer does not change the parent issue.

The order of the selected statuses should correspond to *earliest-to-latest* order of phases of the workflow. For example, the screenshot above shows configuration where *Open* is followed by *Resolved*, which is followed by *Closed*. With that configuration, once all sub-issues of an issue are *Resolved*, the synchronizer will try to make the issue *Resolved* too. Once all sub-issues are *Closed*, the issue will be made *Closed*. But if at least one sub-issue happens to be *Open*, the issue status will be set to *Open* — because it is the earliest status in the specified order.



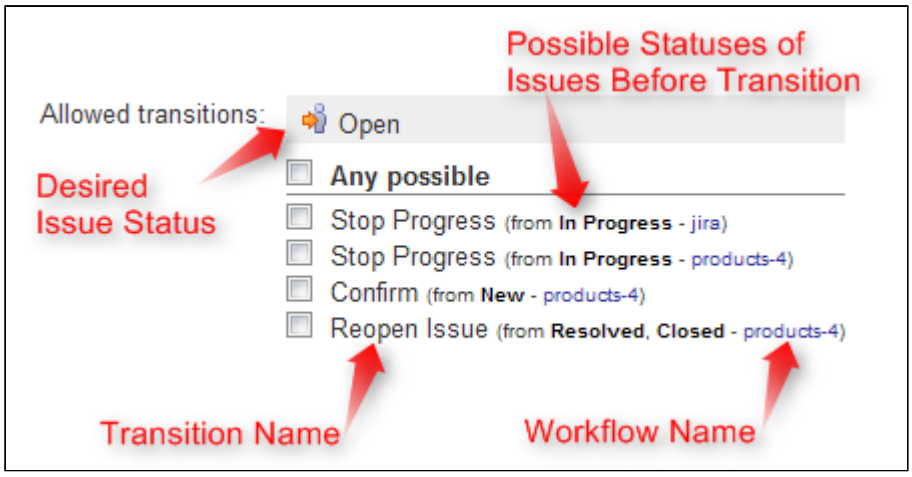
On the screenshot above:

- All **sub-sub-issues** and **sub-issue 1** do not have sub-issues of their own, so the synchronizer does not change their status.
- **Sub-issue 3** has a single sub-issue, which has status **Closed** — so since all of its sub-issues are closed, it should be **Closed** too.
- **Sub-issue 2** has one **Open** sub-issue and one **Resolved** sub-issue — it should be **Open** because Open status comes before Resolved in the order specified earlier.
- **Parent Issue** has sub-issues that have statuses **Open**, **Resolved** and **Closed** — so it should be **Open** for the same reason. Once all sub-issues are **Resolved**, Parent Issue will be automatically **Resolved**. Once all sub-issues are **Closed**, Parent Issue will automatically be **Closed**.

Remember, that whenever one of the sub-issues gets a status not listed in the synchronizer configuration, the synchronizer just skips the issue. For example, if we change the status of **Sub-issue 2** above to **In Progress**, **Parent Issue** will not be updated. If we then change the status of **Sub-issue 2** to **Resolved**, **Parent Issue** status will be updated to **Resolved**.

How Status is Changed

JIRA allows status to be changed only through a workflow transition, so the only way Status Rollup synchronizer can set the desired status on an issue is to apply a workflow transition. Therefore, when you select a status, you also need to select which transitions is synchronizer allowed to make.



So what the synchronizer does is:

1. See what status the issue currently has;
2. Calculate what status it should have, based on the statuses of sub-issues;
3. Find workflow transitions that can transfer the issue from the current status to the required status;
4. Check which of those transitions are allowed by the configuration;
5. Try to apply matching transition number one, if it fails — try the next one, and so on.

Note that all transitions are done under the account of the user who has installed the synchronizer.

Why Can a Workflow Transition Fail

It's not guaranteed that the synchronizer will be able to change the Status, because workflows are too flexible and there are many reasons that a given transition, which you have allowed in the configuration, can fail to execute. Here are some of the possible causes:

- You (the user who has installed the synchronizer) do not have the required permissions to make the transition;
- You are not the Assignee of the issue — required for In Progress status;
- Some other pre-condition defined in the workflow fails;
- Workflow transition requires a field to be set on an issue that has no default value.

As described above, it's possible that there are several possible transitions from one status to another. The synchronizer will try all of them unless one of them succeeds.

If the synchronizer fails to update the status, a warning message will be written into the server logs (subject to logging configuration).

Changing Resolution

You can set up a specific *Resolution* value to be set whenever a transition involves changing the resolution. If you don't specify this parameter, the default resolution or already existing resolution will be used.

In order to tell which issues have been automatically moved to a status like Resolved or Closed, you can set up a special resolution like *Auto-Resolved*.

Manually Changing Status of an Issue That Has Sub-Issues

Even if an issue has sub-issues and is subject to Status Rollup, you can manually change its status. Although the synchronizer will **not** be forced to recalculate the status of that issue immediately, it will recalculate the status if any of the sub-issues change – probably reversing your change, if it finds an allowed transition.

If you'd like the synchronizer to only move issues *forward*, that is, from *Open* to *Resolved*, but not vice versa, you can configure the allowed transitions accordingly.

2.10 Structure Activity Stream

JIRA's **Activity Stream** dashboard gadget lets you see recent activity in JIRA and other connected systems. The activity stream can be filtered (for example, by project) to show you only the changes that concern you or your team. In addition, **Activity** tab on the issue page displays recent activity that has affected the viewed issue.

With the Structure plugin installed, Activity Stream gadget may be configured to include changes made to structures. The activity stream on the issue page automatically includes all changes to all structures that affect the position of the viewed issue.

To activate the Structure stream, select the Structure option in the Available Streams section of the Activity Stream gadget configuration.

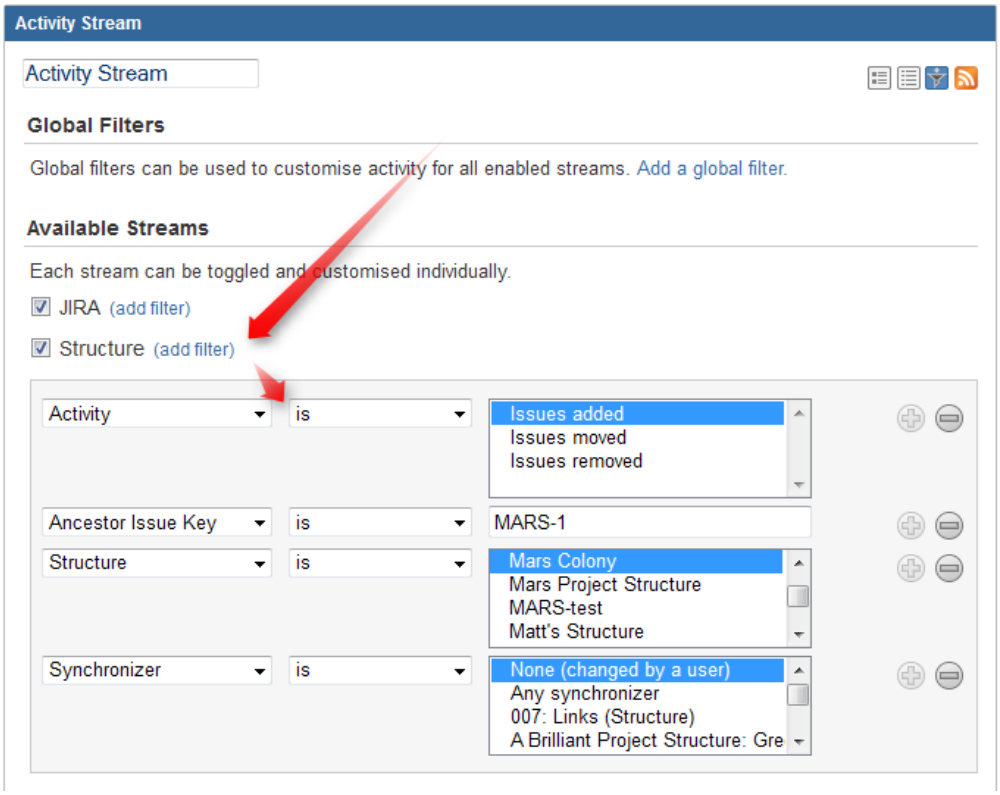
2.10.1 Available Filters

The following filters are available for the Structure activity stream:

- **Structure**
  Use it to see changes only in a specific structure or structures, or to exclude specific structures from the stream. If this filter is not used, changes to all structures are shown.
- **Ancestor Issue Key**
  This filter can be used together with the **Structure** filter if you are interested in changes within a specific part of a specific structure, located "under" the specified issue (if the changed issue is not located under the specified issue, the change will not be shown). You can enter several issue keys separated by spaces.
- **Synchronizer**
  You can include or exclude changes made by a synchronizer (either by any synchronizer or by specific synchronizers). Since synchronizers might make a lot of changes, this might be useful to filter out their "noise". Vice versa, you could verify that a synchronizer works as expected with an activity stream and this filter.

- **Activity**
  All changes to a structure fall into three categories: adding issues to structure, removing issues from structure and moving issues within structure. This filter lets you include or exclude the particular types of changes.

All Global Filters are supported by Structure Stream as well – you can filter structure changes by **Project, Issue Key, Update Date** and **Username**.



2.10.2 Reading Activity Stream

Changes in the Structure activity stream are ordered chronologically, newest first. For each change a short summary is displayed, containing:

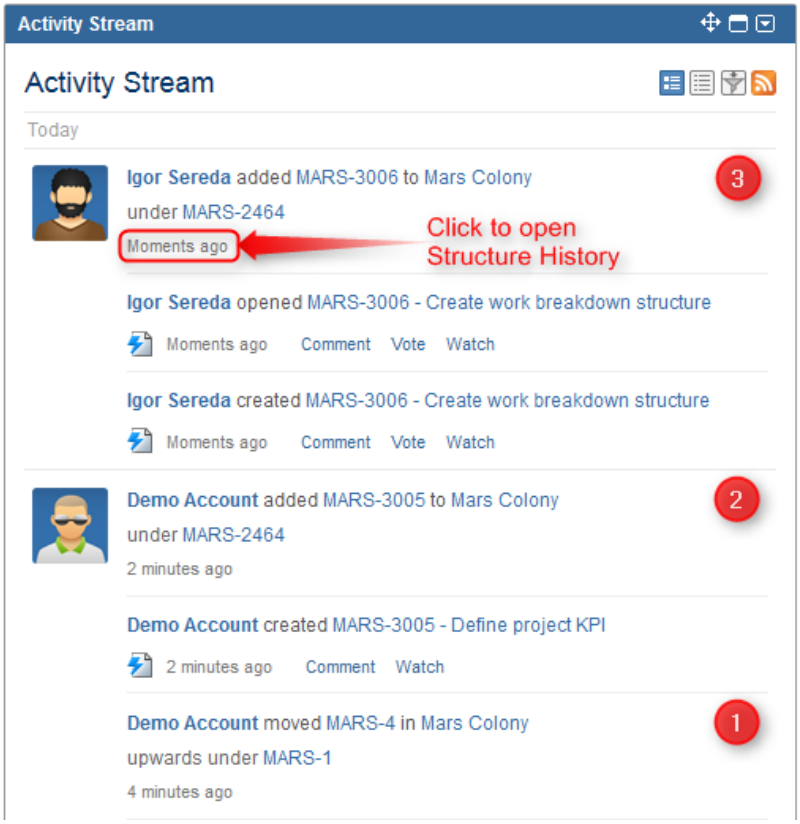
- the full name of the user who made the change;
- for changes made by a synchronizer, the name of the synchronizer;
- the number of affected issues, and whether they were they added, removed or moved;
- if **Project** filter is used, the number of affected issues in each of the selected projects;
- if **Issue Key** filter is used, the affected issues among those selected in the filter;
- the name of the changed structure.

When viewing activity stream in the Full View, the following is also shown:

- the parent path of the affected issues;
- the original and the new parent path for the moved issues;

- if the issues were moved within the same parent issue, the direction of the move (upwards / downwards);
- when the change was made.

Parent Path is a sequence of issue keys: first, a top-level issue, then its sub-issue, then sub-sub-issue, and so on until the parent of the affected issue is displayed. Hover mouse over an issue key to view the issue's summary, or click it to go to that issue.



On this screenshot, items 1, 2 and 3 are Structure activities.

In the Full View, click on the time of the change to open that change on the Structure Board in the History View.

2.10.3 Activity Streams Performance

Structure's activity stream is optimized to quickly provide data for the most common activity requests from Dashboard, Issue Activity, User Activity and Project Activity page.

It is possible however, if you use a complex search query on a JIRA instance with large history of structure changes, that querying database will take longer time than Activity Streams allows and you will not see any results. (There should be a message that "one of the activity streams providers took long time to provide an answer".)

If that is the case, try to reduce the amount of conditions you are using or contact support for help.

2.11 Structured JQL

Structure adds `structure()` JQL function that lets you search for issues that are added to a structure, with the possibility to add constraints on their relationships in the hierarchy. The language used to define the constraints is called **Structured JQL** or **S-JQL**.

To quickly find a solution to a common structure querying problem, consult S-JQL Cookbook, which contains a number of examples. To build your own query, start off with the closest example and modify the query as needed. Consult S-JQL Reference for a comprehensive description of the language and `structure()` JQL function.

- S-JQL Cookbook
- S-JQL Reference

2.11.1 S-JQL Cookbook

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

- Find issues added to a structure
- Quick Filter for JIRA Agile's (GreenHopper) Scrum Board to display only low-level issues in a structure
- Retrieve all Epics in a certain status and all of their children
- Find Test Cases associated with Stories in an active sprint
- Find all issues that are blocking critical issues
- Find all unassigned issues in a part of a project
- Top-level view on unfinished parts of a project
- Find violations of the rule "Tasks must be under Epics or Stories"
- Find violations of the rule "An issue cannot be resolved if it has unresolved children"
- Find issues that can be resolved because all their children are resolved
- Get a view of a second (third, ...) level of the hierarchy

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

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

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

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

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

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

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

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

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

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

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

2.11.2 S-JQL Reference

structure() JQL Function Reference

To specify a structure condition in JQL, use the following format:

issue in structure(*structureName*$_{opt}$, *structureQuery*$_{opt}$**)**

Function arguments:

structureName	*Optional*	The name of the structure. If you omit the structure name, system-wide Default Structure will be searched. Remember to enclose the name in double quotes ("") if it contains spaces or non-letters.
structureQuery	*Optional*	Use this parameter to select only a part of the structure. This parameter specifies a *Structure Query* in a language similar to JQL, *Structured JQL*, which is discussed below.

You can use structure ID instead of the structure name. You can see structure ID in the URL of the Structure Board if you open **Manage Structure** page and click structure name.

If a user does not have access to structure, they will not be able to create new queries with the `structure()` function and existing queries will have `structure()` function return an empty set. However, the user will still see `structure()` function offered in the JQL completion drop-down.

Structured JQL Language Reference

Structure query is a hierarchical condition on the issues added to the structure. Structure query is expressed in the Structured JQL language (S-JQL), described in this section.

This reference assumes that you are familiar with Advanced Searching and Advanced Searching Functions capabilities of JIRA.

List of Structured JQL topics:

Constraints

Structure query consists of *constraints*. Each constraint matches some issues in the structure. In the simplest case, the whole structure query consists of a single constraint; for now, we will consider only this case.

There are two types of constraints: *basic* and *relational* constraints.

^ up to the list of S-JQL topics

Basic constraint

A basic constraint is one of the following:

1. A JQL query enclosed in square brackets:

```
[status = Open]
```

This kind of basic contraint matches all issues in the structure that satisfy the JQL query.

2. Issues having special positions within the structure:

```
leaf
```

or

```
root
```

The first constraint matches issues at the bottom level of the hierarchy, i.e., issues that do not have children.
The second constraint matches issues at the top level of the hierarchy, i.e., issues that do not have a parent.

3. A comma-separated list of issues:

```
TS-129, TS-239
```

or just a single issue:

```
TS-129
```

You can specify issue key, as above, or issue ID:

```
19320
```

This kind of basic constraint matches just the referenced issues. If some of the issues are not contained within the structure, they are ignored. If none of the issues are contained within the structure, the constraint matches no issues.

4. An empty constraint matching no issues:

```
empty
```

This constraint plays the same role as JQL's `EMPTY` keyword. It is intended to be used as a sub-constraint in relational constraints, which are discussed further.

^ up to the list of S-JQL topics

Negation

Any constraint, basic or relational, can be negated using keyword `NOT`. This produces a constraint that matches all issues that the original constraint doesn't:

```
```

```
not root
```

matches all issues that are not top-level issues in the structure.

You can always enclose a constraint in parentheses to ease understanding. So, all issues in the structure except `TS-129` and `TS-239` are matched by this structure query:

```
not (TS-129, TS-239)
```

^ up to the list of S-JQL topics

Relational constraint

A basic constraint matches issues that satisfy a condition. A relational constraint matches issues *related to* issues that satisfy a condition. *Related* corresponds to a relationship between issues in the structure, like parent-child.
For example,

```
TS-129
```

is a basic constraint that matches a single issue `TS-129`;

```
child in TS-129
```

is a relational constraint matching an issue such that its child is `TS-129`.

Relational constraint has the form `relation operator subConstraint`. Here, `subConstraint` is a constraint on the relatives of issues to be matched; other parts of relational constraint are discussed in the following sections.

Note that the form of relational constraint is similar to the form of JQL clause, `field operator value`.
Indeed, let's describe in English a JQL query `type in (Epic, Story)`: it matches issues having *type* that is *in* values *Epic, Story*.
Now, let's describe in English a structure query `parent in [type = Epic]`: it matches issues having *parent* that is *in* constraint "`type = Epic`".
As you can see, the form that can be used to describe the structure query is similar to that of JQL.

^ up to the list of S-JQL topics

Relations

S-JQL has the following relations:

- `child`: issue is a child (sub-issue) of another issue in the structure.
- `parent`: issue is a parent of another issue in the structure.
- `descendant`: issue is a descendant (sub- or sub-sub-...-issue) of another issue in the structure.
- `ancestor`: issue is an ancestor (parent, parent-of-parent, or parent-of-parent-...-of-parent) of another issue in the structure.
- `sibling`: issue is a sibling of another issue in the structure. Two issues are considered siblings if they are under the same parent issue.
- `prevSibling`: issue is a previous (preceding) sibling of another issue in the structure.
  Issue *A* is a preceding sibling of issue *B* if it is a sibling of *B* and *A* is higher than *B* (*A* comes before *B* .)
- `nextSibling`: issue is a next (following) sibling of another issue in the structure.
  Issue *A* is a following sibling of issue *B* if it is a sibling of *B* and *A* is lower than *B* (*A* comes after *B*.)
- `issue` is a relation of an issue to itself. Its role is explained later, in the `issue` relation section, because first one has to grok how operators and sub-constraints work.
  There are also combinations of `issue` with all other relations, listed for completeness below.
- `childOrIssue`
- `parentOrIssue`
- `descendantOrIssue`
- `ancestorOrIssue`
- `siblingOrIssue`
- `prevSiblingOrIssue`
- `nextSiblingOrIssue`

Those familiar with XPath may have recognized these relations; indeed, they work like the corresponding XPath axes.

^ up to the list of S-JQL topics

Operators

These are the operators used in S-JQL:

```
IN, NOT IN, IS, IS NOT, =, !=, OF
```

`operator` specifies how `subConstraint` is applied to `relation`:

1. `IN`, `IS`, and = put constraint on the relatives of a matched issue.

   For example, consider

   ```
   child in (TS-129, TS-239)
   ```

Here, `relation` is `child`, so an issue's relative in question is its child in the structure. Thus, an issue matches if *at least one of its children is `TS-129` or `TS-239`*.

There is no difference between these three operators, unlike JQL. Different forms exist to allow for more natural-looking queries with some sub-constraints.

2. `NOT IN`, `IS NOT`, and `!=` are negated versions of `IN`, `IS`, and =. That is, an issue is matched if it *is not related to* any issue matching `subConstraint`.

As an important consequence, issue that has no relatives is matched.

For example, consider

```
child not in (TS-129, TS-239)
```

An issue matches if *no child is `TS-129` nor `TS-239`*; thus, this constraint matches all issues that either have no children or do not have any of these two issues among their children.

Using one of these operators in a relational constraint is the same as using `IN` (or `IS`, or =) and negating the whole relational constraint. Thus, the constraint above is equivalent to

```
not (child in (TS-129, TS-239))
```

But, using one of these operators is **very not** the same as using operator `IN` and negating `subConstraint`!

First, *having relatives other than X* is not the same as *not having relatives X*. Think of it as of relationships in a human family: having a relative other than brother (e.g., a sister) is **not** the same as not having a brother, because one may have both a sister and a brother.
Second, an issue with no relatives is not matched by the transformed query.

For example,

```
child in (not (TS-129, TS-239))
```

matches all issues that have at least one child that is neither `TS-129` nor `TS-239`. That is, the only issues that are not matched are leaves and those that have only `TS-129` or `TS-239` as children.

3. `OF` matches the relatives of issues that satisfy `subConstraint`.

   For example, consider

   ```
   child of (TS-129, TS-239)
   ```

   An issue matches if *it is a child of either `TS-129` or `TS-239`*.

To have a model of how operators `IN` (`IS`, `=`) and `OF` work and to understand the difference between them, consider the table below. Suppose that we take all issues in the structure and put each of them, one by one, in column **issue**. For each issue, we take all of its relatives and put each of them, one by one, in column **relative**. Thus we get pairs of issues. We examine each pair, and if one of the components satisfies *subConstraint*, we add the other component to the result set. Which component is added, depends on the operator:

operator	issue	relative
`in`	*add to result set*	*satisfies* `subConstraint`
`of`	*satisfies* `subConstraint`	*add to result set*

One may note that for any relation, there is a corresponding "inverse": for example, `child` is the inverse of `parent`, and vice versa. A relational constraint that uses operator `IN` (`IS`, `=`) is equivalent to a relational constraint that uses an inverse relation with operator `OF`. That is,

```
child in (TS-129, TS-239)
```

is the same as

```
parent of (TS-129, TS-239)
```

Again, different forms of expressing the same constraint exist to allow for more natural-looking queries.

^ up to the list of S-JQL topics

Sub-constraints

Any constraint can be used as a sub-constraint, whether basic, relational, or a combination of those.
For example,

```
child of root
```

selects issues on the second level of the hierarchy. To select issues on the third level of the hierarchy, you can once again use relation `child` and the previous query as `subConstraint`:

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

There is a special basic constraint, `empty`, which matches no issues. It is used as a sub-constraint to match issues that have no relatives as per `relation`.

For example, let's take relation `child` and see what the corresponding relational constraints with different operators mean.

`child is empty`	matches all issues that have no children (equivalent of `leaf`)
`child is not empty`	matches all issues that have at least one child (equivalent of `not leaf`)
`child of empty`	matches all issues that are not children of other issues (equivalent of `root`)

Of course, using `leaf` or `root` is more convenient, but you can apply `empty` to any other relation. For instance, `sibling is empty` matches an issue if it is the only child of its parent.

^ up to the list of S-JQL topics

issue relation: adding sub-constraint matches to the result set

A relational constraint with relation `issue` behaves exactly as its sub-constraint, possibly negated if operator `NOT IN` (`IS NOT`, `!=`) is used.
Thus,

```
issue in [status = Open]
```

is equivalent to

```
[status = Open]
```

Similarly,

```
issue not in [status = Open]
```

is equivalent to

```
not [status = Open]
```

When combined with another relation, `issue` allows to add the issues matched by `subConstraint` to the resulting set. For example,

```
descendant of TS-129
```

returns all of the children of `TS-129` at all levels, but does not return `TS-129` itself. To add `TS-129`, use `descendantOrIssue`:

```
descendantOrIssue of TS-129
```

^ up to the list of S-JQL topics

Combining constraints with Boolean operators

We can now define a structure query as a *Boolean combination of constraints*, that is, a structure query consists of constraints connected with `AND` and `OR`. When two constraints are connected with `AND`, together they will match issues that are matched by both constraints. This allows you to limit the results. Likewise, when two constraints are connected by `OR`, together they will match issues that are matched by at least one of the constraints. This allows you to expand the results.

Note that `AND` has higher precedence than `OR`. That means that the Structure query

```
leaf or (parent of leaf) and [status = Open]
```

matches all issues that are either leaves, or are parents of leaves in status *Open*. In order to also constrain leaf issues to be in the status *Open*, you need to use parentheses:

```
(leaf or (parent of leaf)) and [status = Open]
```

^ up to the list of S-JQL topics

Quoting structure query argument in the structure() JQL function

When specifying structure query as a parameter of the `structure()` JQL function, you should enclose it in "double quotes" or 'single quotes' if it contains spaces or non-letters. Please note that if you are using quotes of one kind, you cannot use quotes of the same kind in the inner JQL constraint.

This query will not parse:

```
issue in structure("My personal structure", "child of [Status = "Awaiting
Deployment"]")
```

You should use single quotes in the inner JQL constraint instead:

```
issue in structure("My personal structure", "child of [Status = 'Awaiting Deployment']")
```

If some values in the inner JQL constraint contain quotes, you should escape them with a backslash:

```
issue in structure("My personal structure", "child of [fixVersion = 'funky\"Version']")
```

^ up to the list of S-JQL topics

Backward compatibility with structure() JQL function prior to Structure 2.4

Prior to Structure 2.4, `structure()` JQL function did not take structure query as an argument; you could specify only one issue key or ID, and you would get the referenced issue along with all of its children at all levels. As you might have noticed, this old-style usage can be interpreted as a structure query, but according to the rules of S-JQL, it would return just the referenced issue without its children. To maintain backward compatibility, any structure query in Structure 2.4 that consists of a single basic constraint that references issues by their keys or IDs matches not only these issues, but all of their children as well.

That means that if you were using JQL of the form

```
issue in structure("My personal structure", TS-129)
```

then in Structure 2.4 this query will still return `TS-129` and all of its children at all levels (provided that `TS-129` is added to the structure.)

If this backward compatibility bites you (if, say, you need to check whether an issue is added to a structure), prepend the structure query with `issue in`:

```
issue in structure("My personal structure", "issue in TS-129")
```

This JQL will match only `TS-129` if it is in the structure.

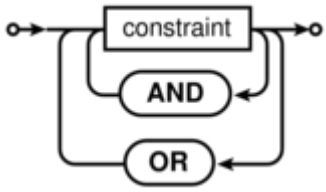
^ up to the list of S-JQL topics

Railroad diagrams

As a final piece of reference, here's the S-JQL syntax in the form of railroad diagrams.

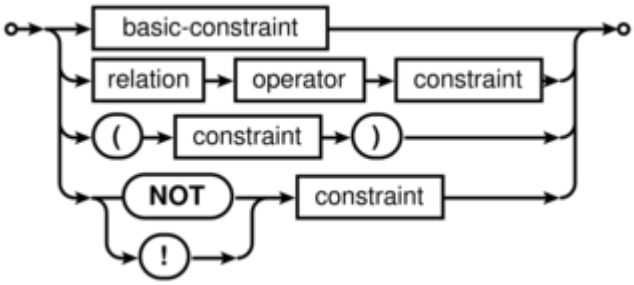
S-JQL keywords are case-insensitive, and all underscores in keywords are optional.

structure-query

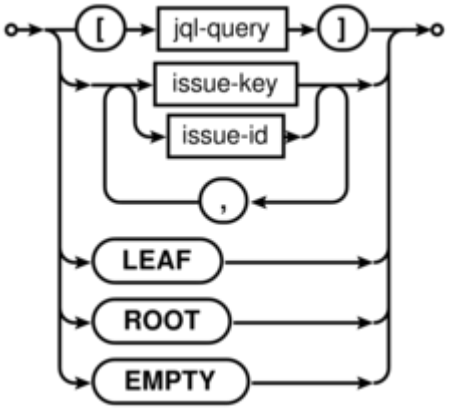


S-JQL admits using `&&` and `&` in place of `AND`, as well as `||` and `|` in place of `OR`.

constraint

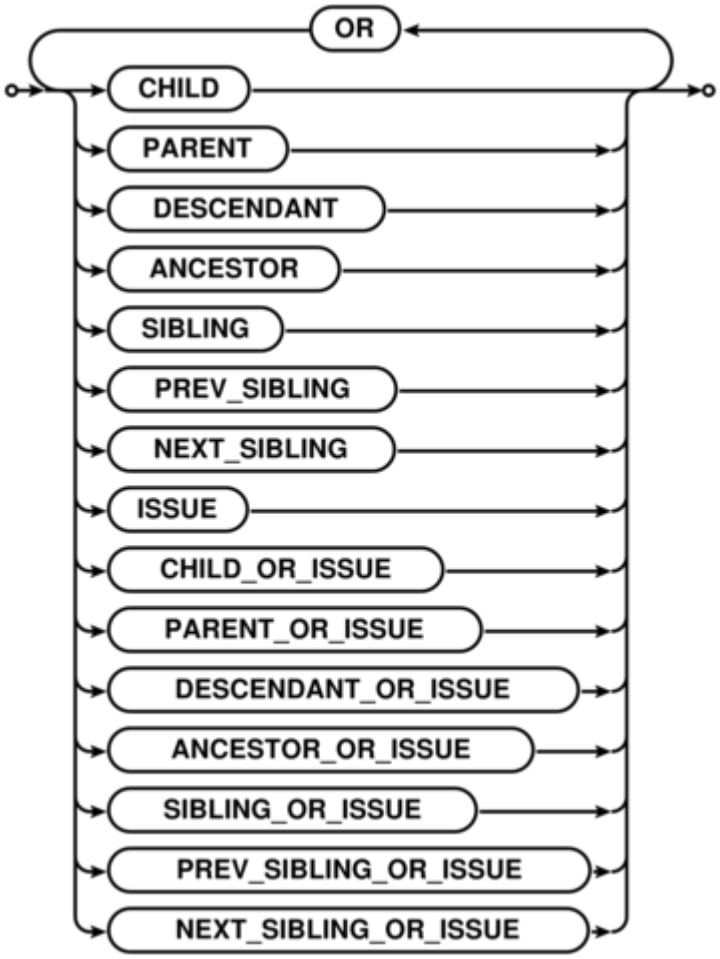


basic-constraint



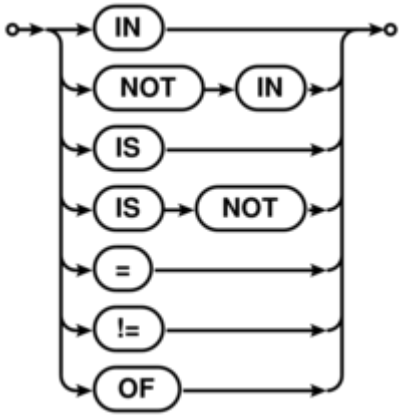
- `jql-query` is any valid JQL query subject to the quoting restrictions.
- `issue-key` is any valid JIRA issue key.
- `issue-id` is any valid JIRA issue ID.

relation



S-JQL admits using `||` and `|` in place of `OR`.

operator



^ up to the list of S-JQL topics

2.12 Keyboard Shortcuts

Structure provides a number of keyboard shortcuts that you can use to speed up your work. These reference cards describe the shortcuts for Mac OS X and PC keyboards.

- Keyboard Shortcuts (PC)
- Keyboard Shortcuts (Mac)

2.12.1 Keyboard Shortcuts (PC)

Navigation

Action	Shortcut
Select Issue	*Left-Click*
Previous Issue	*or* `k`
Next Issue	*or* `j`
Expand Sub-Issues	
Collapse Sub-Issues	
For Large Structure	`PgUp PgDn Home End`
Add Column	`tt`
Expand All	`++`
Collapse All	`--`

Structure Views

Action	Shortcut
Switch View	`vv`
Save View	`vs`
Save View As	`vss`
Revert Changes to View	`vr`

Searching & Adding to Structure

Action	Shortcut
Search	`Alt+/`
JQL On/Off	`Alt+j`
Filter On/Off	`Alt+f`
Add More On/Off	`Alt+m`
Next Matching	`]`
Previous Matching	`[`
Add Issue	`Ctrl+Enter`
Add as Sub-Issue	`Ctrl+Shift+Enter`

Standard JIRA Actions

Action	Shortcut
Operations Dialog	`.`
View Selected Issue	`o`
Edit Issue	`e`
Assign Issue	`a`
Comment on Issue	`m`
Edit Issue Labels	`l`
Actions Drop-Down	`Alt+`

Changing Structure

Action	Shortcut
Move Up	`Ctrl+`
Move Down	`Ctrl+`
Indent	`Ctrl+`
Outdent	`Ctrl+`

Action	Shortcut
Drag and Drop	`Shift+Drag`
New Issue	`Enter`
New Sub-Issue	`Insert` *or* `Shift+Enter`
Remove from Structure	`Delete`

Changing Issues

Action	Shortcut
Edit Field	*Double-Click*
Edit Summary	`Tab` *or* `F2` *or* `ss`
Finish & Save	`Enter` *or* `Ctrl+Enter`
Cancel Field Changes	`Esc`
Edit Next Field	`Tab` *or* `Ctrl+Alt+`
Edit Previous Field	`Shift+Tab` *or* `Ctrl+Alt+`
Edit Next Issue	`Ctrl+Alt+`
Edit Previous Issue	`Ctrl+Alt+`

Selecting Issues

Action	Shortcut
Toggle Selection	`Space`
Select All	`Ctrl+a`
Select All Sub-Issues	`Shift+`
Deselect All Sub-Issues	`Shift+`
Expand Selection Down (Up)	`Shift+` (`Shift+`)

Bulk Selection	`Shift+PgUp`
	`Shift+PgDn`
	`Shift+Home`
	`Shift+End`

Clear Selection	`Escape`

Advanced

Action	Shortcut
Hide/Show Resolved	`rr`
Cut (Prepare to Move)	`Ctrl+x`
Paste (Move)	`Ctrl+v`
Paste Sub-Issue (Move)	`Ctrl+Shift+v`
Fix/Unfix View on Issue	`Ctrl+.`
Switch Panel	`\`
Show Extra Actions on the Toolbar	`xx`
View Full-Size Image	`ii`

2.12.2 Keyboard Shortcuts (Mac)

Navigation

Action	Shortcut
Select Issue	*Left-Click*
Previous Issue	*or* `k`
Next Issue	*or* `j`
Expand Sub-Issues	
Collapse Sub-Issues	
For Large Structure	
Add Column	`tt`
Expand All	`++`
Collapse All	–

Structure Views

Action	Shortcut
Switch View	`vv`
Save View	`vs`

Action	Shortcut
Save View As	`vss`
Revert Changes to View	`vr`

Searching & Adding to Structure

Action	Shortcut
Search	`/`
JQL On/Off	`j`
Filter On/Off	`f`
Add More On/Off	`m`
Next Matching	`]`
Previous Matching	`[`
Add Issue	
Add as Sub-Issue	

Standard JIRA Actions

Action	Shortcut
Operations Dialog	.
View Selected Issue	o
Edit Issue	e
Assign Issue	a
Comment on Issue	m
Edit Issue Labels	l
Actions Drop-Down	

Changing Structure

Action	Shortcut
Move Up	
Move Down	

Action	Shortcut
Indent	
Outdent	
Drag and Drop	Drag
New Issue	
New Sub-Issue	
Remove from Structure	

Changing Issues

Action	Shortcut
Edit Field	*Double-Click*
Edit Summary	`tab` *or* `ss`
Finish & Save	*or*
Cancel Field Changes	`esc`
Edit Next Field	`tab` *or*
Edit Previous Field	`tab` *or*
Edit Next Issue	
Edit Previous Issue	

Selecting Issues

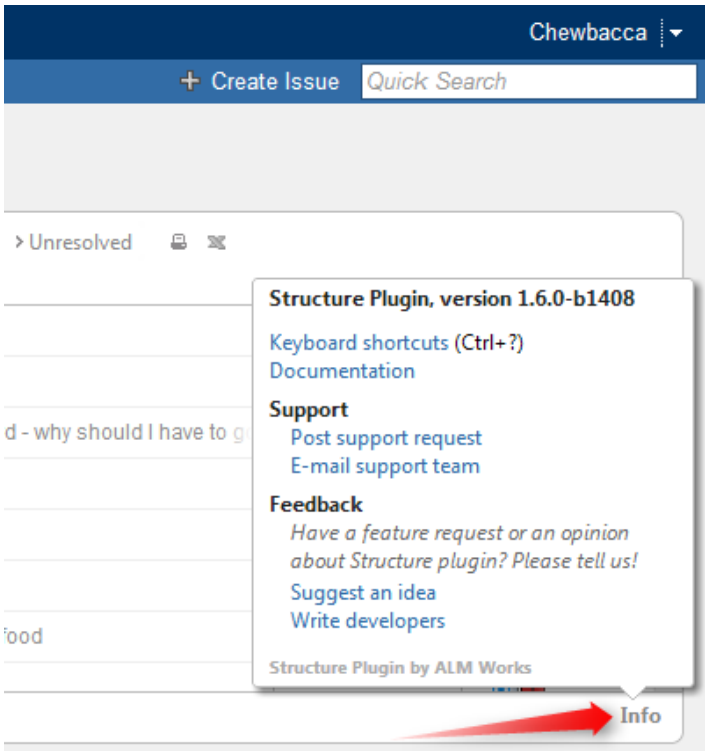
Action	Shortcut
Toggle Selection	`space`
Select All	a
Select All Sub-Issues	
Deselect All Sub-Issues	
Expand Selection Down (Up)	()
Bulk Selection	
Cancel Selection	`esc`

Advanced

Action	Shortcut
Hide/Show Resolved	`rr`
Cut (Prepare to Move)	`x`
Paste (Move)	`v`
Paste Sub-Issue (Move)	`v`
Fix/Unfix View on Issue	`.`
Switch Panel	`\`
Show Extra Actions on the Toolbar	`xx`
View Full-Size Image	`ii`

2.13 Getting Help

Click **Info** link at the bottom right corner of the Structure Widget to bring up Structure Information panel. It contains information about Structure version and useful links.



Feel free to write back to ALM Works if you have any questions, feature requests or problems:

- Post support request and have us resolve it as soon as possible.
- Suggest an idea on our UserVoice forum.
- Write to developers just to say hi or with any comments or questions.

3 Structure Administrator's Guide

This section contains information for JIRA administrators about installing and configuring Structure plugin.

Quick steps to get Structure working:

1. Installing Structure
2. Setting Up Structure License
3. Selecting Structure-Enabled Projects

Contents:

3.1 Installing Structure

Structure is installed like the most other plugins.

1. Before installing Structure in production, make sure your JIRA meets Memory Guidelines.
2. Open Plugin Manager, search for "Structure" by ALM Works on the Atlassian Marketplace and install from there.

   Alternatively, you can download the plugin JAR manually from download page and either place it into *plugins/installed-plugins* subdirectory under your JIRA home (then restart JIRA) or use "Upload Plugin" link in the Plugin Manager.

3. Press **Get Started** button to finish the installation by installing a license key and selecting Structure-enabled projects.

Congratulations! You can now spread the word and help users get started with Structure – consider sharing a link to Structure 101 page.

If Structure Plugin Remains Disabled

It is possible that after you install Structure or enable it from the Plugin Manager, the plugin remains disabled. An error may or may not be shown. If you refresh Plugin Manager page within 5-10 seconds and Structure is disabled, you've got this problem.

See Structure plugin won't start article for possible causes and solutions.

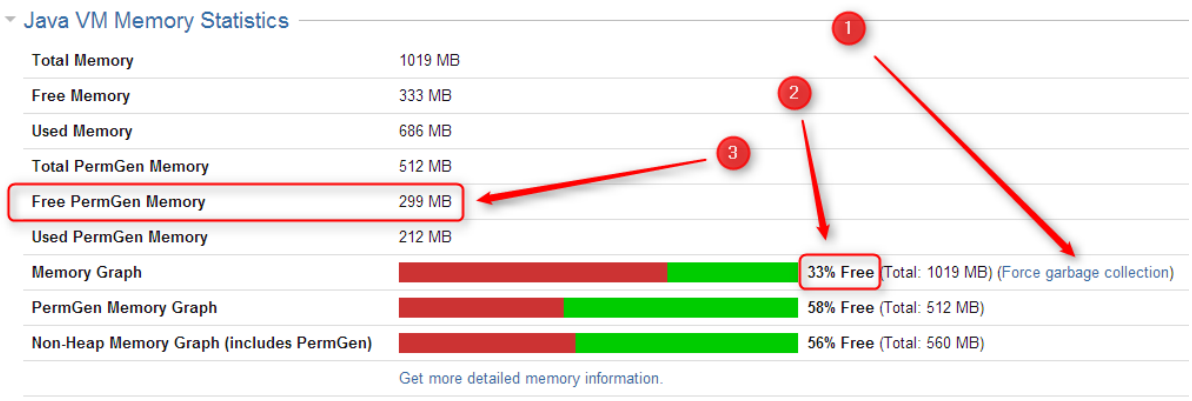
Next: Set up Structure license key

3.1.1 Memory Guidelines

On a production system, it is a good idea to check if you have enough free memory in JIRA's Java process before installing Structure (any other plugin too).

Assessing Available Memory

1. Open menu **Administration | Troubleshooting and Support | System Info** and scroll down to **Java VM Memory Statistics**.
2. Click **Force Garbage Collection**
3. Note the free % number of the **Memory Graph** (heap memory).
4. Note the absolute amount of **Free PermGen Memory** (non-heap memory for Java classes).



Memory Statistic	Recommended Value	Parameter in `setenv.sh` / `setenv.bat`
% of Free Heap Memory	25% – 50%	`JVM_MAXIMUM_MEMORY`
Free PermGen Memory	100 – 200 MB	`JIRA_MAX_PERM_SIZE`

All recommendations are for a general case and do not guarantee that you don't get `OutOfMemoryError`. Individual cases may vary.

Heap Memory Requirements

It is recommended that % of free heap memory is from 25% to 50%.

Structure requires about additional 100 MB of heap memory. You can take your current statistic of **Used Memory** and **Total Memory**, add 100 MB to the **Used Memory** and calculate the recommended value for the **Total Memory**.

If you already have recommended % of free memory, you can just increase total heap memory by 200 MB.

PermGen Memory Requirements

This section applies to JIRA running on Sun/Oracle Hotspot Java VM only.

PermGen space is used for Java classes and may be depleted if you uninstall, install or upgrade plugins frequently, or if you don't restart JIRA over a long period of time. Due to technical reasons PermGen space might not get cleaned up from the obsolete classes and you may end up with `OutOfMemoryError: PermGen space` error.

Structure classes use only about 10 MB of PermGen space. But for the reasons just mentioned, it is good to have a safety margin with a free PermGen space of at least 100 MB.

The out-of-the-box value for maximum PermGen space is 256 MB, and JIRA without any plugins takes up about 180 MB soon after it's started. The default value is not safe enough!

Changing Memory Parameters

To change memory parameters, edit `setenv.sh` (on Windows, `setenv.bat`).

- To change maximum amount of Heap space, edit `JVM_MAXIMUM_MEMORY` parameter near the top of the script.

```
JVM_MAXIMUM_MEMORY="2000m"
```

- To change maximum amount of PermGen space, edit `JIRA_MAX_PERM_SIZE=256m` line. Alternatively, you can add `MaxPermSize` parameter to `JVM_SUPPORT_RECOMMENDED_ARGS`, for example:

```
JVM_SUPPORT_RECOMMENDED_ARGS="-XX:MaxPermSize=400m"
```

You need to restart JIRA for these settings to take effect.

Use 64-Bit Java

It is imperative to use 64-bit Java when allocating large amount of memory to it (1 GB and more). To check if you're running 64-bit Java, look up **Java VM** parameter on the System Info page.

Physical Memory Requirements

Avoid swapping at all costs!

The amount of physical memory should be enough to accommodate the whole heap and non-heap memory. If you have other Java or memory-intensive applications running on the same host, they all should fit in physical memory, plus you need to reserve at least 1 GB for operating system, services and file cache.

Do not allocate more memory to JIRA if it cannot fit into physical memory! If Java running JIRA starts swapping actively used memory, it will be a performance disaster.

Sample calculations for a host running JIRA and Confluence, with Apache and MySQL:

JIRA	Heap: 2 GB Non-heap: 500 MB
Confluence	Heap: 2 GB Non-heap: 500 MB

Operating system Apache HTTPD MySQL	1 GB
Free memory margin / File buffers	2 GB
Total Physical Memory Required	**8 GB**

3.1.2 Uninstalling and Reinstalling Structure

Uninstalling Structure

You can uninstall Structure from Plugin Manager the same way you uninstall other plugins. You can also manually remove structure JAR from `plugins/installed-plugins` directory when JIRA is not running.

When you uninstall Structure plugin, Structure data is **not removed**. It remains in place in `structure` sub-directory under JIRA home. (See Structure Files Location.)

After you have uninstalled Structure, you can completely delete its data by removing `structure` sub-directory. (We advise to at least create a backup copy though.)

Structure License is not kept in the Structure database, but in the main JIRA database. If you'd like to remove Structure license, you need to open **Administration | Structure | License Details** menu and use **Uninstall License** from that page – before you uninstall the plugin.

Reinstalling Structure

It is perfectly safe to uninstall Structure plugin, then install it back again. (This happens, for example, when you upgrade to a newer version.)

All Structure data will be there unless you manually remove it.

3.2 Setting Up Structure License

Unless your JIRA runs on one of the free licenses, Structure requires a license key to work. You can get a free no-obligation 30-day evaluation license key for your JIRA server in a few seconds.

3.2.1 Setting Up Evaluation License

1. Navigate to **Administration | Structure | License Details**.
2. Look at the **Current License** section - if there's no license there or if the license is expired, then you need to get an evaluation license or purchase a commercial license.

- If **Current License** section says that you have a **Free License**, then your JIRA must be qualifying for automatic free license and no further action is needed from you. See When Structure is Available for Free.

3. To get a free 30-day unlimited-users evaluation license, follow **Get Evaluation License** link on the structure license page, or open evaluation license request page directly. In latter case please enter your JIRA Server ID to get a correct license.

You can also get evaluation license from Atlassian in the **Plugin Manager** by clicking a button named **Try** or **Free Trial**.

If you have installed a license from ALM Works, Plugin Manager may show that Structure is *Unlicensed* or *Action Required*, because it's not aware of ALM Works license. You can check true license status on **Administration | Structure | License Details** page — if it shows you that the license is OK, you can safely ignore the status of the license in Plugin Manager.

3.2.2 Licenses from ALM Works and from Atlassian

Structure support two kinds of licenses — issued by ALM Works and issued by Atlassian. These licenses are functionally equal — you can use either kind to get the same functionality in Structure. The prices are also the same.

The following table summarizes the differences and provides instructions for both kinds.

	License from ALM Works	License from Atlassian
Purchased at	ALM Works website	Atlassian Marketplace
Managed at	The license key is sent to you by email	Manage with other Atlassian my.atlassian.com
License key looks like this:	`-----BEGIN CERTIFICATE----- MIIEYTCCAkmgAwIBAgIGAT2oPFqOMA0GCSqGSIb3DQE... ... at least 20 lines of symbols ... -----END CERTIFICATE-----`	`AAABEA0ODAoPeNp9UE1Pg ... at least 4 lines`
Installation Instructions	1. If you have a license from Atlassian installed, first remove it in the **Plugin Manager**. 2. Open **Administration | Structure | License Details**. 3. Copy and paste the key to the **Install License** section and click **Install License**.	1. Open **Plugin Manag** 2. Locate and open Stru 3. Copy the license key click **Update**.
Uninstallation Instructions	1. Open **Administration | Structure | License Details**.	1. Open **Plugin Manag**

	License from ALM Works	License from Atlassian
	2. See the details of installed license and click **Uninstall**.	2. Locate and open Stru 3. Clear the license key click **Update**.
Purchasing differences	• Besides advance payments with credit card, wire transfer or other payment methods supported by our payment processor, we can also accepts purchase orders on Net 30 terms. • VAT and taxes may be handled differently from Atlassian, as our payment processors are located in USA and Germany. ALM Works is based in Russia, and for direct purchases using Wire Transfer we do not charge VAT or any other taxes.	• Purchasing from Atla certain countries.

3.2.3 Purchasing a Commercial License

Structure license can be purchased from ALM Works, from Atlassian, or through resellers and Atlassian Experts.

Purchasing from ALM Works

Commercial license from ALM Works can be purchased at http://almworks.com/structure/purchase.html.

To generate a license, JIRA Server ID is required. JIRA Server ID is a 16-digit code, which JIRA Administrator can look up in JIRA menu **Administration | System Info** or in **Administration | Structure | License Details**.

Purchasing from Atlassian

You can purchase a license via Atlassian on the Atlassian Marketplace.

After purchase is completed, the license key will be available on https://my.atlassian.com.

Purchasing from Resellers or Atlassian Experts

You can purchase through a reseller of your choice. Atlassian Experts can also provide you with additional services and advice.

When you purchase through a reseller, you can get either kind of license (issued by ALM Works or by Atlassian), depending on the reseller's actions. If you prefer one kind of license over another, please don't forget to tell that to the reseller.

3.2.4 Migrating Licenses

You can convert a license of one kind into a license of another kind. Please contact sales@almworks.com for assistance.

Next: Select which projects are enabled for Structure

3.2.5 Structure License Parameters

The following parameters are displayed in the **Current License** section when you install a Structure license.

Parameter	Meaning
License Type	Commercial or Evaluation, may be others
Licensee	Organization authorized to use the license
Serial Number	A unique number assigned to the license
Expires	If present, the license is not perpetual: it will expire at the specified date. After that date passes, the Structure plugin will not be available unless the license key is changed.
Maintenance Expires	If present, the license key can only work with the versions of the Structure plugin released prior to the specified date. If you need to use a newer version of the Structure, you need to renew maintenance.
User Limit	This is the maximum number of users allowed by JIRA that are supported by this license key. The license that JIRA runs on must allow this number or fewer users.
Server ID	Although not shown in the license table, most licenses are tied to a specific JIRA server ID and would not install on a server with a different ID. If you need to move a license key to a different server, please contact support.

3.2.6 When Structure is Available for Free

Structure plugin automatically installs a free license in case your JIRA runs on one of the following free licenses:

- Free license for **open-source** projects;
- Free license for a **non-profit** organization;
- Free **community** license;
- Free **demonstration** license;
- Free **developer** license.

The clauses from the Atlassian EULA that govern the use of those free licenses also apply to using Structure on JIRA servers where these licenses are installed.

3.2.7 License Maintenance and Expiration

Commercial License

Your commercial license for the Structure plugin (including Starter licenses) typically has no expiration date, so it's good to use forever. However, it has *Maintenance Expiration Date* which limits which versions of the plugin can be used with that license – you can only use the versions released prior to that date.

To use versions released later, you need to purchase maintenance renewal, which extends your maintenance expiration date one year forward – independently of the date of purchase.

Example:

Date license purchased	2012-01-01
License expiration date	None
Maintenance expiration date	2013-01-01
Products and terms allowed by the license	All versions released prior to 2013-01-01 can be used indefinitely
Maintenance renewal purchased	2012-12-10 (doesn't matter)
Renewed license maintenance expiration date	2014-01-01
Renewed terms	All versions released prior to 2014-01-01 can be used indefinitely

Evaluation License

Evaluation and temporary licenses have expiration date, after which they just stop working – they allow to use the product before the specified date.

Make sure you renew evaluation or get another license key before expiration.

License expiration and maintenance expiration warnings

If the currently used license becomes invalid (for example, because it is expired, or because you've upgraded to a version of Structure that's not covered by the current license's maintenance), then Structure plugin will function in read-only mode.

The users will be able to view structures, but they won't be able to make any changes until a valid license is installed.

3.3 Selecting Structure-Enabled Projects

Structure can be enabled for any selection of the JIRA projects, or for none of them. (In the latter case noone can use Structure.)

By default, Structure is not enabled for any project. To start working or evaluating Structure, enable it for at least one project.

To select which projects are enabled for Structure:

1. Navigate to **Administration | Structure | Configuration**.
2. Click **Enable/Disable Structure in Projects**.
3. Select whether Structure should be available for **all projects** or for **selected projects**.
4. In the latter case, change the projects list in the **Selected Projects** list by selecting one or more projects and using **Enable** and **Disable** buttons.
5. Click **Apply** when done.
6. In case you have disabled some projects that are already used in a structure (a structure contains issues from that project), you'll be given a warning. You can opt to **Proceed with Changes** or cancel.
    a. If you proceed and disable a project that has issues in some structures, those structures will appear to the users without those issues.
    b. If you later enable that project back - the issues will reappear where they were (all structure changes taken into account).

Which projects are enabled for the Structure affects Who Has Access to the Structure

3.4 Who Has Access to the Structure

Structure is visible only to specific users. Those users who do not have access to the Structure, will not see *Structure* menu and other user interface elements, provided by the Structure plugin.

A user has access to Structure if all of the following conditions are met:

- The user has **Browse** permission on at least one of the projects that are enabled for Structure.
- Structure is enabled for this user:
    - Either Structure is enabled for everyone,
    - Or the user belongs to at least one of the enabled groups.
    - Or the user belongs to at least one of the enabled role in specified project; if role is enabled for "Any" project, the user must be in this role in any of the projects that are enabled for Structure.

Users who have *JIRA Administrators* global permission always have access to Structure.

3.4.1 Restricting User Access to Structure

By default, Structure is accessible to anyone who has *Browse* permission on structure-enabled projects. You can further restrict this access level to one or more user groups.

To select who can use Structure:

1. Navigate to **Administration | Structure | Configuration**.
2. Click **Select Structure Users**.
3. Select whether Structure should be available to **Everyone** or to **Users in selected groups/roles**.
4. In the latter case, change the **selected groups/roles** list by selecting second radio button and use **Add Group/Role** section to add one or more required user groups or project roles. To set up required property, use drop-down selectors to choose either **Group** or **Project** option, then choose required group name or project/role combination and press **Add** button to add it to the list. If **project** is set to "Any", this means that user should be in specified role for any of structure-enabled projects.
5. You can remove permission option by clicking trash can icon on the right of the option.
6. Click **Apply** when done or **Cancel** to dismiss your changes.

Which projects are enabled for the Structure also affects Who Has Access to the Structure.

When Structure is enabled for **anyone**, even anonymous visitors will have access to Structure. To make Structure accessible to only logged in users, restrict access to **jira-users** group.

Structure plugin maintains a cache of users permissions with regards to each structure. In most cases, the cache is recalculated automatically, but in some cases Structure plugin may miss a change in a user's groups or roles. The result could be that the changed permissions take effect several minutes later (but only with regards to Structure Permissions). A user can force the cache to be recalculated by doing **hard refresh** from the browser. Typically, it's done by holding **Ctrl** or **Shift** or both and clicking the **Refresh** button.

3.5 Changing Permission to Create New Structures

By default, any logged-in user with access to Structure can create new structures of their own. However, you can restrict this ability to one or more user groups.

To select who can create new structures:

1. Navigate to **Administration | Structure | Configuration**.
2. Click **Select Who Can Create Structures**.
3. Select whether new structures can be created by **Anyone with access to Structure** or by **Users in selected groups/roles**.

4. In the latter case, change the **selected groups/roles** list by selecting second radio button and use **Add Group/Role** section to add one or more required user groups or project roles. To set up required property, use drop-down selectors to choose either **Group** or **Project** option, then choose required group name or project/role combination and press **Add** button to add it to the list. If **project** is set to "Any", this means that user should be in specified role for any of structure-enabled projects.
5. You can remove permission option by clicking trash can icon on the right of the option.
6. Click **Apply** when done or **Cancel** to dismiss your changes.

The user also needs general access to Structure to be able to create new structures.

Users who have *JIRA Administrators* global permission are always allowed to create new structures.

Structure plugin maintains a cache of users permissions with regards to each structure. In most cases, the cache is recalculated automatically, but in some cases Structure plugin may miss a change in a user's groups or roles. The result could be that the changed permissions take effect several minutes later (but only with regards to Structure Permissions). A user can force the cache to be recalculated by doing **hard refresh** from the browser. Typically, it's done by holding **Ctrl** or **Shift** or both and clicking the **Refresh** button.

3.6 Changing Permission to Manage Synchronizers

By default, any logged-in user with Control access level for a structure can manage that structure's Synchronizers. However, you can restrict this ability to one or more user groups.

To select who can manage synchronizers:

1. Navigate to **Administration | Structure | Configuration**.
2. Click **Select Who Can Control Synchronizers** .
3. Select whether synchronizers can be managed by **Anyone with control access to the structure** or by **Users in selected groups/roles**.
4. In the latter case, change the **selected groups/roles** list by selecting the second radio button and use **Add Group/Role** section to add one or more required user groups or project roles. To set up the required property, use drop-down selectors to choose either the **Group** or **Project** option, then choose the required group name or project/role combination and press the **Add** button to add it to the list. If **project** is set to "Any", this means that the user should be in the specified role for any of the structure-enabled projects.
5. You can remove a permission option by clicking the trash can icon on the right of the option.
6. Click **Apply** when done or **Cancel** to dismiss your changes.

The user also needs Control access level for a structure to be able to manage its synchronizers.

Users who have *JIRA Administrators* global permission are always allowed to manage synchronizers.

3.7 Changing Structure Defaults

JIRA administrator can adjust a number of Structure "defaults", settings that apply when the user does not specify a more specific request or option.

- Changing Default Structure
- Changing Default View Settings
- Changing Default Options for the Issue and Project Pages

3.7.1 Initial Configuration

When Structure plugin is installed, the defaults are configured as follows:

System Default Structure	**Global Structure** - an automatically created structure with ID = 1.
Project Default Structure	Not set - all projects use system default.
Default Views Menu	Preinstalled views **Structure**, **Planning**, **Tracking**, **Triage**, **Entry** (on all pages)
Default View	**Structure**
Auto-switch (Issue Page)	Structure with the displayed issue.
Auto-switch (Project Page)	**Off** - show the last viewed structure.
Keep structure when navigating	**On** - When going from structure widget to an issue page, show the same structure.
Auto-minimize Structure Panel	**On** - Structure panel initially minimized if issue is not in structure.

3.7.2 Changing Default Structure

Default structure is selected when the user opens Structure Board for the first time, or when the Auto-switch option is set to *default structure*. You can change the default structure for the JIRA instance and for a specific project.

Changing system-level default structure

1. Open **Administration | Structure | Defaults** menu.
2. In the **System Default Structure** section, click **Change**.
3. Select the default structure and click **Apply**.

The new system-level default structure will be also default for all structure-enabled projects that don't have this setting overridden.

⚠ Make sure that default structure has correct permissions. If the structure is selected for the user by default, but the user does not have `VIEW` access to it, the user will see an error.

Changing project-level default structure

1. Open **Administration | Structure | Defaults** menu.
2. Locate the project in the **Project Default Structures** section. Un-check **Show only projects with overridden default structure** checkbox if needed. Click **Change** in the corresponding row.
3. Select a structure and click **Change**.
4. or, select **Use system default** to remove the project-level default.

Project administrator can also change project-level structure from the **Structure** tab on the project administration page, or from the options pop-up window on the **Structure** tab on the user's project page.

3.7.3 Changing Default View Settings

View settings determine which views are offered to the users in the Views Menu (on the Structure Board and other pages with Structure widget). Default view settings apply to all structures that don't have view settings customized, configured by a structure administrator (someone who has **Control** permission for that structure) via **Views** link on the **Manage Structures** page.

To change default view settings:

1. Open **Administration | Structure | Defaults** menu.
2. In the **Default View Settings** section, click **Change**.
3. Modify the default settings - for details, see Customizing View Settings.
4. Click Apply

3.7.4 Changing Default Options for the Issue and Project Pages

A number of options define how Structure Panel behaves on the issue page and on the project, component and version pages. When the user opens those pages for the first time, the default settings apply. These settings are adjustable by JIRA administrator.

If the user changes some of the options, those changes are preserved and are applied instead of defaults for that specific user.

To change the defaults:

1. Open **Administration | Structure | Defaults** menu.
2. Scroll down to **Structure User Interface Defaults** and click **Change**.
3. Make the changes and click **Change** again.

Option	Description	See Also
Auto-switch (Issue Page)	Lets you automatically select structure displayed on the Issue page.	Structure Options for the Issue Page
Auto-switch (Project Page)	Lets you automatically select structure displayed on the Project, Component and Version pages.	Structure on the Project, Component and Version Pages
Keep Structure Selection When Navigating	When turned on, clicking an issue in the Structure Widget opens that issue's page and shows the same structure on that page initially.	Structure Options for the Issue Page
Auto-minimize Structure Panel	If turned on, the Structure Panel on the issue page will be initially minimized in case the selected structure does not contain the displayed issue.	Structure Options for the Issue Page

3.8 Structure Backup, Restore, and Migration

Structure data is backed up and restored separately from other JIRA data. Structure data includes structures, hierarchies, synchronizers - everything added to JIRA by the Structure plugin. Structure backup does not include issue data (except for some issue attributes that are added just in case).

Structure is backed up separately due to technical limitations of the supported versions of JIRA platform. In the future we plan to merge Structure backup with the main JIRA backup.

You need the *JIRA System Administrators* global permission to back up, restore, or migrate Structure data.

A backup file can be used in two ways:

- **Full structure restore.** This operation replaces all existing structure data (if any) with the data stored in the backup file. This operation refers to issues by their numeric IDs (*not* issue keys!), so the issues must be present in JIRA before this operation is run, and issue IDs must be the same as they were at the time the Structure backup file was created.

> ⚠ Issue IDs are preserved if JIRA instance is fully restored from backup with **Restore System** command. Issue IDs are *not* preserved if the issues are moved to another JIRA instance with JIRA's **Project Import** feature – use structure migration in this case.

- **Migration / partial import.** This operation lets you restore one or more structures backed up at a different JIRA instance (assuming that the issues have been moved over with the JIRA's **Project Import** command). It also allows you to merge the backed up structure data with the structure data already existing on your JIRA.

> ⚠ A structure in a backup file cannot be restored if it refers to issues in a project that is not present in the JIRA instance.

3.8.1 Proper Backup Sequence

To back up your JIRA with structure:

1. Perform the standard JIRA backup.
2. Back up Structure.

Of course you can back up structure separately if all you need to copy is the hierarchies and structure configuration.

3.8.2 Proper Restore Sequence

To restore your JIRA with structure:

1. Restore JIRA data first.
2. Restore structure from backup.

3.8.3 Proper Import Sequence

To import your JIRA projects with structure to another JIRA instance:

1. Import necessary JIRA projects from JIRA backup.
2. Migrate structure

3.8.4 File-Based Backup

You can also back up and restore the whole `structure` directory in the JIRA home (see Structure Files Location) manually - but only when JIRA is not running or when Structure plugin is disabled. When restoring structure data backed up this way, make sure you're using the same version of the Structure Plugin that was there when the backup was created.

Detailed Instructions:

- Backing Up Structure
- Restoring Structure from Backup
- Migrating Structures

3.8.5 Backing Up Structure

Backing up Structure saves the existing structures, their configuration and issue hierarchies. Structure backup does not save the issues themselves or other JIRA data - see Structure Backup, Restore, and Migration.

To back up Structure:

1. Navigate to **Administration | Structure | Backup Structure**.
2. Enter the name for the backup file. If you omit file extension, either *.xml* or *.zip* will be added to it.

   > ⚠ You cannot specify directory for the backup file. Backup is always done to the *export* sub-directory under JIRA home.

3. Click **Backup**
4. If the file already exists, you will be given an option to overwrite the file or cancel the operation.

3.8.6 Restoring Structure from Backup

Restoring structure from backup brings back the structures, synchronizers and views created at the moment of backup.

> ⚠ Restoring structure will not affect issues in any way or restore them. The issues that make up the hierarchy should already exist in JIRA. If you do full restore, then you need to run the standard JIRA data restore first - see Structure Backup, Restore, and Migration.

> ⚠

The issues in the structures are identified by their internal numeric ID. If you have transferred issues via JIRA's Project Import, issue IDs have changed and so you need to use Structure Migration.

Use Restore Structure when:

- the backup was made on this JIRA instance or on its predecessor,
- and, you need to fully restore structure data,
- and, you can lose the current structure data stored on this JIRA instance (issues are not affected, only their organization into structures).

To restore the structure from backup:

1. Navigate to **Administration | Structure | Restore Structure**.
2. Enter the full path to the structure backup file (either *.xml* or *.zip*).
3. Click **Restore**.
4. If Structure currently has any data, it will ask you to confirm the restore operation. Restoring from backup clears all Structure data, and it cannot be undone! If you have data that you're overwriting, you might want to perform Backup first.
5. Wait until page loads – Structure data has been restored! The plugin will be restarted and will be available a few seconds after Restore has completed.

After the structure has been restored, open **Structure | Manage Structure** page to see if the structures are there.

You also can restore structure data from backup files made with the earlier versions of the Structure plugin.

3.8.7 Migrating Structures

Migrating structure data lets you import one or more structures from a different JIRA instance after you have imported projects with the JIRA's Project Import operation. Also, you can add some structures from a backup file to those that are already present in JIRA.

Migrating structure will not affect issues in any way. The issues that make up the hierarchy should already exist in JIRA. You may need to run JIRA project import or the standard JIRA data restore first - see Structure Backup, Restore, and Migration.

During migration the issues in the structures are located in JIRA by their issue keys and a possibly new numeric ID is being used to construct the structure. A structure cannot be migrated if it refers to issues from a project that is missing in JIRA.

When migrating a structure and there's already an existing structure with the same ID or name, you will have an option to either replace the existing structure with the structure from the backup, or restore the structure from backup as a separate structure, or skip this structure.

To migrate structures from backup:

1. Navigate to **Administration | Structure | Migrate Structure**.
2. Enter the full path to the structure backup file (either *.xml* or *.zip*).
3. Click **Select Structures To Restore**.
4. Select structures that should be restored. If there's an existing structure with same ID or name, select **Overwrite Existing** to replace the existing structure with the one from backup, otherwise the structure will be restored as a new structure, leaving the existing one unaltered.
5. Under the list of structures there's a list of additional restore options:

Restore Structure Permissions	If selected, the plugin will attempt to restore the access permissions for the imported structures. This attempt may fail, for example, if the permission rules refer to users or groups not present in JIRA. If you don't select this option, or if the attempt to restore permissions fails, then the restored structure will have no permission rules, letting JIRA administrators further configure them through Manage Structures page.
Restore Synchronizers	If selected, the synchronizers for selected structures are restored. ⚠ Synchronizers configuration is imported as-is, and might not make sense on a new JIRA instance. After you have restored synchronizers, please visit Synchronization Settings page to check if the synchronizers are configured correctly.
Restore Structure History	If selected, structures are imported along with their history (if it is present in the backup file). If not selected, structures will have no history.
Restore User Favorites	If selected, the plugin will try to restore "favorite" marks made by users for the selected structures.
Restore Views	If selected, all views from the backup files will be restored. If there's a conflict and a view with a given ID already exists, Structure will first verify if the view being restored is different from the one in the system, and if it is, restore it as a new view with a different ID.
Restore View Settings for Structures	If selected, view settings for the selected structures will be restored.

6. Click **Restore**.

7. Wait until page loads – structures have has been migrated! The plugin will be restarted and will be available a few seconds after migration has completed.

After structures have been migrated, open **Structure | Manage Structure** page to see if new structures are there.

3.9 Automatic Structure Maintenance

3.9.1 Automatic Structure Maintenance

Automatic Structure maintenance runs daily and performs Structure backup and Structure database optimization.

- As structures, views, synchronizers are stored separately from JIRA data, they are not backed up with the usual automatic JIRA backup. Daily Structure backup is recommended as a part of your backup strategy.
- Structure database optimization removes stale data from the database and may improve general JIRA responsiveness.

To configure automatic Structure maintenance:

1. Navigate to **Administration | Structure | Maintenance**
2. Click **Configure Scheduled Maintenance**
3. If scheduled maintenance is disabled, click **Enable scheduled maintenance**
4. Select time at which maintenance should run every day.

The time is specified in the server's time zone, displayed near the time fields.

5. Select tasks that scheduled maintenance should run.
6. Configure additional task parameters, if any.
7. Click **Apply**

By default, scheduled maintenance is enabled and set to run daily at 3 AM.

Automatic maintenance can be run only when Structure license is valid.

3.9.2 Maintenance Tasks

Backup Structure Data	Creates a backup of the Structure database in the `export` sub-directory under JIRA home. Parameters: **Include history** - if checked, full structure change history will be included in the backup
Delete Old Backups	A backup is considered old if it's not among *X* latest backups (*X* is specified by the first parameter of this task) **and** it was made earlier than *Y* days ago (*Y* is specified by the second parameter). This task removes all such backups made by the Backup task. Parameters: **Always keep *X* latest backups** **Always keep backups made during last *Y* days**
Optimize Favorites	If a user marks a structure as their favorite, Structure plugin will keep this mark even if the user is later deleted from JIRA. Popularity number of the structure will also account for this user. This task removes marks made by users no longer in JIRA and recounts structure popularity.
Optimize Structures	If an issue is added to a structure and then deleted from JIRA, that structure will still contain a reference to this issue (although it will not display it). This task removes references to deleted issues from structures.
Optimize View Settings	If a view is deleted, some structure view settings may still reference it, and a blank view named **? (Unknown View)** will be shown in its place. This task removes references to the deleted views.
Optimize Synchronizers	Sometimes Structure may keep data related to synchronizers of a structure which was already deleted. This task removes such data.
Reindex Change History	If a user makes a big change in a structure with a single action (say, moves an issue with 500 sub-issues), this change might not be shown in any Structure Activity Stream that is filtered by issues or projects. This task brings such changes back into these streams.
Optimize Structure Perspectives	Removes old perspectives that haven't been used by anyone for a certain amount of time. Parameters: **Delete perspectives that were not used during the last *X* days**

3.9.3 Running Maintenance Tasks Manually

You can run specific maintenance tasks at any time.

To run maintenance manually:

1. Navigate to **Administration | Structure | Maintenance**
2. Navigate to **Run Maintenance Now** section
3. Select tasks to run.
4. Configure additional task parameters, if any.
5. Click **Run Maintenance Now**

Running maintenance manually does not affect automatic maintenance settings or schedule.

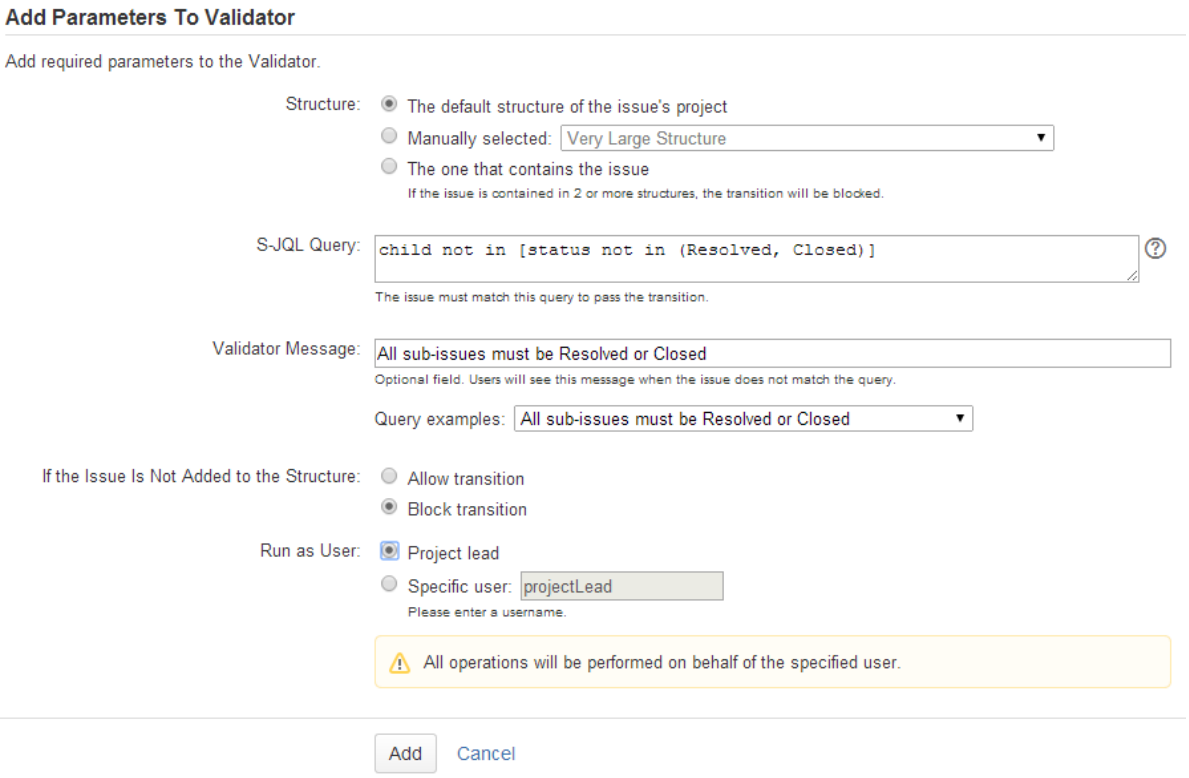
3.10 Workflow Integration

3.10.1 Structure Workflow Validator

Structure Plugin adds a new workflow transition validator to JIRA. This validator blocks the transition if the issue doesn't match an S-JQL query. For example, it can be used to prevent an issue from being resolved if the issue has some unresolved sub-issues in a structure.

To add the Structure validator to a workflow:

1. Create a draft of the workflow and open the **Add Validator to Transition** dialog. (For more information, please refer to the JIRA documentation.)
2. Select the **Issue Matches Structure Query** validator. A configuration window will open:



3. In the **Structure** field, specify how validator should select the structure to check. It can either be a manually selected structure, or it may depend on the issue being checked (the default structure of the issue's project or the structure that contains the issue).
4. In the **S-JQL Query** field, enter the S-JQL query that the issue should match in order to pass the transition.

> You can use one of the examples provided with the form. Just select an example in the **Query Examples** selector, and the corresponding query will be copied into the **S-JQL Query** field.

5. In the **Validator Message** field, enter an explanation message that users will see if their transitions are blocked by the validator.
6. In the **If the Issue Is Not Added to the Structure** field, select whether the transition should be blocked or allowed if the issue is not contained in the checked structure. (Or if the issue does not belong to any structure, in case automatic structure selection is chosen.)
7. In the **Run as User** field, select on behalf of which user the validator should run. It can either be a manually selected user, or it may be the lead of the project of the issue that is being checked.

> Running on behalf of a user means that the validator will only see issues and structures that are accessible to the specified user. The result of the validator check will depend on the permissions of the specified user and will not depend on permissions of the user who performs the transition.

3.10.2 Structure Workflow Condition

Structure Plugin also comes with the Structure condition that is similar to the Structure validator. Using Structure condition may significantly increase the load on server, that's why this condition is not available by default.

To make Structure Workflow Condition available, enable the **structure-workflow-condition** module of the Structure plugin via **Administration | Add-ons | Manage Add-ons** page. For instructions, please see Universal Plugin Manager documentation.

> Checking S-JQL condition may involve querying other issues in the checked structures, and in case the structure is large this may take considerable time – yet within reason if it is done occasionally.
>
> However, workflow conditions are checked every time when a user opens an issue details page, in order to decide which transitions to show. If you have hundreds of active users and thousands of issues in a structure, this may easily degrade server performance.
>
> Use your own best judgement.

3.11 Anonymous Usage Statistics

Please enable anonymous usage statistics, as is helps the developers better understand how Structure plugin is used, prioritize improvement requests and build a better product. No JIRA content or personally identifiable data are collected.

When anonymous usage statistics is enabled, Structure plugin periodically sends some data from the JIRA instance to ALM Works.

The data consists of anonymized information related to the usage of Structure plugin, for example, invocation count of each structure widget action (say, structure history is toggled 56.3 times a day on average, issues are pasted 30.7 times a day on average etc.).

Here's a sample report that is sent to ALM Works: Statistics Sample

3.11.1 Viewing Current Statistics

JIRA administrator can always have a look at the data which is about to be sent. To view the data:

1. Navigate to **Administration | Structure | Support**
2. Click **View Current Statistics**

3.11.2 Turning Anonymous Usage Statistics On and Off

To enable or disable Anonymous Usage Statistics:

1. Navigate to **Administration | Structure | Support**
2. Check or uncheck **Send anonymous usage statistics** checkbox
3. Click **Apply**

The information is collected in accordance with EULA and privacy policy.

3.12 Structure Files Location

3.12.1 Structure Files

$JIRA_HOME/structure

Structure keeps most of its data in the `structure` sub-directory under the JIRA home directory.

Database

Structure uses embedded Apache Derby database engine to store its data in `structure/db` directory.

Undo Logs for Synchronizers

When synchronizers change issues in background - either when running Export or Resync or when running incremental synchronization - they store the undo information in special log files in `structure/syncundo` directory. The log files are kept for some time (~ 30 days by default) and then removed.

In case an incorrectly configured synchronizer ruins issue data (for example, Links synchronizer is able to remove all links of specific type when told to do so), the logs from syncundo sub-directory may be used to recover the data.

Backups

All backup files, made either manually or by Automatic Structure Maintenance, are stored in the `export` sub-directory under JIRA home. Backups made automatically are named `structure-dddddddd-hhhh.zip` (having date instead of `dddddddd` and time instead of `hhhh`).

3.12.2 Backing up Structure Files

When JIRA is not running or when Structure plugin is disabled, you can manually back up structure directory (separately or as a part of JIRA home backup). Likewise, you can manually restore the `structure` subdirectory if Structure plugin is not running - but be careful when downgrading to previous version of the Structure. As a rule, the database is not forward-compatible, so you can't downgrade and keep the database unless the versions of the Structure happen to have the same database schema.

You can also back up Structure into XML by using Structure Backup menu on the administrator's page.

3.13 Turning Off Optional Features

Some features in Structure are designed as modules and can be safely turned off. You can do so to remove unnecessary functionality, or limit the exposure of Structure plugin to the users.

> If your aim is to limit the exposure of Structure, consider restricting permissions to specific groups of users - see Gradual Deployment.

> While it is easy to disable a Structure module, we don't recommend to touch any modules except those listed in this article to ensure stability of Structure and your JIRA application.

To turn off a module:

1. Open Plugin Manager by navigating to *Administration* | *Plugins* | *Manage Plugins*.
2. Locate Structure plugin and expand its row.

3. Click the link that looks like the following: "117 of 117 modules enabled."
4. Use Search feature of your browser to find the module by its name (provided below.)
5. Click the Disable button to the right of the module name.

You can always turn the feature on later by clicking the Enable button.

Feature	Module name	Effect of disabling this module
Activity Streams	`Structure (structure-activity-provider)`	Activity streams provider and Structure-related updates are removed from the following places: • Activity Stream gadgets, • Activity tab on the issue page, • Activity tab on the user page, • Activity tab on the project page.
Structure on the Issue Page	`servlet-filter:Issue Page Decorator (issue-page-decorator)`	Structure section is removed from the issue page.
Structure on Agile Boards	`web-panel:GreenHopper tab (greenhopper-tab)`	Structure tab is removed from the issue details panel on the Agile board.
Synchronizers	`synchronizer:...` (5 synchronizers are bundled with Structure)	Users will not be able to install synchronizers, and installed synchronizers won't run. You will need to restart the plugin to have settings make full effect. (Disable plugin, then enable plugin.)

3.14 Advanced Configuration with System Properties

Certain advanced aspects of Structure's behavior might not have dedicated configuration pages, being controlled by system properties instead. This page lists Structure-related system properties and describes how to set them.

3.14.1 Setting System Properties on Startup

You can set a system property using the `-D` JIRA startup option, for example:

```
-Dstructure.sync.guard.email.admin.cycles=5
```

Configuring JIRA startup options is described in this article. You will need to restart JIRA for the properties to take effect.

3.14.2 Setting System Properties with Script Runner

If you don't want to restart JIRA, you may use the free Script Runner add-on to set system properties.

1. Install Script Runner.
2. Go to **Administration | Add-Ons | Script Runner | Script Console**.
3. Select **Groovy** as the Script Engine.
4. Enter the following code into the Script text box, adjust property name and value as needed, and click **Run Now**.

```
System.setProperty("structure.sync.guard.email.admin.cycles", "5")
```

The changes take effect immediately, but the properties will be reset to their default values when you restart JIRA. If you want the changes to be permanent, please use the `-D` startup option as described above.

3.14.3 Synchronizer Cycle Guard

The cycle guard is a component that detects conflicting synchronizers and prevents them from cycling forever, overriding each other's changes. The table below describes the system properties that control the cycle guard.

Property	Default	Explanation
`structure.sync.guard.disable`	`false`	Set to `true` to disable the cycle guard. Conflicting synchronizers will not be prevented from running forever. **Not recommended.**
`structure.sync.guard.maxAutosyncsWithoutUserChanges`	`10`	The maximum number of times that a synchronizer is allowed to run, processing the changes generated by another synchronizer. If this limit is exceeded, the two synchronizers are considered to be in conflict.
`structure.sync.guard.stop.disable`	`false`	If `true`, conflicting synchronizers will not be disabled automatically.

Property	Default	Explanation
		The cycling may repeat after a user-generated change.
`structure.sync.guard.email.owner.disable`	`false`	If `true`, the cycle guard will never send e-mail notifications to synchronizer owners.
`structure.sync.guard.email.admin.disable`	`false`	If `true`, the cycle guard will never send e-mail notifications to JIRA administrators.
`structure.sync.guard.email.admin.cycles`	`10`	The minimum number of times a cycle must be detected for a synchronizer before an e-mail notification about that synchronizer is sent to JIRA administrators. The counter is reset when a synchronizer is automatically disabled, so if this number is greater than 1 and automatic disabling is on, the administrators will not be notified.

3.15 System Requirements

The following are additions to JIRA Requirements imposed by the Structure.

3.15.1 Browsers

Structure Plugin is compatible with the following browsers:

Browser	Versions	Known to NOT work
Mozilla Firefox	3.6+	3.5
Chrome	5+	

Browser	Versions	Known to NOT work
Internet Explorer	8+	6
Safari	5+	
Opera	10+ (not regularly tested)	

3.15.2 Server Requirements

- Structure needs at least 100MB of free disk space on the server. See Structure Files Location for details.
- Java process running JIRA needs at least additional 200 MB of heap memory. Ensuring sufficient free PermGen space is recommended. See Memory Guidelines for details.

> Extra memory is used by the embedded Derby database and other Structure data and caches. Normal memory consumption by Structure is up to 100 MB. These extra memory requirements are multiplied by 2, a typical free memory factor.

- JIRA process must have read/write permissions to JIRA home directory to create `structure` sub-directory automatically.
  - Alternatively, you can create `structure` sub-directory under JIRA home manually and give JIRA process read/write permissions to write in that sub-directory.

3.15.3 Non-conforming systems

With regards to systems that don't conform to JIRA requirements and Structure requirements: while we sometimes know that a specific configuration doesn't work, more often it's grey area so feel free to try and let us know the results.

3.16 Best Practices

We have collected several guidelines for common situations.

- Backup Strategy
- Gradual Deployment

If you have your own best practice to suggest, please let us know!

3.16.1 Backup Strategy

Structure data is stored separately from JIRA data and is not included in the general System Backup. To ensure that your Structure information is safe, it recommended to make Structure backup a part of your overall backup strategy.

Option 1. Automatic XML Backup + Export Directory Backup

This strategy involves two processes:

- Automatic Structure Maintenance lets you automatically create full hot backups of the Structure data once a day. The backups are stored in the `export` directory under JIRA home.
- Periodic file-level backup of the `export` directory (or the whole JIRA home) to a different storage device increases the safety of the backups. This part should be configured manually by the server administrator.

This is the recommended strategy because it does not require stopping JIRA and can be integrated with already existing overall backup strategy.

When you install Structure, automatic daily backups are enabled by default. You only need to make sure that backup files that will appear in the `export` directory are stored safely.

Option 2. File-Based Backup

Structure stores all its data in `structure/` sub-directory of the JIRA home directory – see Structure Files Location. You can use your operating system tools to back up the whole `structure` directory.

So if you already have backup strategy for the whole JIRA home directory, then you are also backing up Structure.

Hot File-Based Backup

Hot backup can be used only as a complimentary backup strategy to the more reliable automatic XML backup.

Hot backup (while JIRA and Structure are running) is **unsafe**! There's a probability that a database file will be copied while it's being written, making the copy of the database corrupt. (But you can verify that the copied database is correct by opening it with Derby tools and reading all tables.)

When running JIRA on Windows, you won't be able to do hot backup by normal file copying, because the files in `structure` directory will be open. You can use **volume shadow copy** tools from Microsoft to copy a snapshot of the files.

Cold File-Based Backup

Cold backup is safe and sufficient, but it can be done only by disabling Structure plugin, making a backup and enabling Structure again. (There's no need to stop JIRA.)

Option 3. Manual / API-Triggered XML Backup

You can manually back up structure through Structure Backup menu.

If automatic Structure maintenance does not suit you and you have resources to develop your own mini-plugin for backup strategy, you can automatically back up Structure data through the Structure API (use `StructureBackupManager` interface).

Restoring from File Backup

In case of failure, you can restore Structure data from file backup by disabling Structure plugin, copying the whole contents of `structure/` directory back in place, and enabling Structure again.

Watch the logs - if you see a message that the database is corrupt, it is likely that you are restoring from a hot file-based backup. In that case, you'll need to find a cold file-based backup or an XML backup.

Restoring from XML Backup

Restoring from XML backup is error-proof but requires a bit more time. See Restoring Structure from Backup for instructions.

Incremental and Differential Backups

As Structure database is typically not large, full backup is recommended.

Structure XML backup/restore does not support incremental backup, but you can use your operating system tools for incremental or differential backup of the files in `structure` directory.

3.16.2 Gradual Deployment

In an enterprise with JIRA already in production and being used every day, deploying Structure plugin and making it available to everyone might be disruptive – in a good sense, since Structure adds a whole layer of useful functionality to JIRA, but perhaps also in a bad sense, if the users are accustomed to their stable user interface and don't appreciate changes that they do not expect.

As a JIRA admin, you can deal with that situation quite easily by deploying Structure gradually.

Structure can be limited to a number of users – see Restricting User Access to Structure. The users who do not have access to Structure don't see Structure's footprint in JIRA in any way (with one exception, see below).

A common path to gradual deployment is:

1. Create a group called **structure-users** and restrict access to Structure only to that group.
2. Add to the group people who initially championed getting Structure for your company and anybody who actively wants to use it.
3. Let them use Structure and spread the word.
4. Once it is decided that everybody wants to use Structure, remove the restriction.

In the same way, you can gradually enable Structure project-by-project. See Who Has Access to the Structure for details.

Turning Optional Functionality Off

Some Structure features can be turned off – see Turning Off Optional Features.

One notable feature is *Activity Streams*. For technical reasons, even if a user does not have access to Structure, they will still see "Structure" as a possible Activity Streams Provider (although they won't see any events coming out of it). You can turn it off.

Another optional feature to consider is synchronizers. Synchronizers are powerful tools, but they may be harmful if applied carelessly. You can turn off synchronizer modules, or check who in your JIRA has **Bulk Edit** permission.

4 Structure Developer's Guide

4.1 Structure Developer Documentation

Structure for Developers

Structure Plugin contains APIs that allow you to access the hierarchical lists of issues from other plugins and applications. Here are the typical use cases:

Custom Development
You customize JIRA for your customer or employer, and you need to integrate Structure with some other in-house system – see section about integrating plugins and Java API reference.

Plugin Integration
You have your own great JIRA plugin, or plan to create one, and you'd like to use the issue hierarchy provided by Structure – see Accessing Structure from Your Plugin.

Extending Structure
You'd like to extend Structure, adding functionality to the plugin itself – read documentation about extending Structure functionality with additional plugins. (You can also extend the functionality of the Structure plugin itself.)

Remote Access
You need to get or change issue hierarchy remotely from some automated scripts or a client application – read about Accessing Structure Data Remotely and Structure REST API.

4.2 Accessing Structure from Your Plugin

Structure provides a Java API that lets other plugins interact with the Structure data. The API is accessed through a few services that you can have injected into your components.

To start using Structure in your plugin:

4.2.1 Add dependency to your pom.xml

Figure out the version of the API that you need - it may depend on your JIRA and Structure plugin version.

To use API classes, add the following dependency:

```
<dependency>
  <groupId>com.almworks.jira.structure</groupId>
  <artifactId>structure-api</artifactId>
  <version>7.2.0</version>
  <scope>provided</scope>
</dependency>
```

4.2.2 Import an interface

In your atlassian-plugin.xml, use <component-import> module to import the interfaces that you need.

You can import StructureServices and get all other interfaces from there.

```
<component-import key="structure-services"
interface="com.almworks.jira.structure.api.StructureServices"/>
```

4.2.3 Have Structure API service injected into your component.

```
public class MyClass {
  private final StructureManager structureManager;

  public MyClass(StructureServices structureServices) {
    structureManager = structureServices.getStructureManager();
  }

  ...
}
```

Next: Learn API Basics

4.2.4 Structure API Basics

Classes to Start With

Use ...	to ...
StructureManager *and* Forest	create and manipulate structures — add issues to the hierarchy, move issues within hierarchy, delete issues from the hierarchy, listen to structure events

The API uses an open source library Integers, which provides primitive-type collections with `java.util`-like interfaces. When working with `Forest`, you will typically use `LongList` and `LongArray` (an implementation of `LongList`).

See API Usage Samples to get the idea how to work with those interfaces.

The dependency on the Integers library is added automatically when your add dependency on the API. Same goes for another dependency on the small JetBrains annotations library, that provides `@Nullable` and `@NotNull` annotations.

More Power

Use ...	to ...
StructureViewManager	create and manipulate structure views
StructureSynchronizer	create a new synchronizer and declare it in a Synchronizer Module
StructureSyncManager	manage synchronizers
StructureBackupManager	backup structure to a file and restore it back
StructureJobManager	schedule asynchronous jobs
IssueEventBridge	listen for aggregated issue events
StructureQuery	to search for issues added to a structure with hierarchical constraints, as in `structure()` JQL function
Aggregate and AggregateCalculator	calculate aggregate values, such as Total Time Spent or Progress
Aggregates	obtain aggregate instances for summing up time tracking fields or votes
ProgressAggregateFactory	obtain aggregate instances for calculating issue progress (as shown by the Progress column)
DoubleSum and NumericCustomField	create an aggregate instance for summing up a numeric custom field

Next: consider Controlling Compatibility and Making Structure Dependency Optional

4.2.5 Controlling Compatibility

Why Declare Compatible Versions

Structure Java API will change with time, and it is a good practice to ensure that your plugin uses the correct version of the API.

Structure API Versions page explains how version numbers change based on how compatibility is affected. Say, you develop your code using Structure API version **`3.4.5`** – your code will work with any version of the API starting from **`3.4`** and up to, but not including version **`4.0`**.

So what happens if your code is run on JIRA with Structure that provides an incompatible API? It may break, or it may work. The exact answer depends on which parts of the API you use and what are the differences. But if the code breaks, it may not break outright – it may seem to work at first, until it tries to use a method that's not there, for example.

To make your code fail fast, you can declare dependency on a specific range of versions of the Structure API. In that case, if the version of the API is different, your plugin will fail to load and the user will immediately know that there's a problem.

Importing Specific Range of API Versions

You can declare dependency on the specific range of the API versions via OSGi bundle instructions added to your `pom.xml` or `atlassian-plugin.xml`. Figure out the compatible OSGi versions range from the API versions table and modify your `pom.xml` to contain the following:

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-jira-plugin</artifactId>
  <version>3.6</version>
  <extensions>true</extensions>
  <configuration>
    <instructions>
      <Import-Package>
        com.almworks.jira.structure*;version="[3.4,4)",
        com.almworks.integers*;version="0",
        org.jetbrains.annotations;version="0"
      </Import-Package>
    </instructions>
  </configuration>
</plugin>
```

Here we are declaring the acceptable range of versions for the Structure classes, taken from the example above. We don't much care about the versions of Integers and Annotations libraries, so `version="0"` will match any version of those packages.

You may have other `Import-Package` instructions to declare dependency rules for other packages, and you may have other instructions besides `Import-Package` as well. See the API Usage Samples for a more complete example.

Next: Making Structure Dependency Optional

4.2.6 Making Structure Dependency Optional

If you are integrating your plugin with Structure, or when you generally write code that uses Structure API but also should work when Structure Plugin is not present, you need to declare that dependencies are optional and isolate dependencies in the code.

Declare Optional Dependency

Since your plugin must first be loaded as an OSGi bundle, it should declare dependencies from the Structure API packages as optional.

Modify `<Import-Package>` declaration in your `pom.xml` or `atlassian-plugin.xml` and add `resoltion:=optional` classifier. (Add Import-Package to control API compatibility if you don't have this declaration yet.)

```
<Import-Package>
  com.almworks.jira.structure*;version="[3.4,4)";resolution:=optional,
  com.almworks.integers*;version="0";resolution:=optional,
  org.jetbrains.annotations;version="0";resolution:=optional
</Import-Package>
```

Isolate Dependencies in the Code

So once you have declared the optional resolution of the Structure API classes, your bundle will load - but if your code tries to access a class from the Structure API, you'll get a `NoClassDefFoundError`. To avoid that, you need to isolate the dependency on Structure API classes - typically in some wrapper classes.

This is also a point to make design decisions. So your code can use Structure when it's present, and can work independently when Structure is not there. Are there any abstractions that address both of these situation? What are the concepts that are realized through Structure API and through some other means when Structure is not avialable?

Here's a sample wrapper for the Structure API that provides `ForestAccessor` wrapper (whatever it does) when Structure is available and `null` otherwise.

```
public class StructureAccessor {
  public static boolean isStructurePresent() {
    if (!ComponentAccessor.getPluginAccessor().isPluginEnabled("com.almworks.jira.structure")) {
      return false;
    }
    try {
      Class.forName("com.almworks.jira.structure.api.StructureManager");
    } catch (Exception e) {
      return false;
    }
    return true;
  }

  public static ForestAccessor getForest(long structureId, User user) {
    if (!isStructurePresent()) return null;
    StructureManager structureManager;
    try {
      structureManager =
```

```
ComponentManager.getOSGiComponentInstanceOfType(StructureManager.class);
    } catch (Exception e) {
      return null;
    }
    // check user permissions
    if (!structureManager.isAccessible(structureId, user, PermissionLevel.VIEW, false)) {
      return null;
    }
    Forest forest = null;
    try {
      forest = structureManager.getForest(structureId, user, false);
    } catch (StructureException e) {
      return null;
    }
    return new ForestAccessor(forest);
  }
}
```

4.3 Extending Structure Functionality

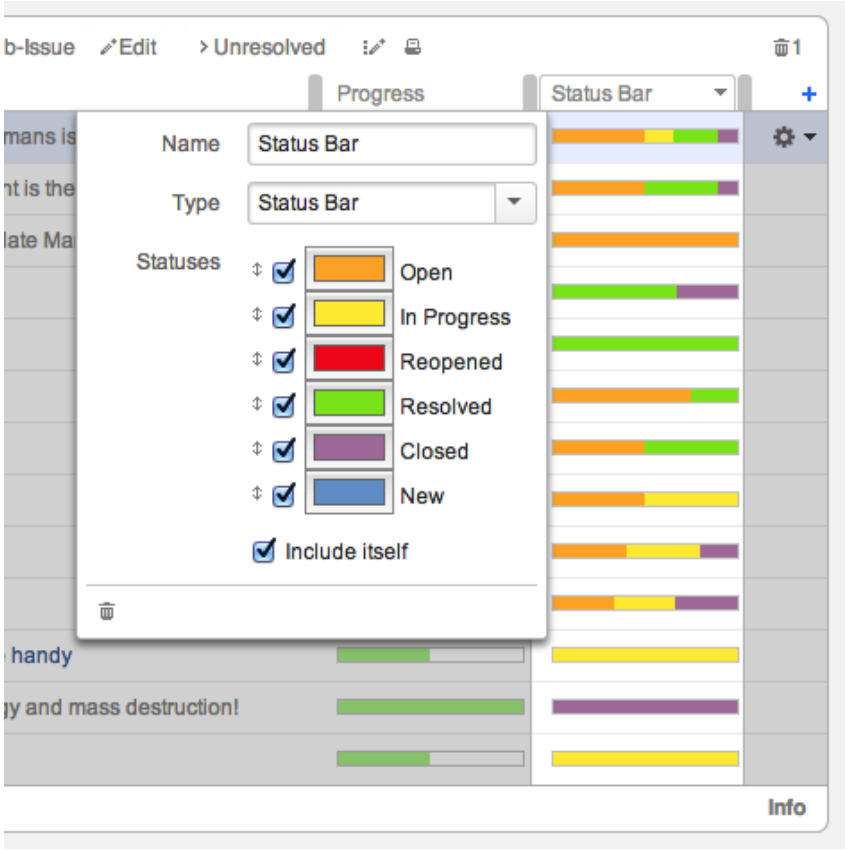
You can add functionality to the Structure Plugin itself by creating a plugin that uses one of the available Structure extension points.

- Creating a New Column Type — Structure 2.5 introduces a set of JavaScript and Java APIs for extending the Structure widget with new column types.
- Creating a New Synchronizer — Structure comes with a number of bundled synchronizers, but you can add another synchronizer to the system, allowing Structure users to install it on structures and run export / import.
- Loading Additional Web Resources For Structure Widget — To include a web resource (such as custom CSS or JavaScript file) on the page every time Structure Widget is displayed, add a structure-widget-extension module to your plugin.

4.3.1 Creating a New Column Type

Structure 2.5 introduces a set of JavaScript and Java APIs for extending the Structure widget with new column types.

In this tutorial we will develop the Status Bar column type, which shows a progress-like bar filled with color stripes, each stripe's color representing a particular issue status, and each stripe's width being proportional to the number of issues having that status in the current issue's subtree. You can download both the compiled plugin and its source code from API Usage Samples.



The Plan

A column type consists of several components. The client-side components are written in JavaScript and have two responsibilities:

- Rendering the cells in the Structure widget.
- Providing the column configuration UI.

The server-side components are written in Java and responsible for:

- Providing the data needed by the client-side part to render the cells.
- Exporting the column into printable HTML and Microsoft Excel formats.

For the Status Bar column we'll need to write code to cover all of the above responsibilities.

In general, however, only the client-side part is strictly necessary. If the data provided by Structure is enough for your column, you can skip the server-side data provider. You can also skip the components related to export, if this functionality is not critical. In this case, you can jump straight to the client-side part, consulting the other chapters as necessary. For the complete treatment, please continue reading from top to bottom.

The Data

Before we begin, let's decide which data we need to pass from the server side to render a status bar. Obviously, the status bar depends on the statuses of all the issues in the given issue's subtree. This suggests that we need to use an Aggregate, and because Structure does not provide such an aggregate out of the box, we'll need to write our own.

Second, the colors and the order of statuses in the status bar are only a presentational matter. If we had a map from status IDs to sub-issue counts in the given issue's subtree, we could count the total number of sub-issues, scale the colored stripes so that they'd fill the whole status bar, and render them in any given order.

Third, the "Include itself" option is somewhat trickier. When it's on, the current issue's status is shown in its status bar, as if there is one more sub-issue. When it's off, the current issue is excluded, and the status bar shows only its sub-issues (on all levels). We could try to implement this on the server side as a separate aggregate, however, this approach has a couple of drawbacks:

- When the user toggles the checkbox, Structure will have to calculate a new aggregate and transfer the results. Because the aggregate values are cached on the server side, and issue data values are cached on the client side, on both sides we'll have increased memory consumption.
- Because of the way the aggregates are calculated and cached on the server side, the aggregate for the option turned off will be somewhat more difficult to write, and use a more complex data structure.

So, we'll do things differently, and use a single, simpler, aggregate, calculating the data with the "Include itself" option turned on. If it's off, we'll adjust the data on the client side. To do that, we'll need another piece of data – the status ID for the current issue, but that can be provided by Structure itself, and the overhead of requiring it is less than that of a separate aggregate.

Status Bar Aggregate

Now that we know which aggregate we need, let's write one. Because the aggregate does not depend on any other aggregate, we'll extend `Aggregate.Independent`. Because it has no internal state or configuration parameters, we'll only need a single instance, which we'll keep in a `public static final` field.

The result type will be `Map<String, Integer>`, where the keys are status IDs and values are issue counts. The calculation is pretty straight-forward:

- First, we create an empty `result` map and put `1` for the current issue's status ID.
- Second, for all child issues (if we have any) we extract their status maps from the cache and iterate over them, incrementing the values in the `result` map.
- Finally, we return an immutable copy of the `result` map.

StatusBarAggregate.java

```
public class StatusBarAggregate extends Aggregate.Independent<Map<String, Integer>> {
  public static final StatusBarAggregate INSTANCE = new StatusBarAggregate();

  @Nullable
  @Override
  public Map<String, Integer> calculate(@NotNull Issue issue, @NotNull LongList children,
@NotNull AggregateResult cache) {
    Map<String, Integer> result = new HashMap<String, Integer>();
    result.put(issue.getStatusObject().getId(), 1);
    for (LongIterator it : children) {
      Map<String, Integer> map = cache.getValue(it.value(), this);
      if (map != null) {
        for (Map.Entry<String, Integer> e : map.entrySet()) {
          Integer count = result.get(e.getKey());
          if (count == null) {
            count = 0;
          }
          result.put(e.getKey(), count + e.getValue());
        }
      }
    }
    return ImmutableMap.copyOf(result);
  }
}
```

Issue Data Providers and Field Specifications

`IssueDataProvider` and `IssueDataField` are responsible for providing the data for the client-side columns.

Each data provider is given a **field specification**, which is a string generated by the client-side column. The specification must be recognizable by the corresponding data provider and contain all the parameters that the provider needs to obtain the data. On the client side these field specifications are used as cache keys.

The Structure widget has a data cache to avoid requesting the same data for the same issues (e.g. when you scroll a long structure down and then back up) if the data didn't change. The client-side column is not limited to a single data field, it can request as many data fields as it needs, and then combine their values. This allows multiple columns or multiple versions of the same column reuse the same data, minimizing network traffic and cache memory consumption. For example, the Icons column uses up to six different data fields, combining their values in the specified order.

The format of the field specification is up to the column type developer, but usually the specification begins with a unique prefix, recognized by the data provider, optionally followed by the representation of its parameters, delimited by colons. Because the aggregate data field for the Status Bar column has no parameters, we'll use a simple `String` constant as the field specification.

Issue data providers are registered as modules in the plugin descriptor, and their instances are created by the JIRA module system. If the data provider "recognizes" the field specification and can serve it, it must return a non-null `IssueDataField` instance. Because our `IssueDataField` implementation is stateless and has no parameters, we can reuse the single `static final` instance, but a configurable data provider could create and return new data field instances for each call. The returned data field instance will then be called once for each issue needed to display the Structure grid (or its visible part). The data field must return the corresponding value as a `String`.

If an `Aggregate` needs to be calculated to obtain the value, it is the responsibility of the `IssueDataProvider` to create or otherwise obtain an `Aggregate` instance and ask Structure to calculate it by calling `prepareAggregate()` on the given `context`. The `IssueDataField` will then be able to obtain the aggregation result by calling `getPreparedAggregates()` on its `context` instance. The provider can also associate arbitrary data with the current request by calling `putObject()` on the `context`. The data will later be available to the `IssueDataField` via the `getObject()` method.

The above should suffice to implement `IssueDataProvider` and `IssueDataField` for the Status Bar column. As with the `Aggregate` above, the implementations are straight-forward. The data provider compares the given field specification with a `String` constant, and if it matches, it asks Structure to calculate the status bar aggregate and returns the canned `IssueDataField` instance. The field obtains the status map for the given `issue` from its `context` and serializes it into a simple format that the client-side column will understand.

StatusBarDataProvider.java

```
public class StatusBarDataProvider implements IssueDataProvider {
  private static final String STATUS_BAR = "com.almworks.jira.structure.sbcolumn:statusbar";

  private static final IssueDataField FIELD_INSTANCE = new IssueDataField() {
    @NotNull
    @Override
    public String getIssueDataValue(@NotNull Issue issue, @NotNull ColumnRenderContext context)
{
      Map<String, Integer> map = context.getPreparedAggregates().getValue(issue,
StatusBarAggregate.INSTANCE);
      if (map != null) {
        StringBuilder sb = new StringBuilder();
        for (Map.Entry<String, Integer> e : map.entrySet()) {
          sb.append(e.getKey()).append(':').append(e.getValue()).append(';');
```

```
        }
        return sb.toString();
      }
      return "";
    }
  };

  @Override
  public IssueDataField getIssueDataField(@NotNull String fieldSpec, @NotNull
ColumnRequestContext context) throws StructureColumnException {
    if (STATUS_BAR.equals(fieldSpec)) {
      context.prepareAggregate(StatusBarAggregate.INSTANCE);
      return FIELD_INSTANCE;
    }
    return null;
  }
}
```

When the data provider is ready, we register it in the plugin descriptor.

atlassian-plugin.xml

```
<structure-issue-data-provider key="idp-sbcolumn" name="issue-data:Status Bar Column Provider"

class="com.almworks.jira.structure.sbcolumn.StatusBarDataProvider"/>
```

Client-Side Column

We now come to the most visible part of the column – the client-side JavaScript code, responsible for rendering the cells of the Structure grid and showing the column configuration UI. Having almost 400 lines of JavaScript, the code is too long to be reproduced in its entirety. We advise you to download the API examples source code from the API Usage Samples page and open `sbcolumn.js` from the `status-bar-column` sample plugin in your favorite editor.

First, we'll take a high-level overview of the API and look at a few common concepts – column specifications, column context, and the metadata. After that we'll discuss each of the API classes and their implementations.

API Overview

The whole API is accessible through the `window.almworks.structure.api` global object. There are a few utility functions and four main classes that the developer needs to extend (by using the `api.subClass()` function) in order to create a fully-functional column. These classes are linked together by the **column specification**, which is a JSON object representing all of the column's parameters. Column specifications are discussed in detail in the following section. Now let's overview the classes and functions.

Class or Function	Description
`api.ColumnType`	The **column type** is the gateway between Structure and your code. The column type is registered with Structure and has the following responsibilities:

Class or Function	Description
	• creating column presets for the "Add Column" menu; • creating the column preset used when switching to your column type from a different type; • creating `Column` and `ColumnConfigurator` instances for given column specifications.
`api.Column`	The **column** is responsible for value rendering. It creates the HTML for the widget cells and controls the column's name and width. It can require issue data from the data providers.
`api.ColumnConfigurator`	The **configurator** is responsible for the column configuration panel as a whole. Its most important task is to create `ColumnOption` instances.
`api.ColumnOption`	The **option** is the workhorse of the configuration UI, corresponding to a single "row" of the configuration panel. It is responsible for creating the input elements and routing changes between them and the specification.
`api.registerColumnType(columnType, columnKey)`	Registers a column type with the Structure, making it responsible for handling the given column key (see below).
`api.registerColumnGroup(parameters)`	Registers a new group in the Add Column menu.

Column Specifications

A **column specification** is a JSON object representing the complete configuration of a Structure widget column. Column specifications are stored as parts of view specifications. Each `Column`, `ColumnConfigurator` and `ColumnOption` instance has its own current specification, accessed via `this.spec`. A `ColumnType` is given a column specification when Structure wants it to create a `Column` or a `ColumnConfigurator`. `ColumnType` also creates column specifications for column presets. Finally, column specifications are passed to the export renderer providers on the server side (see below).

Here is an example of a Status Bar column specification.

```
{ "csid":   "7",
  "key":    "com.almworks.jira.structure.sbcolumn",
  "name":   "Status Bar",
  "params": {
    "statuses":      ["1", "3", "4", "5", "6", "10000"],
    "colors":        ["#fcaf3e", "#fce94f", "#ef2929", "#8ae234", "#ad7fa8", "#729fcf"],
    "includeItself": true }}
```

Key	Description
`csid`	The **CSID** ("column sequential ID") is a string that uniquely identifies a column within a view. CSIDs are assigned and managed by Structure, and should not bother you as a column developer. **Do not change a column's CSID!**
`key`	The **key** is a string identifying the column type. Structure uses the key to decide which `ColumnType` or `ExportRendererProvider` to use for a particular column. The key is required.
`name`	The column **name** is shown in the column header. The name is often omitted from the specification, in which case a default name is generated for the column.
`params`	This is a JSON object containing the column's parameters. The layout of this object is up to the column developer. In the example we see two parallel arrays for the selected status IDs and their colors, and a `boolean` for the "Include itself" option.

The Column Context

A **column context** is a JavaScript object providing various kinds of information about the environment, in which columns and their configurators operate. It is not to be confused with the somewhat similar in purpose, but unrelated `ColumnContext` on the server side. When Structure makes requests to the `ColumnType`, it passes the context as a parameter. Each `Column`, `ColumnConfigurator` or `ColumnOption` instance has its own current context, accessed via `this.context`. The table below describes the methods of the column context.

Method	Description
`structure.isPrimaryPanel()`	Returns `true` if the column belongs (or will belong, for presets) to the primary panel of the Structure widget.
`structure.isSecondaryPanel()`	Returns `true` if the column belongs (or will belong, for presets) to a secondary panel of the Structure widget.
`structure.isStructureBoard()`	Returns `true` if the current widget is on the Structure Board page.
`structure.isIssuePage()`	Returns `true` if the current widget is in the Structure section of an issue page.
`structure.isGadget()`	Returns `true` if the current widget is embedded in a Structure gadget.
`structure.isLocalGadget()`	Returns `true` if the current widget is embedded in a **local** Structure gadget (i.e. a gadget provided and rendered by the same server).
`structure.isRemoteGadget()`	

Method	Description
	Returns `true` if the current widget is embedded in a **remote** Structure gadget (i.e. a gadget provided and rendered by different servers).
`structure.isGreenHopperTab()`	Returns `true` if the current gadget is in the Structure section of an Agile (GreenHopper) board.
`structure.isProjectPage()`	Returns `true` if the current gadget is in the Structure tab of a project page.
`structure.isComponentPage()`	Returns `true` if the current gadget is in the Structure tab of a component page.
`structure.isVersionPage()`	Returns `true` if the current gadget is in the Structure tab of a version page.
`structure.isPCVPage()`	Returns `true` if the current gadget is in the Structure tab of a project, component, or version page.
`jira.getAllIssueFields()`	Returns an array of JSON objects representing available JIRA issue fields.
`jira.getIssueFieldById(fieldId)`	Returns a JSON object representing the JIRA issue field with the given ID, or `undefined` if there is no such field.
`getMetadata(key)`	Returns the metadata object associated with the given `key`. See the section below for the description of metadata.

We will show the Status Bar column only on the primary panel of a widget, so we'll use `context.structure.isPrimaryPanel()` to check for that. The column will also make use of `context.getMetadata()`.

Requesting and Using Metadata

Metadata, in the context of the column API, is any data needed by column types, columns, and configurators to to do their duties, except for the issue data served by the issue data providers. For example, the Status Bar column needs to know the IDs and names of all the issue statuses in order to render tooltips and create presets – this is metadata. Structure provides some metadata by default – the `getAllIssueFields()` and `getIssueFieldById()` methods of the column context are examples, but you can load more via AJAX by issuing **metadata requests**.

Metadata is requested by overriding one or more of the methods in `ColumnType`, `Column`, and `ColumnConfigurator` classes. Let's look at an example from the Status Bar column type:

sbcolumn.js

```
getMetadataRequests: function() {
  return {
    status: {
```

```
      url: baseUrl + '/rest/api/2/status',
      cacheable: true,
      extract: function(response) {
        var result = { order: [], names: {} };
        if ($.isArray(response)) {
          response.forEach(function(status) {
            result.order.push(status.id);
            result.names[status.id] = status.name;
          });
        }
        return result;
      }
    }
  };
}
```

The method is supposed to return a JavaScript object. Each key in that object will become a metadata key for obtaining the corresponding result from the column context. In this example, the status-related metadata object will be obtained by calling `context.getMetadata('status')`.

The values in the returned object are request specifications. Let's look at the request properties:

- The `url` property is the URL to be requested. Here we call a JIRA REST API method that returns all available issue statuses. **Don't forget the JIRA base URL!**
- The `cacheable` property is an opt-in mechanism for response caching. If a metadata request is cacheable, and this URL has already been requested (e.g. by a different column type), the previous response will be used instead of making a new AJAX request. You should **declare your requests cacheable whenever possible** to conserve traffic and improve responsiveness.
- The `extract` property is the function that receives the response and produces the value stored in the metadata map. If omitted, the response is stored unchanged. In the example, we convert the resulting array of JSON objects into an array of status IDs and a map from status IDs to status names.
- You can add any other properties supported by `jQuery.ajax()` to the request specification. Remember, though, that the jQuery success and error handlers will not be called for cacheable requests if a cached response is used.

Different metadata may be required for different operations. Therefore, there are several methods in the API that you can override to request metadata:

- A column type may request metadata to be able to:
  - create column presets – `ColumnType.getPresetMetadataRequests()`;
  - create columns from specifications – `ColumnType.getColumnMetadataRequests()`;
  - create configurators from specifications – `ColumnType.getConfigMetadataRequests()`;
  - do all of the above – `ColumnType.getMetadataRequests()`, the "catch-all" method.
- A column may need metadata to render its values – `Column.getMetadataRequests()`.
- A configurator may need metadata to set up the UI – `ColumnConfigurator.getMetadataRequests()`.

Please note that the corresponding type-level metadata is also available to the columns and configurators created by the type. So, for example, there is no need to issue *the same* requests in both

`ColumnType.getColumnMetadataRequests()` and `Column.getMetadataRequests()`, the former alone will suffice.

Structure will delay loading the metadata as long as possible. For example:

- the metadata for a column will not be loaded unless there is a column in the widget that needs it;
- the metadata for creating column presets will not be loaded until the user clicks "Add Column" or "Edit Column" icons;
- and so on.

Structure guarantees that the metadata request will be completed by the time it calls your type, column, and configurator methods (obviously, except for the `getMetadataRequests()` methods themselves). If the requests succeed, the metadata will be available in the column context. If they fail, the corresponding metadata will be `undefined`, but the methods will still be called, and they should not fail in that case.

Column

The `api.Column` class is responsible for rendering the cells of the Structure grid. Please refer to the Column class reference for the list of methods that you can override. The `StatusBarColumn` class in `sbcolumn.js` overrides four methods.

`getDefaultName()` simply returns a localized string as the column name when the name is not present in the column specification. A more involved column could use its specification, context, or metadata to determine the default column name.

`canShrinkWhenNoSpace()` allows Structure to make the column narrower than its minimum width when the widget is very low on horizontal space. Because we do not override any other sizing-related methods, the column will be resizable, with the default and minimum width of 120 and 27 pixels, respectively. Autosizing will not be applied to it, because there is no variable-size content, so autosizing makes no sense.

`getRequiredFields()` always requests the status bar aggregate data from `StatusBarDataProvider`. If the "Include itself" option is off, it additionally requests the status ID of the current issue, which is provided by Structure as `jira:raw:status`.

`getCellViewHtml()` returns the actual HTML for the cells. It obtains the serialized status bar map from the `renderParameters`, deserializes it, adjusts for the "Include itself" option, if necessary, distributes the full status bar width of 100% among the selected statuses according to their issue counts, and finally generates and returns the status bar HTML code as a string. Please refer to the source code for the implementation details.

Please note, that for simple columns, displaying textual information, we advise you to override `getCellValueHtml()` instead, and let Structure take care of the boilerplate HTML surrounding your value. However, since we want the Status Bar to look similar to Structure's Progress Bar, we need to override a higher-level method and mimic the Progress Bar HTML layout.

ColumnConfigurator

The `api.ColumnConfigurator` class is responsible for the column configuration UI. Because most of the work is delegated to `ColumnOption` instances (see below), the configurators themselves are usually quite simple. Let's look at `StatusBarConfigurator` in its entirety.

sbcolumn.js

```
var StatusBarConfigurator = api.subClass('StatusBarConfigurator', api.ColumnConfigurator, {
  init: function() {
    this.spec.key = COLUMN_KEY;
    this.spec.params || (this.spec.params = {});
  },
  getColumnTypeName: function() {
    return AJS.I18n.getText("sbcolumn.name");
  },
  getGroupKey: function() {
    return GROUP_KEY;
  },
  getOptions: function() {
    return [new StatusesOption({ configurator: this }), new IncludeItselfOption({ configurator:
this })];
  }
});
```

The constructor, `init()` simply sanitizes the current column specification.

`getColumnTypeName()` returns the human-readable name for the column type. This name is used in the "Type" drop-down of the column configuration panel. You can also override `getDefaultColumnName()` to generate column names if the type name cannot always be used as the default column name.

`getGroupKey()` returns the key of the group in the "Add Column" menu that will contain this preset. See the sections on `ColumnType` and column groups below.

`getOptions()` creates and returns an array of `ColumnOption` instances that create input controls for the column configuration panel and route events. Please note how the configurator instance is passed to each option's constructor – this is crucial. The order of the options in the resulting array is also important – the rows of the configuration panel will be created in that order.

Although the methods of `StatusBarConfigurator` always return the same values, this is not a requirement. The result of any of the methods can depend on the current column specification (`this.spec`) and metadata.

ColumnOption

Each `api.ColumnOption` instance is responsible for editing a single logical "part" of the column specification, and corresponds to a single "row" of the column configuration panel. The option creates the actual input elements and sets up event handlers to transfer the values between the inputs and its column specification. An option can hide itself if it's not applicable to the current specification. Also, each option can prohibit saving the column configuration if it considers the current specification invalid – see `isInputValid()` method in the class reference.

Status Bar column has two options:

- `StatusesOption` is responsible for status selection, colors, and ordering. It "owns" the `statuses` and `colors` arrays of a Status Bar column specification. This option is somewhat more involved than the next one, but you can still refer to its source code in `sbcolumn.js`.
- `IncludeItselfOption` is responsible for the "Include itself" checkbox and "owns" the `includeItself` specification parameter. This is one of the simplest options imaginable, so we'll look at its code in detail.

sbcolumn.js

```
var IncludeItselfOption = api.subClass('IncludeItselfOption', api.ColumnOption, {
  createInput: function(div$) {
    this.checkbox$ = div$.append(
        AJS.template('<div class="checkbox"><label><input
type="checkbox"> {label}</label></div>')
          .fill({ label: AJS.I18n.getText("sbcolumn.include-itself") })
          .toString()).find('input');
    var params = this.spec.params;
    this.checkbox$.on('change', function() {
      if ($(this).is(':checked')) {
        params.includeItself = true;
      } else {
        delete params.includeItself;
      }
      div$.trigger('notify');
    });
  },
  notify: function() {
    this.checkbox$.prop('checked', !!this.spec.params.includeItself);
    return true;
  }
});
```

Because the option class specifies no `title` and doesn't override `createLabel()`, there is no label to the left of the checkbox.

The `createInput()` method creates the checkbox and sets up event handling. It is passed a jQuery object to append the input elements to.

Please note that Structure column configuration panels use the AUI Forms HTML layout (with modified CSS styles). You should use the same layout in your HTML code to make your options look consistent with Structure's. In the example above, the checkbox is wrapped in a `<div class="checkbox">` element to comply with AUI Forms.

Also note how the `change` event handler of the checkbox modifies the current specification parameters and always triggers a `notify` event on the provided jQuery object. These are the crucial parts of the option contract.

The `notify()` method is called whenever the current specification changes. Its job is to transfer the data in the opposite direction – from the specification to the input elements. This method also decides whether the

option is applicable – if it returns a "falsy" value, the option's row on the configuration panel is hidden from the user.

ColumnType

The `api.ColumnType` class is the main entry point used by the Structure plugin to call your client-side column code. A column type instance creates column presets, columns, and configurators. To find the complete source code for the Status Bar column type, please open `sbcolumn.js` from the API example sources in your favorite editor and scroll to the `StatusBarType` class definition.

The `getMetadataRequestsMethod()` declares the column-level metadata request to load the available issue statuses from JIRA. See Requesting and Using Metadata above for details.

The `createSwitchTypePreset()` method creates a single column specification, which is used as a preset when the user selects our type in the "Type" drop-down on the column configuration panel.

Note the call to the `isAvailable()` function that checks that the preset is needed for the primary panel and that the status metadata is indeed available. If that check fails, the method returns `null`, making it impossible to switch to the Status Bar column type. You can try it yourself – open the Search Result secondary panel, add any column to it and try to change its column type. You should see that the Status Bar type is not available.

The switching preset doesn't have to be fully configured, because the configuration panel is already open when it's used. However, because the Status Bar column configuration is quite complex, we make an extra effort and pre-populate the preset with all the known statuses and some default colors for them. This way the user will quickly see what a status bar looks like without having to configure anything at all. This tactic can be useful for other columns with a lot of parameters.

The `createAddColumnPresets()` method creates an array of column specifications that will be used as presets in the "Add Column" menu. Unlike the "switch" preset above, these presets must be completely configured. Like `createSwitchTypePreset()`, this method calls `isAvailable()` first, so a Status Bar column cannot be added to a secondary Structure panel.

Because the "Add Column" menu is the first place where the user discovers your column type, it would be best if your presets are interesting and cover the whole range of the type's functionality. It's not easy to be creative with the Status Bar column though, unless we know the semantics of statuses, which can be arbitrary. So, for simplicity `StatusBarType` adds only a single preset to the "Add Column" menu, reusing the "switch" preset, which is fully configured.

Besides the usual `key`, `name`, and `params`, the "add" presets can have two special properties:

- `presetName` is a string that specifies the name of the preset in the "Add Column" menu. This name will be used *only in the menu*, the added column will have either the `name` from the specification or the default name generated for it. If omitted, the column name will be used as the preset name.
- `shouldOpenConfigurator` – if this flag is set to `true`, the column configuration panel will open immediately after adding the column with this preset. This can be used to create a "Custom..." kind of preset that lets the user explore the available options.

The `createColumn()` and `createConfigurator()` methods return a `Column` or a `ColumnConfigurator` for the given specification, respectively. The methods are similar – they check whether the type is available and the given specification is valid, and if both checks succeed, they instantiate the appropriate subclass. Please note how the column context and the specification are passed to the constructors, this is crucial.

Finally, at the end of the script we instantiate and register our column type, making it available to Structure:

sbcolumn.js

```
api.registerColumnType(new StatusBarType(), COLUMN_KEY);
```

Structure will use our column type instance to handle the columns with the given key. You can also pass an array of keys as the second argument, to associate your type with more than one column key.

Column Groups

Column groups are used to organize column presets in the "Add Column" menu. Each group has a string key and a human-readable name. Column configurator's `getGroupKey()` method should return the appropriate group key for its preset specification.

Structure specifies four column groups for its built-in columns – `fields`, `icons`, `totals`, and `progress`. For the Status Bar column we will register a separate column group:

sbcolumn.js

```
api.registerColumnGroup({ groupKey: GROUP_KEY, title: AJS.I18n.getText("sbcolumn.name"), order:
1000 });
```

The `order` parameter determines the position of the group within the menu. The higher the order, the lower the group will be. Structure's predefined groups have order between 100 and 400, inclusive.

Widget Extension

You need to register your JavaScript and CSS code as a web resource in the plugin descriptor. The Status Bar column has no CSS of its own, and all of its JavaScript code is in a single file, `sbcolumn.js`. Because we use the Structure JavaScript API and the `AJS.template()` function from the Atlassian API, we need to declare two dependencies. We also declare a resource transformation to make `AJS.I18n.getText()` calls work.

atlassian-plugin.xml

```
<web-resource key="wr-sbcolumn" name="web-resource:Status Bar Column">
  <dependency>com.atlassian.auiplugin:ajs</dependency>
  <dependency>com.almworks.jira.structure:widget</dependency>
  <transformation extension="js">
    <transformer key="jsI18n"/>
  </transformation>
  <resource type="download" name="sbcolumn.js" location="js/sbcolumn/sbcolumn.js"/>
</web-resource>
```

Finally, we add our web resource to a Structure widget extension:

atlassian-plugin.xml

```
<structure-widget-extension key="we-sbcolumn" name="widget-extension:Status Bar Column">
  <web-resource>com.almworks.jira.structure.sbcolumn:wr-sbcolumn</web-resource>
</structure-widget-extension>
```

Because of the limitations of the web resource context mechanism, it is impossible to add a web resource to all the pages containing the Structure widget by adding a single context to it. Therefore, structure adds a special module type, called the **widget extension** that lets you add web resources to the widget. Technically, a widget extension is itself a kind of web resource that depends on the specified resources and gets added to several resource contexts. You can combine multiple web resources in a single widget extension.

Export Renderers

Any structure (or its part, defined by a search query) can be exported into printable HTML and Microsoft Excel formats. Exporting is different from rendering the Structure widget in several aspects:

- It is entirely a server-side task, so the code is written in Java.
- The data needed for exporting need not be transferred over the network and cached.
- The export result need not be updated as the exported issues or structure change.
- There are two distinct formats, or media, that are quite different from each other. More formats may be added in the future.

It is because of these differences, that the exporting architecture and APIs are distinct from their widget rendering counterparts, being simpler in some aspects and more complex in others, while quite similar overall, sometimes making it non-trivial to avoid "repeating yourself".

Please refer to the javadoc for an overview of the export API and SPI. In short, to export a column, you need to write and register an **export renderer provider**, that will recognize the column specification and return and **export renderer** instance for the given column and export format. The returned renderer will then be given an **export column** instance to configure and **export cell** instances to render the values. The **column context** and **issue row** instances will provide all the data, including the aggregate values.

Speaking of the interfaces that must be implemented, `ExportRendererProvider` is analogous to `IssueDataProvider`, and `ExportRenderer` is a mixture of `IssueDataField` and the client-side `Column`.

Export Strategies

The main difficulty with export is having different output formats with different features. For example, if you have a method for converting a value to HTML, you could reuse it for the printable HTML export. But when exporting to Excel, HTML support is very limited, and if your values correspond to one of Excel's data types, e.g. date, you need to set an appropriate column style. On the other hand, if you have a simple plain-text column, the format doesn't matter – you can have a single export renderer that calls `setText()` on any type of cell.

The export SPI is flexible, and allows you to use different strategies for different column types. There are three basic kinds of export renderer providers.

- A **specific renderer provider** declares which particular export format it supports in the plugin descriptor. It is parameterized with the expected column and cell types, and returns similarly parameterized renderers, that use format-specific methods.
- A **generic renderer provider** does not declare an export format in the plugin descriptor, so its priority is lower than that of a specific renderer provider. It returns generic renderers, that only call the methods of the basic `ExportCell` and `ExportColumn` interfaces. Though limited, such a provider will work for any other export format that may be added in the future.
- A **multi-format renderer provider** either declares no supported formats (like a generic provider), or declares multiple supported formats. It is not parameterized with specific cell and column types, but it keeps track of the current export format, and its renderers may call format-specific methods by casting the given column and cell instances to the appropriate types. Though more difficult to write, a multi-format provider can combine the benefits of generic and specific providers and help avoid code duplication.

Exploring the extremes, we will create two export renderer providers for the Status Bar column. The first will be a generic provider, that will present the data as plain text instead of drawing a progress bar. The second one will be an advanced Excel provider that will use the underlying low-level Apache POI API to draw pseudo-graphic progress bars in Excel cells.

Generic Renderer Provider

The `StatusBarRendererProvider` class in the `status-bar-column` example plugin source contains both the generic provider and its renderer. The code is quite long, but that's mostly due to defensive checks and the general verbosity of Java. The operation of both the provider and the renderer is quite straight-forward.

The provider's `getColumnRenderer()` method does the following:

- Checks that the given column specification indeed represents a Status Bar column, just in case.
- Obtains the column name form the specification, generating a default name if there is none.
- Extracts the `statuses` array and the `includeItself` flag from the specification parameters. These are needed for rendering.
- Asks Structure to calculate the Status Bar aggregate for the exported structure.
- Creates and returns an instance of the `StatusBarRenderer` inner class, passing it the column name and parameters.

The renderer's `configureColumn()` method sets the column name by calling `setText()` on the given column's header cell.

The renderer's `renderCell()` method does the following:

- Obtains the aggregation result from the context and the status bar data map from the result.
- Adjusts the data if the the "Include itself" option is off, by decrementing the issue count for the current issue's status.

- Iterates over the selected statuses, adding each non-zero sub-issue count and the corresponding status name to a `StringBuilder`.
- If the resulting value is not empty, calls `setText()` on the given cell.

Here is the module declaration for the generic renderer provider. Note that it specifies the column key, but no export format.

atlassian-plugin.xml

```
<structure-export-renderer-provider key="erp-sbcolumn" name="export-renderer:Status Bar Column
Provider"

class="com.almworks.jira.structure.sbcolumn.StatusBarRendererProvider">
  <column-key>com.almworks.jira.structure.sbcolumn</column-key>
</structure-export-renderer-provider>
```

Advanced Excel Renderer Provider

The `StatusBarExcelProvider` class contains the advanced Excel renderer and the corresponding provider.

The provider's `getColumnRenderer()` method is very similar to the generic provider's, with two additions:

- it checks that the export format is indeed `MS_EXCEL`;
- it also extracts the `colors` array from the specification parameters, as the renderer will use those (or similar) colors for the progress bar.

The renderer's `configureColumn()` method is the same as the generic version. The `renderCell()` method begins in a similar way, by extracting the data map and adjusting it for the "Include itself" option, if needed.

The interesting part is the actual rendering. The pseudo-graphic "progress bar" that the renderer creates is a string of 30 "pipe" characters, split into colored stripes with lengths proportional to issue counts. `ExcelCell` provides no support for rich text formatting (besides `setRichTextFromHtml()`, which is not up to the task), but we can access the lower-level API, Apache POI HSSF, by obtaining the underlying POI objects from `ColumnContext.getObject()` using the keys from `ColumnContextKeys.Excel`.

The code that distributes the 30 characters among the stripes is ported from `sbcolumn.js`. To completely understand how the rich text part works, you'll need some knowledge of the POI HSSF API, which is quite complex and outside of the scope of this document. Please refer to the POI documentation and the `StatusBarExcelProvider` source code for more information.

The module declaration for the Excel renderer provider is given below. Note that it specifies both a column key and an export format, thus overriding the generic provider for the Excel format.

atlassian-plugin.xml

```
<structure-export-renderer-provider key="erp-sbcolumn-excel" name="export-renderer:Status Bar
Column Excel Provider"
```

```
class="com.almworks.jira.structure.sbcolumn.StatusBarExcelProvider">
  <column-key>com.almworks.jira.structure.sbcolumn</column-key>
  <export-format>ms-excel</export-format>
</structure-export-renderer-provider>
```

4.3.2 Creating a New Synchronizer

Structure comes with a number of bundled synchronizers, but you can add another synchronizer to the system, allowing Structure users to install it on structures and run export / import.

Implement StructureSynchronizer

Create your implementation of StructureSynchronizer interface.

Use AbstractSynchronizer or AbstractIssueListeningSynchronizer as the base class.

Define structure-synchronizer Module

Add Synchronizer Module module to your `atlassian-plugin.xml`, referring to your implementation of the `StructureSynchronizer`.

Test Thoroughly

Test how your synchronizer works when other synchronizers are also installed onto the same structure.

Sample Project

This project can be used to bootstrap writing your own synchronizer. It compiles into a working plugin, which does not do anything except writing to console at the times the synchronizer would do some work.

Name	Version	Date
foo-synchronizer-0.1.zip	2	2012-07-09 21:32

4.3.3 Loading Additional Web Resources For Structure Widget

To include a web resource (such as custom CSS or JavaScript file) on the page every time Structure Widget is displayed, add a `structure-widget-extension` module to your plugin.

Use cases:

1. You create your own custom field and would like it to be editable in the Structure grid. The field is powered by additional JavaScript or CSS, which should be loaded on the page that displays structure.

2. You create your own column type. You'll need to use the Structure JavaScript API and register the web resource with your JavaScript code as a widget extension.

Widget Extensions

Structure 2.5 introduces a new module type – `structure-widget-extension`. Widget extensions contain references to the web resource modules that will be included on all pages where the Structure widget is present. Creating a widget extension is the preferred method of adding JavaScript and CSS code to the Structure widget since Structure 2.5.

Sample snippet from `atlassian-plugin.xml`:

```
<structure-widget-extension key="we-sbcolumn" name="widget-extension:Status Bar Column">
  <web-resource>com.almworks.jira.structure.sbcolumn:wr-sbcolumn</web-resource>
</structure-widget-extension>
```

structure.widget Web Resource Context

Before Structure 2.5, the only way to add JavaScript or CSS to the Structure widget was to add a web resource to the `structure.widget` context. Note, however, that due to Atlassian API limitations, context-provided web resources may not be loaded on **all** pages with the Structure widget. The notable exceptions are the Structure Gadget and the JIRA Agile (GreenHopper) Structure panel.

In Structure 2.5+ the `structure.widget` resource context is still present, but its resources are loaded only for the Structure Board and the Structure Gadget. We recommend you to use widget extensions instead.

Sample snippet from `atlassian-plugin.xml`:

```
<web-resource key="custom-field-resource" name="My Custom Field Web Resource">
  <context>structure.widget</context>
  <resource type="download" name="custom-field-resource.js"
location="js/myplugin/custom-field-resource.js"/>
</web-resource>
```

structure.printable Web Resource Context

Web resources from the `structure.printable` context are loaded on the printable page. While this context is not directly associated with the Structure widget, you'll need it to add a new column type that uses custom CSS in its printable representation.

4.4 Accessing Structure Data Remotely

Structure plugin provides REST API, which is primarily used by the structure widget. The same API can be used to access the hierarchical data remotely from an automation script or another user agent application.

See details in the REST API Reference.

4.5 Reference

4.5.1 Structure Developer Reference

- Structure Java API Reference
  - Structure API Versions
- Structure Plugin Module Types
  - Synchronizer Module
  - Widget Extension Module
  - Issue Data Provider Module
  - Export Renderer Provider Module
- Structure REST API Reference
  - Structure Resource
  - Forest Resource - GET
  - Forest Resource - POST
- Structure JavaScript API Reference
  - JavaScript API Functions
  - JavaScript API Classes
    - Column Class
    - ColumnConfigurator Class
    - ColumnOption Class
    - ColumnType Class
- Web Resource Contexts

4.5.2 Structure Java API Reference

> Structure API Reference for the latest version: http://almworks.com/structure/javadoc/latest

You can download javadocs from the Maven repositories into your IDE.

Check out information about Structure API Versions to select the correct API artifact, and you can also download Javadoc JARs there.

Structure API Versions

Current Versions

Version	Supported JIRA Versions	Supported Structure Versions	OSGi Import Version	Release Date
8.5.0 Javadocs	JIRA 6.0+	2.8.0+	"[8.5,9)"	2014-07-01

Version	Supported JIRA Versions	Supported Structure Versions	OSGi Import Version	Release Date
8.4.0 Javadocs	JIRA 6.0+	2.7.0+	"[8.4,9)"	2014-02-26
8.3.0 Javadocs	JIRA 6.0+	2.6.0+	"[8.3,9)"	2014-01-24
7.7.0 Javadocs	JIRA 5.0.1 – 5.2+	2.6.0.jira5	"[7.7,8)"	2014-01-24
8.2.0 Javadocs	JIRA 6.0+	2.5.0+	"[8.2,9)"	2013-09-25
7.6.0 Javadocs	JIRA 5.0.1 – 5.2+	2.5.0.jira5	"[7.6,8)"	2013-09-25
8.1.0 Javadocs	JIRA 6.0+	2.4.0+	"[8.1,9)"	2013-06-14
7.5.0 Javadocs	JIRA 5.0.1 – 5.2+	2.4.0.jira5	"[7.5,8)"	2013-06-14
8.0.0 Javadocs	JIRA 6.0+	2.3.0+	"[8.0,9)"	2013-05-18
7.4.0 Javadocs	JIRA 5.0 – 5.2+	2.3.0.jira5	"[7.4,8)"	2013-05-18
7.3.0 Javadocs	JIRA 5.0 – 5.2+	2.1 – 2.2.1, 2.3.0.jira5	"[7.3,8)"	2013-02-15
7.2.0 Javadocs	JIRA 5.0 – 5.2+	2.0 – 2.2.1, 2.3.0.jira5	"[7.2,8)"	2012-11-19
7.1.0 Javadocs	JIRA 5.0 – 5.2+	1.7 – 2.2.1, 2.3.0.jira5	"[7.1,8)"	2012-07-11
6.2.0 Javadocs	JIRA 4.4 – 4.4.5	2.0.jira44	"[6.2,7)"	2012-07-11
6.1.0 Javadocs	JIRA 4.4 – 4.4.5	1.7.jira44 – 2.0.jira44	"[6.1,7)"	2012-07-11
6.0.0 Javadocs	JIRA 4.3 – 4.4.5	1.6.jira43 – 2.0.jira44	"[6.0,7)"	2012-05-11
3.0.0 Javadocs	JIRA 4.2*	1.4.jira42	"[3.0,4)"	2012-01-08

Latest Version Javadocs

To see how to include the API jar into your project dependencies, read about Accessing Structure from Your Plugin.

Version Compatibility

Versioning of the API artifact follows these generally accepted rules:

- Major version is increased when the client code – your code – might not compile with the new version.
- Minor version is increased when new methods are added to the API (so your code might break if you downgrade to a lower minor version).
- Micro version is changed when there's no impact on the compatibility.

Getting Versions

The API jars can be downloaded from the public Maven repositories. This is the recommended way.

If you can't download API jars from Maven repository for any reason, you can download them from this page and install into your local Maven repository:

```
mvn install:install-file -DgroupId=com.almworks.jira.structure -DartifactId=structure-api
-Dversion=7.2.0 -Dpackaging=jar -Dfile=structure-api-7.2.0.jar
```

Name	Version	Date
structure-api-7.2.0-javadoc.jar	1	2012-11-19 02:17
structure-api-7.2.0-sources.jar	1	2012-11-19 02:17
structure-api-7.2.0.jar	1	2012-11-19 02:17
structure-api-7.3.0-javadoc.jar	1	2013-02-15 19:03
structure-api-7.3.0-sources.jar	1	2013-02-15 19:03
structure-api-7.3.0.jar	1	2013-02-15 19:03
structure-api-7.4.0-javadoc.jar	1	2013-05-18 22:13
structure-api-7.4.0-sources.jar	1	2013-05-18 22:13
structure-api-7.4.0.jar	1	2013-05-18 22:13
structure-api-7.5.0-javadoc.jar	1	2013-06-14 16:43
structure-api-7.5.0-sources.jar	1	2013-06-14 16:43
structure-api-7.5.0.jar	1	2013-06-14 16:43
structure-api-7.6.0-javadoc.jar	1	2013-10-29 13:59
structure-api-7.6.0-sources.jar	1	2013-10-29 13:59
structure-api-7.6.0.jar	1	2013-10-29 13:59

Name	Version	Date
structure-api-8.0.0-javadoc.jar	1	2013-05-18 22:13
structure-api-8.0.0-sources.jar	1	2013-05-18 22:13
structure-api-8.0.0.jar	1	2013-05-18 22:13
structure-api-8.1.0-javadoc.jar	1	2013-06-14 16:43
structure-api-8.1.0-sources.jar	1	2013-06-14 16:43
structure-api-8.1.0.jar	1	2013-06-14 16:43
structure-api-8.2.0-javadoc.jar	1	2013-10-29 13:59
structure-api-8.2.0-sources.jar	1	2013-10-29 13:59
structure-api-8.2.0.jar	1	2013-10-29 13:59
structure-api-8.3.0-javadoc.jar	1	2014-02-25 14:42
structure-api-8.3.0-sources.jar	1	2014-02-25 14:42
structure-api-8.3.0.jar	1	2014-02-25 14:42
structure-api-8.4.0-javadoc.jar	1	2014-02-25 14:43
structure-api-8.4.0-sources.jar	1	2014-02-25 14:43
structure-api-8.4.0.jar	1	2014-02-25 14:43

4.5.3 Structure Plugin Module Types

The following module types are added by the Structure plugin:

- `structure-synchronizer` defines a new synchronization type
- `structure-widget-extension` lets you add Javascript and CSS extensions to the Structure widget
- `structure-issue-data-provider` lets you provide the data for new column types in the Structure widget
- `structure-export-renderer-provider` lets you export new column types to printable HTML and Excel files

Synchronizer Module

Synchronizer module allows you to plug additional synchronizers into Structure.

Module description sample

Here's a template of a synchronizer module declaration, and explanation of the parameters follows.

```
<structure-synchronizer key="module-key" order="100"
                        class="com.company.your.plugin.sync.SyncClass">
    <label key="label.i18n.key">Name of Synchronizer</label>
    <description key="description.i18n.key">Description of Synchronizer</description>
    <rules key="rules.i18n.key">Large text to be shown at the top of synchronizer's
configuration page.</rules>

    <resource type="velocity" name="form" location="/templates/myplugin/sync-form.vm"/>

    <inbound>
      <resync-description key="inbound.resync.i18n.key"/>
    </inbound>
    <outbound>
      <resync-description key="outbound.resync.i18n.key"/>
    </outbound>
  </structure-synchronizer>
```

Element	Required	Description
structure-synchronizer	Yes	The module descriptor
structure-synchronizer/@key	Yes	Unique module key within the plugin
structure-synchronizer/@order	Yes	Order of the synchronizer among other synchronizers, whenever a list of synchronizers is present
structure-synchronizer/@class	Yes	The class that implements the synchronizer. Must implement StructureSynchronizer - better extend AbstractSynchronizer or AbstractIssueListeningSynchronizer
structure-synchronizer/label	Yes	The name of the synchronizer
structure-synchronizer/description	No	Description of the synchronizer
structure-synchronizer/rules	No	Possibly large text, that is shown at the top of synchronizer configuration page.
structure-synchronizer/resource[@name="form"]	Yes	A velocity template that contains form for the synchronizer parameters.
structure-synchronizer/inbound	Either inbound, outbound or both	If present, it means that Resync can run "from outside into Structure"
structure-synchronizer/inbound/resync-description		
structure-synchronizer/outbound	Either inbound,	If present, it means that Resync can run "from Structure to outside"

Element	Required	Description
	outbound or both	
structure-synchronizer/outbound/resync-description		

Widget Extension Module

Widget extension module lets you add Javascript and CSS resources to the Structure widget. Your extensions will be loaded on all pages where the widget is shown. You can use the Structure JavaScript API and the Atlassian APIs in you JavaScript extensions.

Widget extension example

```
<web-resource key="wr-sbcolumn" name="web-resource:Status Bar Column">
  <resource type="download" name="sbcolumn.js" location="js/sbcolumn/sbcolumn.js"/>
</web-resource>

<structure-widget-extension key="we-sbcolumn" name="widget-extension:Status Bar Column">
  <web-resource>com.almworks.jira.structure.sbcolumn:wr-sbcolumn</web-resource>
</structure-widget-extension>
```

Element	Description
structure-widget-extension	The module descriptor.
structure-widget-extension/@key	The unique identifier of the plugin module. Required.
structure-widget-extension/@name	The human-readable name of the plugin module.
structure-widget-extension/web-resource	The complete key of the web resource plugin module containing the Javascript and/or CSS for the extension. You can have multiple `web-resource` elements in the extension declaration, thus grouping several web resources in a single widget extension.

Issue Data Provider Module

Issue data provider module lets you register data providers for the Structure widget columns.

Issue data provider example

```
<structure-issue-data-provider key="idp-sbcolumn" name="issue-data:Status Bar Column Provider"

class="com.almworks.jira.structure.sbcolumn.StatusBarDataProvider"/>
```

Element	Description
structure-issue-data-provider	The module descriptor.
structure-issue-data-provider/@key	The unique identifier of the plugin module. Required.
structure-issue-data-provider/@name	The human-readable name of the plugin module.
structure-issue-data-provider/@class	The class that implements the data provider. Must implement IssueDataProvider. Required.

Export Renderer Provider Module

Export renderer provider module lets you register the components responsible for exporting Structure columns to printable HTML and Microsoft Excel formats.

Export renderer provider example

```
<structure-export-renderer-provider
     key="erp-sbcolumn-excel"
     name="export-renderer:Status Bar Column Excel Provider"
     class="com.almworks.jira.structure.sbcolumn.StatusBarExcelProvider">

  <column-key>com.almworks.jira.structure.sbcolumn</column-key>
  <export-format>ms-excel</export-format>
</structure-export-renderer-provider>
```

Element	Description
structure-export-renderer-provider	The module descriptor.
structure-export-renderer-provider/@key	The unique identifier of the plugin module. Required.
structure-export-renderer-provider/@name	The human-readable name of the plugin module.
structure-export-renderer-provider/@class	The class that implements the renderer provider. Must implement ExportRendererProvider. Required.
column-key	The column key that this provider is associated with. You can have multiple `column-key` elements in a single descriptor. If no column key is specified, the renderer provider is considered generic – such a provider will be consulted for every column not served by a type-specific provider.
export-format	The export format that this provider is associated with. The values are `printable` for the printable HTML format and `ms-excel` for the Microsoft Excel XLS format. You can have multiple `export-format` elements in a single descriptor. If no export format is specified, the renderer provider is

Element	Description
	considered generic – such a provider will be consulted for every column not served by a format-specific provider.

4.5.4 Structure REST API Reference

Structure REST API is yet under active development. The functionality available through REST is not complete, yet it allows to work with the structures.

The API is also not stable, although we're not seeing major changes coming to the main resources.

General Notes

REST Resource Address

Structure REST API resources have the URL

```
BASEURL/rest/structure/1.0/NAME
```

where BASEURL is the base JIRA address (`http://localhost:2990/jira` being standard base URL for development environment) and NAME is the name of the resource.

Authentication

Authentication is done via standard JIRA authentication engine and supported by cookies. When accessing REST API from a remote application, you may need to set up the session first by calling JIRA authentication REST resource. (You don't need to do that if you access Structure REST API from a JavaScript on a page from the same JIRA instance.)

Most read operations are available to non-authenticated access (subject to permission checks for the anonymous user). Most mutation operations are available to authenticated users only.

REST Resources and Operations

Structure resources

`/structure/ GET`	list structures
`/structure/ POST`	create a structure
`/structure/{id} GET`	read information about a structure
`/structure/{id}/update POST`	update one or several structure fields
`/structure/{id} DELETE`	delete a structure

Forest resources

`/structure/id/forest GET`	get a forest
`/structure/id/forest POST`	make changes to a forest

Structure Resource

This page describes resources with which you can list, create, read, update, and delete structures. Structures contain general information such as name and permissions, but not the hierarchy itself. Issue hierarchy is accessed through the so-called *forest* resources (read, modify). This page also documents structure shape and its fields, and the error entity that may be returned in case of the REST API user error.

`/structure/ GET`	list structures
`/structure/ POST`	create a structure
`/structure/{id} GET`	read structure
`/structure/{id}/update POST`	update one or several structure fields
`/structure/{id} DELETE`	delete structure

Quick navigation:

- Structure Representations
- Structure Fields
- Permission Rules
- Error Entity

Structure Representations

Structure is represented via JSON. All resources are also capable of producing XML.

```
{
  "id": 103,
  "name": "Structure with all fields",
  "description": "Voilà! This structure exhibits all fields.",
  "readOnly": "true",
  "editRequiresParentIssuePermission": true,
  "permissions": [
    {
      "rule": "apply",
      "structureId": 102
    },
    {
      "rule": "set",
      "subject": "group",
      "groupId": "jira-developers",
      "level": "edit"
```

```
    },
    {
      "rule": "set",
      "subject": "projectRole",
      "projectId": 10010,
      "roleId": 10020,
      "level": "admin"
    },
    {
      "rule": "set",
      "subject": "anyone",
      "level": "view"
    },
    {
      "rule": "set",
      "subject": "user",
      "username": "agentk",
      "level": "none"
    }
  ],
  "owner": "user:admin"
}
```

Top

Structure Fields

Structure objects accessible through these resources have the following fields, most of which represent structure details as outlined in the Structure User's Guide:

`id`	The ID of the structure (integer, `1..2^63 - 1`.)
`name`	The name of the structure. A structure must have a non-empty name, which does not have to be unique.
`description`	The description on the structure. May be absent.
`readOnly`	`true` if the user has only View access level to the structure, otherwise absent.
`editRequiresParentIssuePermission`	`true` if the Require Edit Issue Permission on Parent Issue flag is set on this structure, otherwise absent.
`permissions`	The list of structure permission rules. Present only if the user has Control access level to the structure. Some resources do not include permissions unless requested to do so. List order is as important as the rules themselves.
`owner`	The owner of the structure. Present only if the user is the owner of this structure or if he has Browse Users

	permission. A string of the form `user:USERNAME`, where USERNAME is the JIRA user login name. Example: `user:jsmith`.

Please note that structure resources described on this page do not include information about issue hierarchies. The so-called *forest* of a structure, i.e. its hierarchy of issues, can be read or modified using the dedicated resources.

Top

Permission rules

There are two types of permission rules, those that *set* permissions and those that *apply* permissions from another structure. They have different fields depending on the type.

Set rules

`rule`	Must be equal to `set`, case-insensitive.
`subject`	Identifies the type of the subject to which the rule applies. Must be one of `group, projectRole, user, anyone`. See below how to identify the subject.
`level`	Access level to set to the specified subject. Must be equal to one of the names of the Permission Level enum constants, case-insensitive. Please note that Control permission is represented by the ADMIN enumeration constant.

In addition, there are fields to identify the subject.

group
The rule applies to all users within the JIRA group.

`groupId`	The name of the JIRA group. Example: `jira-developers`.

REST API user can create such rule only for a group he belongs to.
projectRole
The rule applies to all users that have a role in a project.

`projectId`	The ID of the project. Example: `10010`.
`roleId`	The ID of the role. Example: `10010`.

REST API user can create such rule only for roles in projects where Structure is enabled, and for which he has Browse Projects permission.
user
The rule applies to the user.

`username`	Name of the user. Example: `jsmith` for user John Smith.

REST API user can create such rule only if he has Browse Users permission, and if such user exists.

anyone
The rule applies to all users, even anonymous (not authenticated.) The rule shouldn't have any additional fields.
Apply rules

`rule`	Must be equal to `apply`, case-insensitive.
`structureId`	The ID of the structure which permissions should be applied. Example: `112`

Apply rule creates a dependency on another structure. Circular dependencies are not allowed. Also, a REST API user can create such rule only if he has Control access level to the referenced structure.

Top

Error entity

```
{
  "code": 4005,
  "error": "STRUCTURE_NOT_EXISTS_OR_NOT_ACCESSIBLE[4005]",
  "structureId": 160,
  "message": "Referenced structure [160] does not exist or you don't have Control permissions on it.",
  "localizedMessage": "Das Struktur [160] existiert nicht oder sie haben keine Kontrolle Berechtigungen."
}
```

In some cases, requests to structure resources result in an error response containing an error entity. Any of its fields may be absent.

`code`	Integer code of the error
`error`	Brief technical description of the error. Contains a name of the corresponding StructureError enum constant.
`structureId`	The ID of the structure involved.
`issueId`	The ID of the JIRA issue involved.
`message`	More detailed message, may contain technical details.
`localizedMessage`	User-displayable message in the REST API user locale or JIRA default locale if the user is not authenticated.

Top

Structure Resources

GET /structure

```
GET $baseUrl/rest/structure/1.0/structure
GET
$baseUrl/rest/structure/1.0/structure?name=$name&permission=$permission&issueId=$issueId&withPermi
```

A list of all structures visible to the REST API user. Optionally, the result can be filtered by name, REST API user's access level, or issue contained within the structure. By default, permission rules and owners are not included, you should use query parameters if you want them to be included.

> **Who can access this resource**
>
> All users who have access to the Structure Plugin. The returned list contains only structures to which the REST API user has at least View access level.

Request
Query parameters:

`name`	If present, the returned list will contain only structures which names equal the specified name (case-insensitive.)
`permission`	If present, the returned list will contain only structures to which the REST API user has the specified access level. Must be equal to one of the names of the Permission Level enum constants, case-insensitive. NONE is treated in the same way as VIEW. Please note that Control permission is represented by the ADMIN enumeration constant.
`issueId`	If present, the returned list will contain only structures to which the specified issue belongs. Must be an integer. Example: `12147`. *REST API user must have permissions to access the specified issue.*
`withPermission`	If `true`, permission rules will be included in the response. Default is `false`.
`withOwner`	If `true`, owner will be included in the response. Default is `false`.

Each of the filter parameters `name`, `permission`, or `issueId` can be specified only once, otherwise the first is used. Different parameters are combined with AND.

HTTP headers:

`Content-Type`	Should be one of `application/json`, `application/xml`.
`Accept`	Should be one of `application/json`, `application/xml`.

Response
Success

`200 OK`	Response entity contains the only field, `structures`, which contains the list of the structure objects.	`application/json`, `application/xml`

Example 1: all structures

```
GET $baseUrl/rest/structure/1.0/structure
```

```
{
  "structures": [
    {
      "id": 1,
      "name": "Global Structure",
      "description": "Initial general-purpose structure.",
      "editRequiresParentIssuePermission": true
    },
    {
      "id": 102,
      "name": "Test plan",
      "description": "Test plan #3",
      "readOnly": true
    },
    {
      "id": 100,
      "name": "Test plan",
      "description": "Test plan #1"
    },
    {
      "id": 101,
      "name": "Test plan",
      "description": "Test plan #2"
    }
  ]
}
```

Example 2: only "Test plan"

```
GET $baseUrl/rest/structure/1.0/structure?name=test+plan
```

```
{
  "structures": [
    {
      "id": 102,
      "name": "Test plan",
      "description": "Test plan #3",
      "readOnly": true
    },
    {
      "id": 100,
      "name": "Test plan",
      "description": "Test plan #1"
```

```
    },
    {
      "id": 101,
      "name": "Test plan",
      "description": "Test plan #2"
    }
  ]
}
```

Example 3: only with a specific issue

```
GET $baseUrl/rest/structure/1.0/structure?issueId=12147
```

```
{
  "structures": [
    {
      "id": 1,
      "name": "Global Structure",
      "description": "Initial general-purpose structure.",
      "editRequiresParentIssuePermission": true
    },
    {
      "id": 100,
      "name": "Test plan",
      "description": "Test plan #1"
    }
  ]
}
```

Example 4: "Test plan" containing specific issue

```
GET $baseUrl/rest/structure/1.0/structure?name=test+plan&issueId=12147
```

```
{
  "structures": [
    {
      "id": 100,
      "name": "Test plan",
      "description": "Test plan #1"
    }
  ]
}
```

Example 5: structures that the user can edit with permissions and owners shown

```
GET $baseUrl/rest/structure/1.0/structure?permission=edit&withPermissions=true&withOwner=true
```

```
{
  "structures": [
    {
      "id": 1,
      "name": "Global Structure",
      "description": "Initial general-purpose structure.",
      "editRequiresParentIssuePermission": true
    },
    {
      "id": 100,
      "name": "Test plan",
      "description": "Test plan #1",
      "permissions": [
        {
          "rule": "set",
          "subject": "group",
          "groupId": "jira-users",
          "level": "edit"
        },
        {
          "rule": "set",
          "subject": "projectRole",
          "projectId": 10010,
          "roleId": 10010,
          "level": "none"
        },
        {
          "rule": "apply",
          "structureId": 101
        }
      ],
      "owner": "user:jsmith"
    },
    {
      "id": 101,
      "name": "Test plan",
      "description": "Test plan #2",
      "owner": "user:admin"
    }
  ]
}
```

Example 6: require XML representation

Note that the same can be achieved by specifying `application/xml` in the `Accept` HTTP header.

```
GET $baseUrl/rest/structure/1.0/structure.xml?name=test+plan&issueId=12147
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<structureList>
  <structures>
    <structure>
      <id>100</id>
      <name>Test plan</name>
      <description>Test plan #1</description>
```

```
    </structure>
  </structures>
</structureList>
```

Error

`400 Bad Request`	`permission` parameter is set to an unknown value, or request is invalid for other reasons. In the first case, response contains error entity, in the second it is empty.	`application/json`, `application/xml`
`403 Forbidden`	If Structure Plugin is not accessible to the REST API user, or if issue with ID `issueId` does not exist or the REST API user does not have enough permissions to access it. Response contains error entity.	`application/json`, `application/xml`
`404 Not Found`	If `issueId` is not an integer. Response entity contains a standard JIRA error HTML page.	`text/html`
`500 Internal Server Error`	If an internal error has occurred while processing this request.	
`503 Service Unavailable`	If Structure Plugin is stopped at the time of request. For example, the Restore operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top

POST /structure

```
POST $baseUrl/rest/structure/1.0/structure
```

Create a structure by POSTing to this resource.

> **Who can access this resource**
>
> Only logged in users who have access to the Structure Plugin and a permission to create structures.

Request
Request entity should contain the new structure. Structure name, `name`, must be present and non-empty. Fields `id`, `readOnly`, and `owner` are ignored. All rules in `permissions` are validated according to their respective rule types.

Please note that this resource accepts only JSON structure representation.

HTTP headers:

`Content-Type`	Must be `application/json`.
`Accept`	Should be one of `application/json`, `application/xml`.

Response
Success

`201 Created`	Response entity contains the created structure with fields, including `permissions` and `owner`.	`application/json`, `application/xml`

Example 1: minimal structure

```
POST $baseUrl/rest/structure/1.0/structure
```

Request entity

```
{
  "name":"Test plan"
}
```

Response entity

```
{
  "id": 104,
  "name": "Test plan",
  "description": "",
  "permissions": [],
  "owner": "user:admin"
}
```

Example 2: structure with some permissions

```
POST $baseUrl/rest/structure/1.0/structure
```

Request entity

```
{
  "name":"Structure with some permissions",
  "editRequiresParentIssuePermission":"true",
  "permissions":[
    {
      "rule":"apply",
      "structureId":102
    }
  ]
}
```

Response entity

```
{
  "id": 105,
  "name": "Structure with some
permissions",
  "description": "",
  "editRequiresParentIssuePermission":
true,
  "permissions": [
    {
      "rule": "apply",
      "structureId": 102
    }
  ],
```

Request entity	Response entity
	<pre> "owner": "user:admin" }</pre>

Error

`400 Bad Request`	Structure data is not well-formed (syntax error) or invalid (semantic error.) Not well-formed structure data examples: request JSON is syntactically incorrect; JSON contains unknown field; `name` is not present or empty; `permissions` list contains a *set* rule with `level` set to an invalid value. Invalid structure example: `permissions` list contains a rule that fails validation. Response entity contains error. Problems with *apply* rule usually have `structureId` to indicate the invalid reference.	`application/json`, `application/xml`
`403 Forbidden`	If REST API user is not logged in or does not have permissions to access Structure Plugin or to create structures. Response contains error entity.	`application/json`, `application/xml`
`500 Internal Server Error`	If an internal error has occurred while processing this request.	
`503 Service Unavailable`	If Structure Plugin is stopped at the time of request. For example, the Restore operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top

GET /structure/{id}

```
GET $baseUrl/rest/structure/1.0/structure/$id
GET
$baseUrl/rest/structure/1.0/structure/$id?withPermissions=$withPermissions&withOwner=$withOwner
```

This resource allows to obtain structure details for the particular structure. By default, `permissions` and `owner` are not included, use query parameters to include them.

Who can access this resource

> All users who have access to the Structure Plugin. To access the particular structure, the user has to have at least View access level.

Request
Path parameter:

`id`	the ID of the structure

Query parameters:

`withPermission`	If `true`, permission rules will be included in the response. Default is `false`.
`withOwner`	If `true`, owner will be included in the response. Default is `false`.

HTTP headers:

`Content-Type`	Should be one of `application/json`, `application/xml`.
`Accept`	Should be one of `application/json`, `application/xml`.

Response
Success

`200 OK`	Response entity contains the created structure along with all of its fields. Field `permissions` is included if the REST API user has Control permission on this structure. Field `owner` is included if the REST API user is either the owner of this structure or has Browse Users permission.	`application/json`, `application/xml`

Example 1: retrieve structure with ID 100 without permissions and owner

```
GET $baseUrl/rest/structure/1.0/structure/100
```

```
{
  "id": 100,
  "name": "Test plan",
  "description": "Test plan #1"
}
```

Example 2: permissions and owner are requested to be included, but only owner is shown, because the user has only View access as indicated by readOnly

```
GET $baseUrl/rest/structure/1.0/structure/102?withOwner=true&withPermissions=true
```

```
{
  "id":102,
  "name":"Test plan",
  "description":"Test plan #3",
  "readOnly":true,
  "owner":"user:admin"
}
```

Example 3: XML representation may be requested in the request URL instead of the Content-Type HTTP header

```
GET $baseUrl/rest/structure/1.0/structure/102.xml
```

```
<structure>
  <id>102</id>
  <name>Test plan</name>
  <description>Test plan #3</description>
  <readOnly>true</readOnly>
</structure>
```

Error

`400 Bad Request`	One of the query parameters is too long.	
`403 Forbidden`	If REST API user does not have permissions to access Structure Plugin or does not have at least View permission on this structure. Response contains error entity.	`application/json`, `application/xml`
`404 Not Found`	If `id` is not an integer in `1..2^63-1`. Response entity contains a standard JIRA error HTML page.	`text/html`
`500 Internal Server Error`	If an internal error has occurred while processing this request.	
`503 Service Unavailable`	If Structure Plugin is stopped at the time of request. For example, the Restore operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top

POST /structure/{id}/update

```
POST $baseUrl/rest/structure/1.0/structure/$id/update
```

Update one or several fields of a structure by POSTing to this resource.

> **Who can access this resource**
>
> Only logged in users who have access to the Structure Plugin and Control permission on this structure.

Request
Request entity should contain those structure fields that need to be changed. Non-present fields will not be changed (for this user; `readOnly` may change for other users as a result of changing `permissions`.) Fields `id`, `readOnly`, and `owner` are ignored.

Please note that `permissions` field is modified as a whole, so to add a rule, you have to provide the new list of rules in the proper order.

If `permissions` field is present, all rules are validated according to their respective rule types.

Please note that this resource accepts only JSON structure representation.

HTTP headers:

`Content-Type`	Must be `application/json`.
`Accept`	Should be one of `application/json`, `application/xml`.

Response
Success

`200 OK`	Response entity contains the updated structure with all fields, including `permissions` and `owner`.	`application/json`, `application/xml`

Example 1: change description of the Global Structure

```
POST $baseUrl/rest/structure/1.0/structure/1/update
```

Request entity

```
{
  "description":"Company-wide structure
providing the Big Picture."
}
```

Response entity

```
{
  "id":1,
  "name":"Global Structure",
  "description":"Company-wide structure
providing the Big Picture.",
  "editRequiresParentIssuePermission":true,
  "permissions":[
    {
      "rule":"set",
      "subject":"anyone",
```

Request entity	Response entity

	<pre>"level":"view"
	},
	{
	"rule":"set",
	"subject":"group",
	"groupId":"jira-users",
	"level":"edit"
	},
	{
	"rule":"set",
	"subject":"group",
	"groupId":"jira-administrators",
	"level":"admin"
	}
	]
	}</pre>

Example 2: changing permission rules

```
POST $baseUrl/rest/structure/1.0/structure
```

Request entity	Response entity

<pre>{
"permissions":[
{
"rule":"set",
"subject":"group",
"groupId":"jira-users",
"level":"edit"
},
{
"rule":"apply",
"structureId":101
}
]
}</pre>
	<pre>{
	"id": 105,
	"name": "Structure with some permissions",
	"description": "",
	"editRequiresParentIssuePermission": true,
	"permissions": [
	{
	"rule": "set",
	"subject": "group",
	"groupId": "jira-users",
	"level": "edit"
	},
	{
	"rule": "apply",
	"structureId": 101
	}
	],
	"owner": "user:admin"
	}</pre>

Error

`400 Bad Request`	Structure data is not well-formed (syntax error) or invalid (semantic error.)	

	Not well-formed structure data examples: request JSON is syntactically incorrect; JSON contains unknown field; `permissions` list contains a *set* rule with `level` set to an invalid value. Invalid structure example: `permissions` list contains a rule that fails validation. Response entity contains error. Responses to problems with an *apply* rule usually have `structureId` to indicate the invalid reference.	`application/json`, `application/xml`.
`403 Forbidden`	If REST API user is not logged in, does not have permissions to access Structure Plugin, or does not have Control access level to this structure. Response contains error entity.	`application/json`, `application/xml`
`500 Internal Server Error`	If an internal error has occurred while processing this request.	
`503 Service Unavailable`	If Structure Plugin is stopped at the time of request. For example, the Restore operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top

DELETE /structure/{id}

Deletes the designated structure.

> **Who can access this resource**
>
> Only logged in users who have access to the Structure Plugin and Control permission on this structure.

Request
Path parameter:

`id`	the ID of the structure

HTTP headers:

`Content-Type`	Must be `application/json`.
`Accept`	Should be absent or equal to one of `application/json`, `application/xml`.

Response

Success

`200 OK`	Contains an object with the only field `empty` with value `true`.	`application/json`, `application/xml`

Note: it should have been `204 No content` instead, but there were reports of some browsers (Firefox) incorrectly processing such results, so it's as it is.

Example

```
DELETE $baseUrl/rest/structure/1.0/structure/108
```

```
{
  "empty": true
}
```

Error

`403 Forbidden`	If REST API user is not logged in, does not have permissions to access Structure Plugin, or does not have Control access level to this structure. Response contains error entity.	`application/json`, `application/xml`
`404 Not Found`	If `id` is not an integer in `1..2^63-1`. Response entity contains a standard JIRA error HTML page.	`text/html`
`404 Not Found`	If `id` is an integer in `1..2^63-1`, but the structure with the specified `id` does not exist or the user does not have View access level to it.	`application/json`, `application/xml`
`500 Internal Server Error`	If an internal error has occurred while processing this request.	
`503 Service Unavailable`	If Structure Plugin is stopped at the time of request. For example, the Restore operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top

Forest Resource - GET

Request

```
GET $baseUrl/rest/structure/1.0/structure/$id/forest
GET $baseUrl/rest/structure/1.0/structure/$id/forest?root=$root
```

Returns the hierarchical issue list (forest) of the specified structure.

Parameters:

$id	*required*	the ID of the structure
$root	*optional*	the ID of the pinned issue for the Pinned Issue Mode

Response

```
{
   "structure":100,
   "version":2981,
   "root":10001,
   "formula":"10009:0,10008:1,10006:2,10002:3,10001:4,10062:4,10070:3,10071:2,10041:2"
}
```

Response Fields:

structure	the ID of the structure
version	the version of the foresst
root	the pinned issue if the fixed structure view has been requested (absent otherwise)
formula	a string representation of the issue forest

Forest Formula

The forest of issues is represented by a string formula, which consists of pairs "*issue ID*:*depth*", delimited by a comma. Each pair represents a row in the fully expanded view of the forest.

```
ForestFormula ::= NodeSequence
NodeSequence ::= Node [ "," NodeSequence ]
Node ::= IssueID ":" Depth
IssueID ::= Integer
Depth ::= Integer
```

Forest Resource - POST

Request

```
POST $baseUrl/rest/structure/1.0/structure/$id/forest

{
   "base":0,
   "root":0,
   "actions":[
      {
```

```
            "action":"move",
            "issue":13212,
            "under":10037,
            "after":13210
         },
         {
            "action":"delete",
            "issue":13213
         },
         {
            "action":"add",
            "issue":10410,
            "under":10092,
            "after":0
         }
      ]
   }
```

Updates the hierarchical issue list (forest) by applying the specified actions.

Parameters:

$id	*required*	The ID of the structure.
base	*required*	The base version for the update reply from the server *Set to 0. Detailed explanation of this parameter will be published later.*
root	*required*	The ID of the pinned issue for the Pinned Issue Mode, used only to create the reply.
actions	*required*	Array of actions to be applied to the forest, must contain at least one action. The actions are applied in the specified order.

Possible actions

The `actions` array may contain the following JSON objects:

Move action

action	"move"
issue	the ID of the issue to move
under	the new parent for the moved issue, or 0 to put it at the top level
after	the sibling issue (issue under the same parent) that must come before the moved issue, or 0 to place the issue as the first child issue

If the moved issue contains sub-issues, the whole sub-tree will be moved.

This action object corresponds to the `moveSubtree` method of the ForestAccessor interface.

Add action

action	"add"

issue	the ID of the issue to add
under	the new parent for the moved issue, or 0 to put it at the top level
after	the sibling issue (issue under the same parent) that must come before the moved issue, or 0 to place the issue as the first child issue

The new issue will be added to the structure at the position specified by `(under, after)` coordinates.

This action object corresponds to the `addIssue` method of the ForestAccessor interface.
Delete action

action	"delete"
issue	the ID of the issue to delete

The issue will be removed from the forest. (The issue itself will not be changed.)

If the removed issue contains sub-issues, the whole sub-tree will be removed.

This action object corresponds to the `removeSubtree` method of the ForestAccessor interface.

Response

```
{
   "update": {
      ...
   },
   "errors": [
      "error text"
   ]
}
```

Response Fields:

update	the object containing update to the client's structure specified by parameters `root` and `base`, absent if `base` is `0`
errors	array of the error strings, absent if no errors

4.5.5 Structure JavaScript API Reference

Structure's JavaScript API provides ways to extend the client-side functionality of the Structure plugin.

JavaScript API Functions

This page lists static functions exposed by the Structure API.

JavaScript API Classes

Structure Javascript API provides a number of classes to be used as base for your own column type implementations. This should be done using subClass() method.

- Column Class — A subclass of Column class represents column objects of a specific type.
- ColumnConfigurator Class — ColumnConfigurator class encapculates everything related to column type configuration.
- ColumnOption Class — ColumnOption class represents a single column configuration parameter.
- ColumnType Class — ColumnType class represents column type.

JavaScript API Functions

This page lists static functions exposed by the Structure API.

window.almworks.structure.api.subClass(className, superclass, prototype)

Creates a subclass of a specific class. Returns a constructor function that will create the instances of the class.

This function provides light-weight polymorphism for the purposes of extending Structure's classes.

Parameters

`className`	*string*	Class name as string (optional, used for friendly instance names in debugger)
`superclass`	*Object*	Superclass reference
`prototype`	*Object*	Subclass prototype

The returned value – class constructor – takes a single optional `options` parameter.

The prototype may contain a special `init()` initializer method, which is called when an instance is being constructed. Superclass' `init()` method is called before subclass' method. Options that were passed to the constructor are passed through to the initializer.

Example

```
var MyClass = window.almworks.structure.api.subClass('MyClass', BaseClass, {
  init: function(options) {
   ...
  },

  someMethod: function() {
   ...
  }
});

var options = { ... };
var instance = new MyClass(options);
```

window.almworks.structure.api.registerColumnType(type, key)

Registers a new column type. If you're extending Structure by adding a new type of column to the grid, the type must be registered from your additional JavaScript web resource.

Column types are identified by a unique key, which is recorded in the view specification, along with the type-specific parameters and column name.

Parameters

`type`	*Object*	A `ColumnType` instance, implementing a specific column type – see ColumnType Class
`key`	*string*	Column type key (can also be array of strings if the type can handle multiple variations of a column specification)

Example

```
window.almworks.structure.api.registerColumnType(new MyColumnType(),
'com.acme.structure.awesome-column');
```

We recommend using a unique key that has low chance of conflicting with column types provided by other, independent developers. A good approach is to have Java-like package notation for the keys.

window.almworks.structure.api.registerColumnGroup(options)

Registers a new column type group. Column groups are used in the "Add Column" panel to group column configuration presets, provided by column types.

Parameters

`options.groupKey`	*string*	Group key. The same key should be returned by all `ColumnConfigurator`'s `getGroupKey()` method for all column types which need to appear in this particular group.
`options.title`	*string*	Group title
`options.order`	*number*	Group order is used to sort groups. Order value for groups provided by Structure are 100, 200, 300 and so on.

Example

```
window.almworks.structure.api.registerColumnGroup({
  groupKey: 'com.acme.structure.colgroup', title: 'Acme Columns', order: 50
});
```

JavaScript API Classes

Structure Javascript API provides a number of classes to be used as base for your own column type implementations. This should be done using subClass() method.

- Column Class — A subclass of Column class represents column objects of a specific type.
- ColumnConfigurator Class — ColumnConfigurator class encapculates everything related to column type configuration.
- ColumnOption Class — ColumnOption class represents a single column configuration parameter.
- ColumnType Class — ColumnType class represents column type.

Column Class

window.almworks.structure.api.Column

A subclass of Column class represents column objects of a specific type.

Columns need to be subclassed for a particular column type implementation. You can override methods while subclassing to modify the default behavior.

Example

```
var api = window.almworks.structure.api;
var MyColumn = api.subClass('MyColumn', api.Column, {
  init: function() {
    ...
  },
  getCellViewHtml: function() {
    return '<div> ... </div>';
  }
});
```

Properties

context
Contains context information about where the column is used. See The Column Context for more information.
spec
Contains column specification object. Specification object is serialized as a part of the overall view specification and stored on the server and in the browser's local storage. See Column Specifications for more information.

Methods

init(options)
Initializer method.
getCellValueHtml(renderingParameters)
Returns HTML that is displayed in the grid cell for a specific issue. The HTML should contain the value provided by this column. Structure will also wrap the value in decorative elements – this could be overridden by providing `getCellViewHtml()` method.

Parameters

renderingParameters.getFieldValue()	returns current row's field value
renderingParameters.getIssueId()	returns current row's issue id
renderingParameters.getEscapedFieldValue()	returns current row's field value escaped

Example

```
getCellValueHtml: function(rp) {
  return '<span class="acme-field">' +
rp.getEscapedFieldValue('com.acme.structure:awesome-field-data') + '</span>';
}
```

getCellViewHtml(renderingParameters)
Returns customized HTML that is displayed in the grid cell for a specific issue. By default, calls `getCellValueHtml()` and wraps the retrieved value into the default Structure style. Can be overridden to allow higher degree of control over the cell appearance.

Parameters

renderingParameters.getFieldValue()	returns current row's field value
renderingParameters.getIssueId()	returns current row's issue id
renderingParameters.getEscapedFieldValue()	returns current row's field value escaped

getRequiredFields()
Returns array of data fields required to display column. Those data must be provided by pluggable data providers in Java code.

Example

```
getRequiredFields: function() {
  return [ 'com.acme.structure:awesome-field-data', 'jira:raw:status' ];
}
```

The fields below can be provided by Structure, so you don't need to write your own data providers.

Format	Example	Description
`jira:html:<fieldID>`	`jira:html:assignee`	The HTML representation of an issue field value, as seen on the issue page or in the Issue Navigator.
`jira:raw:<fieldID>`		

Format	Example	Description
	`jira:raw:issuekey` `jira:raw:project`	The "raw", unformatted representation of an issue field. This can be a string or numeric ID of the referenced entity. The following fields are supported: • summary • issuekey • issuetype • resolution • status • project • priority • assignee • reporter
`jira:permission:<permission>`	`jira:permission:edit`	Returns "1" if the user has the given permission for the issue, "0" otherwise.
`jira:icon:<fieldID>`	`jira:icon:project`	The icon for the referenced entity, as an HTML fragment. The following fields are supported: • issuetype • priority • status • assignee • reporter • project

`structure:sum:<fieldID>`	`structure:sum:timespent`
	`structure:sum:votes`
	`structure:sum:customfield_10153`
		The sum of a numeric or duration field over the issue's subtree in the structure. The value is formatted according to the user's locale.

getDefaultName()
Must return default column name, assigned when user adds column of specified type to the structure view. Returns empty string by default.

Example

```
getDefaultName: function() { return 'My Column'; }
```

isResizable()
Returns whether the column is resizable or not. Returns `true` by default.

Example

```
isResizable: function() { return false; }
```

canShrinkWhenNoSpace()
Returns whether column can shrink beyond minimum size if there's not enough space on the screen. Returns `false` by default.

Example

```
canShrinkWhenNoSpace: function() { return true; }
```

isAutoSizeAllowed()
Returns if the column should be auto-resized to fit its contents. Returns `false` by default.

Example

```
isAutoSizeAllowed: function() { return true; }
```

getMinWidth()
Returns minimum width of the column in pixels. Returns 27 by default.

Example

```
getMinWidth: function() { return 100; }
```

getDefaultWidth()
Returns default width of the column in pixels. Returns 120 by default.

Example

```
getDefaultWidth: function() { return 100; }
```

getHeaderCellHtml()
Returns HTML that will be used in the grid header. By default returns cell with column name in default Structure style.

Example

```
getHeaderCellHtml: function() { return '<div>' + this.name + '</div>'; }
```

getMetadataRequests()
Returns a JavaScript object specifying the metadata needed by this column to render the values. See Requesting and Using Metadata for more information. By default returns `null`, which means that no metadata is needed.

Example

```
getMetadataRequests: function() {
  return {
    status: {
      url: baseUrl + '/rest/api/2/status',
      cacheable: true
    }
  };
}
```

ColumnConfigurator Class

window.almworks.structure.api.ColumnConfigurator

ColumnConfigurator class encapculates everything related to column type configuration.

It needs to be subclassed for a particular column type implementation and passed as return value in ColumnType.createConfigurator() method.

Example

```
var api = window.almworks.structure.api;

var MyColumnConfigurator = api.subClass('MyColumnConfigurator, api.ColumnConfigurator, {
  getDefaultColumnName: function() { return 'My Column'; }
  getOptions: function() {
    return [ new MyOption1({configurator: this}), new MyOption2({configurator: this}) ];
  }
});
```

Required Methods

You have to override the following methods in the subclass.

getColumnTypeName()
Returns column type name, used in the column configuration panel.

getDefaultColumnName()
Returns default column name.

Other Methods

These methods may be optionally overridden.

init(options)
Optional initializer.
getGroupKey()
Return column preset's group key. See registerColumnGroup() for reference.
getMetadataRequests()
Returns a JavaScript object specifying the metadata needed by this configurator to set up the UI. See Requesting and Using Metadata for more information. By default returns `null`, which means that no metadata is needed.

Example

```
getMetadataRequests: function() {
  return {
    somedata: {                         // metadata key
      url: baseUrl + '/some/data/url', // request URL
      cacheable: true,                  // if the response for this URL can be reused for other
cacheable requests
      extract: function(response) {    // response to the AJAX request
        return response.property || 1; // the actual value for context.getMetadata('somedata')
      }
    },
    otherdata: {
      url: baseUrl + '/other/data/url',
      cacheable: true
    }
  };
}
```

getOptions()
Returns array of column type options. Each option should be a subclass of ColumnOption Class.

Example

```
getOptions: function() {
  return [ new MyOption1({configurator: this}), new MyOption2({configurator: this}) ];
}
```

ColumnOption Class

window.almworks.structure.api.ColumnOption

ColumnOption class represents a single column configuration parameter.

It needs to be subclassed for particular column type implementation and passed as return value in ColumnConfigurator.getOptions() method.

Options are displayed in column configuration dialog one after another with labels on the left and inputs on the right.

Example

```
var api = window.almworks.structure.api;

var MyOption1 = api.subClass('MyOption', api.ColumnOption, {
  title: 'Some option',
  init: function() {
    this.input$ = null;
  },
  createInput: function(div$) {
    this.input$ = div$.append('<input type="text" class="text">').find('input');
    var params = this.spec.params;
    this.input$.on('change', function() {
      if (params.someOptionAvaiable) {
        params.someOption = $(this).val();
        div$.trigger('notify');
      }
    });
  },
  notify: function() {
    var available = this.spec.params.someOptionAvaiable;
    this.input$.val(available ? (this.spec.params.someOption || '42') : '');
    return available;
  }
});
```

Properties

title

If set, title is displayed as a label to the left of the input controls. Option title representation may be overridden in #createLabel(div$) method.

Required Methods

You need to override the following methods.

createInput(div$)

Should be overridden to provide custom HTML for the option input. `div$` parameter provides parent option element to append your view to. Created input should trigger `'notify'` event on `div$` to notify Structure of any column parameters change.

Please honor the AUI Forms HTML layout when creating your input controls!

Example

```
createInput: function(div$) {
  var self = this;
```

```
  this.input$ = $('<input type="text" class="text">').appendTo(div$).on('change', function() {
    if (self.spec.params.myOption !== $(this).val()) {
      self.spec.params.myOption = $(this).val();
      div$.trigger('notify');
    }
  });
}
```

Other Methods

init(options)

Optional initializer.

createLabel(div$)

May be overridden to provide custom HTML view for the input label. `div$` parameter provides parent option element to append your view to. By default creates a right-aligned label with text of the #title property.

Please honor the AUI Forms HTML layout if you override this method!

notify()

This method is called when the column configuration has changed. The implementation may want to update its controls to reflect those changes. The method should return a `boolean` indicating whether this option is available. Unavailable options will not be shown on the configuration panel. The default implementation does nothing and always returns `true`.

Example

```
notify: function() {
  this.input$.val(this.spec.params.myOption);
  return true;
}
```

isInputValid()

Returns `true` if the current column specification is valid from the point of view of this option. The column configuration won't be saved unless all of the options approve the specification. The default implementation does nothing and returns `true`.

Example

```
isInputValid: function() {
  // Check that the "field" specification parameter is present.
  return !!this.spec.params.field;
}
```

ColumnType Class

window.almworks.structure.api.ColumnType

ColumnType class represents column type.

It needs to be subclassed for particular column type implementation.

Example

```
var api = window.almworks.structure.api;

var AwesomeColumnType = api.subClass('AwesomeColumnType', api.ColumnType, {
  createSwitchTypePreset: function(context) { return { key: 'com.acme.structure.awesome-column',
params: {} }; },
  createAddColumnPresets: function(context) { return [
   { key: 'com.acme.structure.awesome-column', params: {} },
   { key: 'com.acme.structure.awesome-column', name: 'Awesome Column with a Twist', params: {
twist: true } }
  ]; },
  createConfigurator: function(context, spec) { return new AwesomeColumnConfigurator({context:
context, spec: spec}); },
  createColumn: function(context, spec) { return new AwesomeColumn({context: context, spec:
spec}); }
});

api.registerColumnType(new AwesomeColumnType(), 'com.acme.structure.awesome-column');
```

Methods

createSwitchTypePreset(context)

Returns default column specification to use when the user switches to this column type from another column type in the column configuration panel. May return `null` if the column type is unavailable.

createAddColumnPresets(context)

Returns an array of column presets (specifications) for this type to be offered to the user in the Add Column panel. May return an empty array if the column type is unavailable.

createColumn(context, spec)

Returns a new instance of `Column` subclass for the specified column specification. May return `null` if the specification is invalid, the column type is unavailable, etc.

createConfigurator(context, spec)

Returns a new instance of `ColumnConfigurator` subclass for the specified column specification. May return `null` if the specification is invalid, the column type is unavailable, etc.

getPresetMetadataRequests()

Returns a JavaScript object specifying the metadata needed by this column type to create presets. Unless the AJAX requests fail, the metadata will be available through `context.getMetadata(key)` when `createSwitchTypePreset()` or `createAddColumnPresets()` is called. See Requesting and Using Metadata for more information. By default returns `null`, which means that no metadata is needed.

getColumnMetadataRequests()

Returns a JavaScript object specifying the metadata needed by this column type to create `Column` instances. Unless the AJAX requests fail, the metadata will be available through `context.getMetadata(key)` when `createColumn()` is called, and also will be available to the created `Column` instance via `this.context`. See Requesting and Using Metadata for more information. By default returns `null`, which means that no metadata is needed.

getConfigMetadataRequests()

Returns a JavaScript object specifying the metadata needed by this column type to create `ColumnConfigurator` instances. Unless the AJAX requests fail, the metadata will be available through `context.getMetadata(key)` when `createConfigurator()` is called, and also will be available to the

created `ColumnConfigurator` instance via `this.context`. See Requesting and Using Metadata for more information. By default returns `null`, which means that no metadata is needed.

getMetadataRequests()

Returns a JavaScript object specifying the metadata needed by this column type. Unless the AJAX requests fail, the metadata will be available through `context.getMetadata(key)` when `createSwitchTypePreset()`, `createAddColumnPresets()`, `createColumn()`, or `createConfigurator()` is called, and also will be available to the created `Column` and `ColumnConfigurator` instances via `this.context`. See Requesting and Using Metadata for more information. By default returns `null`, which means that no metadata is needed.

Example

```
getMetadataRequests: function() {
  return {
    somedata: {                         // metadata key
      url: baseUrl + '/some/data/url', // request URL
      cacheable: true,                  // if the response for this URL can be reused for other
cacheable requests
      extract: function(response) {    // response to the AJAX request
        return response.property || 1; // the actual value for context.getMetadata('somedata')
      }
    },
    otherdata: {
      url: baseUrl + '/other/data/url',
      cacheable: true
    }
  };
}
```

4.5.6 Web Resource Contexts

The resources from the following web resource contexts are included by Structure pages:

Web resource context	Pages that include it
`structure.widget`	Structure Board, Structure Gadget See Loading Additional Web Resources For Structure Widget for the recommended way of extending the widget.
`structure.printable`	Printable page

For details about how to use web resource contexts, see Atlassian Developer Documentation.

4.6 API Usage Samples

Use the sample plugins to learn by example. Download the source bundle from this page and use it with the latest API version.

4.6.1 Download

Name	Version	Date
scheduled-sync-0.3.jar	1	2014-02-07 12:05
status-bar-column-1.0.jar	1	2013-09-26 14:57
structure-api-examples-2.5.0.zip	1	2013-09-26 14:57

The provided code is not production-quality and not supported. It is provided as a sample of how one can use Structure API.

The sample code is in public domain – feel free to copy, modify and base your work on it.

4.6.2 Sample Plugins List

Sample Plugin	Description
simple-plugin	Very basic demo of using `StructureManager`.
scheduled-sync	A plugin that allows to schedule periodical synchronization Resync - uses `StructureManager`, `StructureSyncManager` and `StructureJobManager`.
foo-synchronizer	A skeleton project for starting your own synchronizer plugin.
status-bar-column	Adds a column to the Structure widget that shows a colored bar, depending on the statuses of the sub-issues.
workflow-condition	Adds a workflow condition that checks if an issue matches a JQL.

5 Structure FAQ

5.1 Frequently Asked Questions

5.2 Cannot Create an Issue With +Next Issue (+Sub-Issue) Because of the Required Fields

5.2.1 Question

I have a number of fields required for the issues. When I try to use Structure's **+Next Issue** or **+Sub-Issue** button, the creation of the issue fails, because the values of the required fields were not provided.

5.2.2 Answer

You can enter other fields when creating a new issue.

1. Use "+" button to add the required fields to the view.
2. When entering a new issue, use **Tab** and **Shift+Tab** to switch between edited fields. You can also click in a cell to edit it, or use other Keyboard Shortcuts.

If the initial creation of an issue has failed, you don't have to lose the entered data. Just add the required fields and double-click on the value you need to edit, or click **Edit** button in the toolbar. You can change the values of the new issue and try to create it again.

For convenience, you can set up a separate view for entering new issues (or modify the preset view called *Entry*), so you can quickly switch between different sets of columns. See Saving and Sharing Views for details.

5.3 Plugin Manager Says Structure Is Unlicensed

5.3.1 Question

I have a valid license installed. Why do I see Structure as **Unlicensed** or having **Action Required** in the Plugin Manager?

5.3.2 Answer

That may be so because Plugin Manager is not aware of ALM Works licenses. To verify the true status of your Structure license, please check **Administration | Structure | License Details** page. If it shows you that the license is OK, you can safely ignore the status of the Structure license in Plugin Manager.

Structure supports two kinds of licenses — purchased via Atlassian and issued by ALM Works. For details, please see Setting Up Structure License.

5.4 No Check Mark Displayed for a Resolved Issue

5.4.1 Question

Why do I see a resolved issue in Structure, but there's no green check mark, which usually indicates that an issue is resolved?

This article answer these questions as well:

- Why do I see a check mark on a unresolved issue?
- Why does an open issue that still in the work have 100% progress indication?
- When I turn on "Unresolved" filter button, why do I see some of the resolved issues anyway?

5.4.2 Answer

The JIRA's notion of a "Resolved Issue" (or "Completed Issue") can be quite confusing. The source of confusion is that an issue is considered to be resolved based on its **Resolution** field, not based on its Status:

- **Unresolved** means that the Resolution field is empty, regardless of issue Status.
- **Resolved** means that the Resolution field has some value, regardless of issue Status.

If an issue has a non-empty Resolution field (i.e. considered Resolved):

- The green check mark is displayed in Structure on that issue;
- The issue is filtered out by the Unresolved button;
- The progress of the issue is 100% regardless of other fields.

See also: Flags Column, Searching and Filtering Within Structure, Progress Column

Problems Caused By Custom Workflows

The default workflow in JIRA contains the "Resolved" status and if you select this status, JIRA requires you to select some non-empty value for the Resolution field too. Thus, the issue gets the Resolved status and becomes truly resolved (or completed), because it has a value in the Resolution field.

The confusion may arise, if in a custom workflow / screen configuration, Resolution field is not set as required or not added to the screens, associated with transitions to the Resolved status. In this case, a user may move an issue to the Resolved status, but the issue will still be unresolved/uncompleted, because the Resolution field is still empty.

If you have such a configuration, in the Structure this problem may manifest itself when you are trying to use the Unresolved filter button (which works as a shorthand for filtering using JQL: "`Resolution is EMPTY`"). The issues with the Resolved status but with no Resolution will still be visible even if you switch the filter on.

Solution:

1. Edit your workflow: in all transitions to a status that should be considered resolved, use a screen with the Resolution field.
2. In all transitions to a status that should not be considered resolved, use "Clear Resolution" step.
3. Make Resolution field required. (It will matter only if Resolution is added to the screen configuration.)
4. Check all screens - "Edit Issue" screen and all screens not mentioned in (1) above should not contain Resolution field.

Problems Caused By Manually Added "Unresolved" Resolution Value

To make matters worse, sometimes JIRA administrators add a new resolution option, named "Unresolved". Then, for example, on the workflow's "Reopen" step configuration, instead of clearing the Resolution, they change it to this "Unresolved" value.

The problem is that the new "Unresolved" resolution is still a non-empty value, and any issue having this value in the Resolution field will be considered resolved, by JIRA and Structure and other plugins.

But on the issue page, the user will see *Resolution: Unresolved*. So it will be practically impossible to distinguish this resolved (completed) issue from the issues which are really unresolved (have empty Resolution field).

Solution:

1. Use JIRA's Bulk Change to clear resolution from all issues that have Resolution "Unresolved".
2. Remove resolution "Unresolved".

5.5 Structure plugin won't start

5.5.1 Question

I try to install (enable) Structure plugin, but it doesn't work. When I reload Plugin Manager page, Structure plugin is disabled. What is the problem?

5.5.2 Answer

Structure plugin may fail to start due to the following reasons. To better understand what's going on, check JIRA logs (`catalina.out` or `jira-application.log`) and verify each of the following possible causes.

Structure database cannot be created or opened, filesystem read-only or full

Structure stores all its data in `structure/` sub-directory of the JIRA home directory. At first launch, it tries to create that directory and shuts down if fails to do so. At every start it tries to open the database contained there and also shuts down if fails to do so. In all cases, there should be a big warning or error message in the JIRA log.

Possible actions:

- Create `structure` sub-directory manually and grant full permissions on it to the account that is used to run JIRA.
- Verify that filesystem is not read-only.
- Verify that there's enough free disk space (at least 100 MB).
- Verify that Structure's database is not opened with some other tool, like Derby console.

See also: Structure Files Location

Some of the required system plugins are disabled

Structure relies on some of the system plugins. If they are disabled, you may get all kind of weird messages from JIRA when it tries to start Structure.

Note that it is quite likely that the error messages will be completely unrelated to the disabled plugins. For example:

```
com.atlassian.plugin.PluginParseException: Unable to load the module's display conditions: Could
not load 'com.almworks.jira.structure.web.UserCanCreateStructureCondition' in plugin
com.almworks.jira.structure
... stack trace ...
Caused by: com.atlassian.plugin.web.conditions.ConditionLoadingException: Could not load
'com.almworks.jira.structure.web.UserCanCreateStructureCondition' in plugin
com.almworks.jira.structure
```

```
... stack trace ...
Caused by: java.lang.IllegalStateException: Cannot autowire object because the Spring context is
unavailable.  Ensure your OSGi bundle contains the 'Spring-Context' header.
... stack trace ...
```

Possible actions:

- Open **Administration | Plugins | Manage Plugins**, click **Show System Plugins**. Verify that all plugins are enabled. If some are disabled, enable them, then try to enable or reinstall Structure.

If for some reason you need to keep some of the plugins disabled, and Structure wouldn't start without them, please write to support@almworks.com.

Incomplete download or corrupt plugin JAR file

It is possible for the Plugin Manager to download the plugin JAR file only partially, if there are any problems with the server or the connection.

Also, it has been reported that if you download the plugin manually with Internet Explorer, it completely messes up the JAR file and turns it into a ZIP file with absolutely invalid content.

To verify that you have a correct JAR file, locate plugin JAR in `plugins/installed-plugins` directory under your JIRA home. Structure plugin has the word "structure" in its file name. Verify that the JAR file MD5 hash is the same as listed on the Download Archive page.

Incorrect JIRA setup

A symptom that provides evidence in favor of this cause is that JIRA application logs contain one or several lines that look like the following:

```
ERROR [plugin.osgi.factory.OsgiPlugin] Unable to start the Spring context
for plugin com.almworks.jira.structure
```

In order for Structure plugin to work, it requires some of standard Atlassian plugins, such as the one that allows Structure to post to the Activity Streams. We have been reported of cases where these plugins cannot start because `-Datlassian.org.osgi.framework.bootdelegation` variable was set in `JAVA_OPTS` in `setenv.sh` (`setenv.bat`), as recommended in this comment to the Upgrade to JIRA 4.2 Guide. If you are using JIRA 5.0 or later, please try to remove the variable from `JAVA_OPTS` and see whether it resolves the problem.

If none of the above help resolve the problem, please contact ALM Works support.

5.6 After an Issue is Moved to Another Project, It Cannot Be Found in the Structure

5.6.1 Question

An issue was added to the structure. Afterwards, the issue was moved in JIRA to another project. Now, the issue cannot be found in the structure, either by summary, or by the new or old issue key. What happened?

5.6.2 Answer

Please check that the project where the issue was moved to is enabled for Structure. Structure plugin ignores issues in the projects that are not Structure-enabled, so the moved issue is ignored too, as if it ceased to exist.

If you need this issue in the structure, either include the project where the issue resides now into the list of Structure-enabled projects or move the issue to an already Structure-enabled project, e.g., to the original project.

5.7 User Cannot Access Structure, Although Permissions Have Been Granted

5.7.1 Question

Initially, the user (either a normal JIRA user or a JIRA administrator) could not access Structure plugin because it was not enabled for the user or not enabled in any project. A JIRA administrator has granted permissions for the user (by either adding her to the group that can access Structure or enabling the group the user belongs to for Structure access) or enabled Structure for some projects. However, the user still cannot see the Structure menu and cannot access any structure. How to resolve this problem?

5.7.2 Answer

Configured permissions related to Structure are cached on the server, so for a couple of minutes after the JIRA administrator makes changes to the permissions, the user may not be able to access Structure. These caches will last for approximately 5 minutes before they automatically refresh, after that the user will be able to use Structure.

There is a way to enforce cache refresh: the user should do a *hard refresh* of a JIRA page in their browser, after that they should be able to use Structure immediately. In most browsers, hard refresh is achieved by clicking the Refresh button while holding `Ctrl` or `Shift` button. There's a good list of ways to do a hard refresh in all popular browsers on Wikipedia: http://en.wikipedia.org/wiki/Wikipedia:Bypass_your_cache.

5.8 Issues Not Added to a Structure when Using Links Synchronizer or Import

5.8.1 Question

I'm trying to use Links synchronizer (Import) with link type X but the issues are not added to the structure.

5.8.2 Answer

Link synchronizer's ability to add issues to the structure is controlled by the **Scope** parameter.

- If you'd like to add **all** issues that have a link of type X to the structure, run Import with **Synchronize all issues** turned on.

  > Note that if you install a synchronizer (rather than run an Import) with **Synchronize all issues** on, it will continuously work both ways - removing issues from the structure will cause links to be removed. If you run a Resync and choose direction from Structure to Links, then all links of type X between issues that are not in the structure (but from projects enabled for Structure) will be **deleted**. If you Resync an empty structure to links with **Synchronize all issues** on, you'll effectively remove all links of that type.

- If you'd like to add **some** of the linked issues to the structure, you need to first add them via Search or Filter Synchronizer, and then run Import with **Synchronize issues that are already in the structure** selected. Use **Expand to...** options if you want the synchronizer to add missing sub-issues or parent issues to the structure.

See also: Links Synchronizer

5.9 Where to find JIRA Server ID

Structure license is tied to a particular JIRA Server and for generating a license for a server, a Server ID is required.

Server ID is a 16-digit code, that JIRA Administrator can look up in JIRA menu Administration | System Info or in Administration | Structure | License Details.

5.10 Integration with JIRA Agile (Greenhopper)

5.10.1 Question

We're using JIRA Agile (GreenHopper) - are there any conflicts with Structure? Can we see the Structure's hierarchy in JIRA Agile?

5.10.2 Answer

You can use JIRA Agile and Structure side by side. Structure plugin stores its data in a separate place, so it will not conflict with any other plug-in.

By default, Structure's hierarchy and issue order are independent from JIRA Agile's, but you can install a JIRA Agile (GreenHopper) Synchronizer to have Agile Rank synchronized with the position in the Structure and Epics synchronized with the positions of stories under epics in the Structure.

JIRA Agile displays the hierarchy of a selected issue in a separate Structure-provided tab in the issue details panel.

See also: JIRA Agile (GreenHopper) Synchronizer, Structure on Agile Boards

5.11 Using Subtasks and Structure

5.11.1 Question

Should I disable sub-tasks to use Structure?

5.11.2 Answer

Not necessarily. While Structure plugin can be a good replacement for sub-tasks, they can be used in parallel — for example, if you want to try Structure on a single project without affecting other JIRA users.

Structure treats sub-tasks as any other issues. You can also install a Sub-Tasks Synchronizer, which makes sure that JIRA sub-tasks are positioned under their JIRA parent issues.

5.12 Difference from Sub-tasks

5.12.1 Question

How is issue hierarchy provided by Structure plug-in different from the standard sub-tasks?

5.12.2 Answer

Sub-tasks have several major limitations:

- sub-tasks are only a one-level hierarchy;
- sub-tasks are separate issue types;
- sub-tasks always inherit project and security level from their parent task.

None of these limitations are present in Structure. At the same time, Structure plugin provides all the features that sub-tasks have, and more.

See also: Structure Widget Overview

5.13 Why Use Structure Plugin?

5.13.1 Question

Why use Structure plugin?

5.13.2 Answer

Structure plugin makes Atlassian JIRA a perfect tool for planning, mind mapping or task decomposition. Project managers can make extensive plans and monitor progress using JIRA issues. Project team members can break down tasks into smaller pieces for more precise estimation and reporting.

See also: Why Structure?

5.14 Some Link Synchronizer Operations Are Not Written to the History

5.14.1 Question

Link creation and removal operations, when performed by link synchronizers, are not written to the issue history and activity streams. Why?

5.14.2 Answer

They are not written because doing so may affect JIRA performance. If you want them to be written, please perform the following steps:

1. Add a new JIRA startup system property:
   `-Dstructure.bulkLinkProcessor.useLinkManager=true`

   > This page describes how to add JIRA startup properties.

2. Restart JIRA

5.15 Performance Considerations

For those, who have large JIRAs (hundreds of thousands of issues) there are a few things to bear in mind when working with the Structure.

The recommended limit for the number of issues in one structure is 100K and with this Structure already might be working noticeably slower, especially if there are many users working with the Structure Board at the same time.

So what we recommend is to distribute the issues between several smaller structures (5-10K issues per structure works perfectly) - the number of structures does not affect the performance that much.

Another thing that may affect the performance are the synchronizers. Incorrect synchronizers configuration may lead to conflicts when one synchronizer reverses the actions of the other, and vice versa. There is a safeguard mechanism that stops the cycle and sends warnings, but the next user action might trigger it again, so there's a possibility of wasted CPU cycles and overgrowth of the structure change history.

There were also several customer specific issues, which only reproduced in the customer's environment, but they were successfully resolved each time.

If necessary, you can also switch off some parts of the structure to reduce the load (for example, the Structure panel on the Issue Page) and limit the group of users Structure is exposed to.

5.16 How to restore the structure using History

Sometimes you might want to restore a structure to some previous state. For example, if it was incorrectly modified by some user, or if some synchronizer was not configured correctly and it did not work the way the user expected.

Here is what you can do in this situation:

1. Open the structure History panel.
2. In the history, find and select the moment when the structure was in the desired state (before the unwanted changes took place).
3. Press CTRL+A to select all issues and press CTRL+C to cut them to clipboard.
4. Switch off history panel and press CTRL+V – this should rearrange structure according to the view you selected in the history.

If you have some complicated synchronizers (for example, the ones, which use S-JQL in their configuration), it may be a good idea to temporarily disable the synchronizers before restoring and then enable them back and run the resync.

6 Structure Troubleshooting

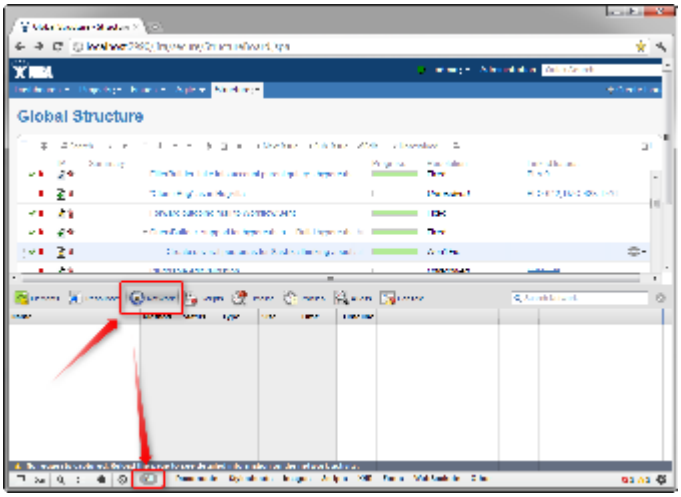
6.1 HAR Network Report

HAR Network Report is something we (ALM Works Support) may ask you to collect, to help us understand a tricky problem that we could not reproduce.

> HAR stands for HTTP Archive Format, a text-based format for the log of network communications between a user agent (the browser) and a web server. You can also use this report with a HAR Viewer for in-depth analysis of your JIRA page load performance. (Be aware though that with an online viewer you may transfer sensitive or security-related information to a third party.)

6.1.1 Collecting HAR Report with Google Chrome

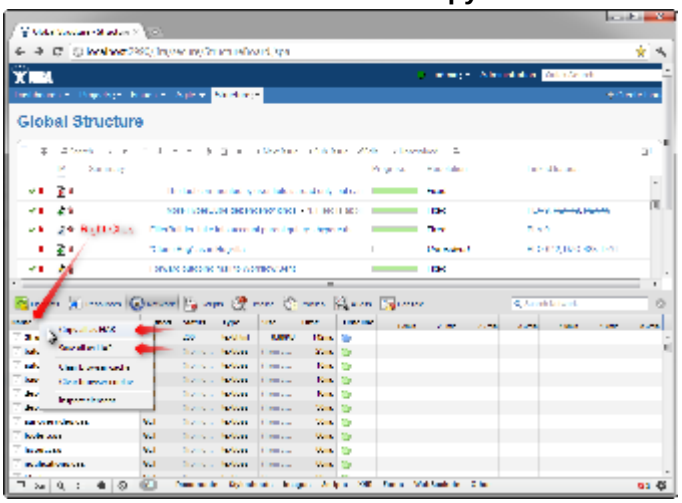
1. Open a new Chrome window and navigate to the page where the problem happens.
2. Press **Ctrl+Shift+I** or use menu **Wrench | Tools | Developer Tools** to open a section with developer tools. Switch to the **Network** tab there. Make sure **All** tab is selected below.



3. Reload the page by using **Ctrl+R** or clicking the Reload button. This will make Network tab log all network exchange during page load.

> The network tab will start collecting information or network exchange automatically after it's opened. If you know that the problem is not related to the initial page load, you may skip this step to avoid adding extra data to the log. If unsure, reload the page to collect the full report.

4. Reproduce the problem being analyzed.
5. After the problem has been reproduced, **right-click** on the **Name** column in the Network tab and choose either **Save all as HAR** or **Copy all as HAR**



6. Paste the report into an e-mail to our support, or attach the saved .HAR file.

6.2 Collecting Performance Snapshots

Performance snapshots allow ALM Works support team to analyze performance-related problems on your JIRA server without direct access to it.

6.2.1 Download and install Atlas-Yourkit plugin.

Get the latest version from this page. In JIRA 4.3 and later, you can install this plugin without JIRA restart.

The performance analysis plugin and redistributed parts of YourKit profiler are free, but if you'd like to analyze the performance snapshots yourself, you'll need to obtain YourKit license and download YourKit software (they provide a free evaluation period).

Name	Version	Date
atlas-yourkit-0.2.jar	1	2012-01-11 13:58

6.2.2 Load Profiling Agent

1. Open menu **Administration | Troubleshooting and Support | YourKit Profiling** (hint: in JIRA 4.4 and later versions, press **g,g** ("g" twice) and search for "yourkit").

2. If agent is already loaded, you'll see profiling controls - skip this step then.
3. Click **Load Agent** to load profiling agent. You'll need to have JDK installed. If you don't have JDK installed – follow the link on that page, download and install a matching JDK on JIRA host. It is not necessary to restart JIRA, just install the JDK and load agent.

There's certain risk that JVM will crash when loading profiling agent into JVM. A safer method of loading profiling agent is by changing JIRA start-up parameters (in `setenv.sh`/`setenv.bat`) and specifying `agentpath` parameters with other options. See YourKit Documentation for details.

6.2.3 Capturing CPU Performance Snapshot

After profiling agent is loaded, you can click **Start CPU Sampling** on the YourKit page, then perform the actions that make JIRA slow, or wait for some time to collect the statistics. When finished, click **Stop CPU Sampling**. Performance snapshot will be saved to a directory within your JIRA Home, and the path will be shown on the YourKit page.

6.2.4 Capturing Memory Snapshot

Click "Take Memory Snapshot" - memory dump will be collected and saved in a file under your JIRA Home. Do not take memory snapshots unless you need to!

Taking memory snapshot is usually a long operation, which could last several minutes. During that time JIRA will be completely frozen. Make sure you've got enough disk space (several GBs). Don't panic - it does take that much time. After you click the button the page will be reloading. The browser may fail to load the page due to timeout - check JIRA logs to see when snapshot is finished.

6.2.5 Sending the Snapshots to Support Team

By default, snapshots are written into `<jira_home>/yourkit/snapshots` directory. Locate it and create a ZIP archive of all relevant snapshot files.

Please send the ZIP to us as described here: Sending Files to Support Team.

6.2.6 After Profiling Session

There's no way to unload the profiling agent. You may want to continue running JIRA with the profiling agent loaded, since it does not product much overhead. (Make sure you have stopped all the monitoring.)

For a safer / cleaner environment, you can restart JIRA. (If you made additional effort to enable profiler agent in `setenv` script, you'll need to comment that options out.)

6.2.7 Performance Snapshot Without Yourkit Plugin

Performance Profile allows ALM Works support team to analyze performance-related problems on your JIRA server without direct access to it.

We are using Java Profiler product called YourKit. In order to collect the profile, you'll need to download freely distributed "agent" library, connect it to your JIRA instance and capture a performance snapshot. You will need to purchase a license from YourKit only if you want to analyze the captured profile yourself.

No special knowledge is required to collect the performance profile, but being familiar with using the command-line on the server that runs JIRA helps.

Download Profiling Agent

Download the ZIP with profiling agent from here: jira-profiler-v1-yjp956.zip `md5sum e3ea2b72ef4b22584c641425275050d0`

Unpack the downloaded ZIP file into the directory where you have JIRA installed (**not** JIRA home!). This will create `<jira_install>/profiler` directory under your JIRA installation path.

You can unpack the profiler into any other directory, but this instructions and our scripts assume that the profiler is unpacked into JIRA install dir.

If you will be able to restart JIRA before profiling, this is all you need — you can proceed to restarting JIRA with Profiling.

Additional Download to Profile JIRA Without Restart

If you need to profile JIRA without restarting it first (and assuming it is not already started with a profiler agent), you will need to download full distribution of the YourKit Java Profiler:

1. Open http://yourkit.com/download/index.jsp
2. Click on **ZIP Archive** type of download - **NOT** the installer! ZIP archive is typically downloaded under "Solaris" section - it is the correct link even if you run JIRA on Windows.
3. License key is not required for our purpose! Do not request evaluation license. (Unless you intend to do an evaluation of YourKit, of course.)
4. Unpack the downloaded ZIP into `<jira_install>/profiler` – this is the directory created at step 1. Unpacking will create a sub-directory there - for example, `<jira_install>/profiler/yjp-9.5.6`.

Restart JIRA with Profiling

If you need to profile without restart, skip this step.

The following instruction is provided for a standalone JIRA installation.

To restart JIRA with profiling, you need to pass additional options to Java that runs JIRA. This is done by editing `<jira_install>\bin\setenv.bat` on Windows or `<jira_install>/bin/setenv.sh` on a Unix-based OS and pointing Java to a profiler agent that you have unpacked at step 1.

1. Find out which profiler agent to use.
    a. Look into `<jira_install>/profiler/bin` directory. Typically there will be two sub-directories for your operating system: 32-bit and 64-bit. The bitness must match the bitness of JVM that runs JIRA. You can verify which Java your JIRA runs on if you open **Administration | System Info** in JIRA and look for "Java VM". If it mentions "64-Bit", then JIRA runs on a 64-bit Java.
    b. Note the name of the subdirectory under `profiler` directory that corresponds to the bitness of target JVM: it may be *win64* or *linux-x86-32* or something like that.
2. Edit `setenv` script:
    a. On Windows, set or append the following parameters to `JVM_SUPPORT_RECOMMENDED_ARGS` in `<jira_install>\bin\setenv.bat` (following is a single long line):

```
set
JVM_SUPPORT_RECOMMENDED_ARGS=-agentlib:%~dp0..\profiler\bin\win64\yjpagent=port=10001,
-XX:MaxPermSize=500m
```

    b. On other OS, set or append the following parameters to `JVM_SUPPORT_RECOMMENDED_ARGS` in `<jira_install>/bin/setenv.sh` (following is a single long line):

```
JVM_SUPPORT_RECOMMENDED_ARGS="-agentpath:`dirname
\"$0\"`/../profiler/bin/linux-x86-64/libyjpagent.so=port=10001,onlylocal,dir=`dirname
\"$0\"`/../profiler/snapshots,delay=20000 -XX:MaxPermSize=500m"
```

3. Note that in the lines above, you should change **win64** or **linux-x86-64** to the name of the directory where the correct profiler agent for your OS/Java is located.
4. You may also need to change **port=10001** to make profiling agent listen on some other TCP port - in case port 10001 is already taken.
5. Stop JIRA and start it again.
6. Watch `<jira_install>/logs/catalina.out` for YourKit message like *[YourKit Java Profiler 9.5.6] Loaded.*

Use Copy & Paste to copy the parameters and then edit them in the setenv.sh

If the parameters are set incorrectly, JIRA start may fail. Verify that you have specified the agent directory correctly. Also verify that `<jira_install>` directory path does not contain spaces.

Profiler agent will use directory `<jira_install>/profiler/snapshots` to write performance snapshots - it must be write-accessible to the JIRA process.

Now you can proceed to #Running Profiling Session.

Attach Profiler Agent to JIRA without Restarting

If you have restarted JIRA with profiling, skip this step.

If possible, restart JIRA with profiling instead of attaching profiler agent on the fly.

You will need the full distribution of YourKit downloaded at step 1.1. You will need to run a Java program as specified below - with the same version of Java that JIRA runs on. We assume that it is in your PATH variable in the command-line, but if it's not - you need to specify a full path to `java`.

1. Find out the process ID of the process that runs JIRA. You can use `jps` command from the Java distribution.
2. Find out the location of JDK (Java Development Kit). If you don't have JDK installed (only JRE), this procedure won't work. Typically JDK home is stored in the command-line environment variable JAVA_HOME.
3. Change current directory to `<jira_install>/profiler/yjp-9.5.6`. (You may have a different version of yjp.)
4. Run the following command, substituting JIRA process ID instead of ***PID***.
    a. On Windows:

    ```
    java -cp lib\yjp.jar;%JAVA_HOME%\lib\tools.jar com.yourkit.Main -attach PID
    port=10001,onlylocal,dir=<jira_install>\profiler\snapshots
    ```

    Replace <jira_install> with the full path of the JIRA installation folder.
    b. On other OS:

    ```
    java -cp lib/yjp.jar:$JAVA_HOME/lib/tools.jar com.yourkit.Main -attach PID
    port=10001,onlylocal,dir=`pwd`/../snapshots
    ```

The command should output something like this:

```
Attaching to process 60108 using options
port=10001,onlylocal,dir=..\snapshots The profiler agent has attached.
Waiting while it initializes... The agent is loaded and is listening on port
10001. You can connect to it from the profiler UI.
```

Running Profiling Session

To successfully run a profiling session, you need to have JIRA running with a profiling agent, as explained above. The agent does not add much overhead when being idle — it sits there waiting for your commands to start a profiling session.

General Procedure

The profiling session is controlled by sending commands to the profiling agent (within the JIRA process). The program that is used to send the commands is `yjp-controller-api-redist.jar`, located in `<jira_install>/profiler`. The common format for running this program is:

```
java -jar yjp-controller-api-redist.jar localhost 10001 <command>
```

The *<command>* is replaced with some actual command, and if you changed the default port of the agent from 10001 to something else, you need to specify that port number here instead of 10001. This command should be run from `<jira_install>/profiler` directory.

We are assuming that `java` is on your PATH. If not the case, use the full path to `java` executable.

CPU Performance Analysis

If JIRA is unresponsive or burns CPU extensively, you can run CPU analysis session.

1. Start session with the following command:

   ```
   java -jar yjp-controller-api-redist.jar localhost 10001 start-cpu-sampling
   ```

2. Let JIRA work for some time. If needed, take a specific action that causes the problem to manifest.
3. Stop session and record a snapshot:

   ```
   java -jar yjp-controller-api-redist.jar localhost 10001 capture-performance-snapshot
   ```

Sending the Snapshots to Support Team

By default, snapshots are written into `<jira_install>/profiler/snapshots` directory. Locate it and create a ZIP archive of all relevant snapshot files. If the ZIP is less than 10 Megabytes, it's ok to send it to us by e-mail.

If the ZIPPed snapshot is 10 MB or larger, you need to use FTP to send it over to us:

1. Use any FTP client (`ftp` or `lftp` from the command line).
2. Connect to host `f.almworks.com`
3. Use login name `almftp` and password `almftp`
4. Upload files to the root folder.
5. After the upload is finished, please send us an e-mail with a notification that you have uploaded the snapshots.

You will not be able to list or download files from that FTP, and your FTP client may show errors about that. That's ok and should not prevent you from uploading snapshots.

After Profiling Session

You may want to continue running JIRA with the profiling agent loaded, since it does not product much overhead. Make sure you have stopped all the monitoring.

For a safer / cleaner environment, you can restart JIRA with the profiling options in `setenv` script commented out.

6.3 Troubleshooting Synchronizers

Structure synchronizers work in background and can lead to changes in the structures or issue data that might be hard to trace. Complex configuration rules don't make things better, so it's important for JIRA admin to be able to track which synchronizers are doing what and what has caused a particular change a user is complaining about.

6.3.1 Log Files

To get detailed reports about what's going on, you can reconfigure your JIRA logging so that structure synchronizers can produce more verbose messages. Also, you might want to direct messages from the synchronizers into separate log files.

The appearance of detailed synchronizer messages is governed by the log level: the lower the log level, the more detailed messages can appear. By default, log level for structure synchronizers is `WARN`, and you can set it to lower levels, like `DEBUG` (the lowest one.) You can set the logging level either temporarily (until the next JIRA restart) or permanently.

To see the list of possible log levels and other general information regarding logging in JIRA, please refer to JIRA logging documentation.

Temporarily change log level for structure synchronizers

If you set log level in this way, it will not persist after you restart JIRA. This is a relatively simpler way than setting the log level permanently.

1. Log in as a user with the JIRA System Administrators global permission.
2. Select *Administration* | *System* | *Troubleshooting and Support* | *Logging & Profiling* (tab). The 'Logging' page will be displayed, which lists all defined log4j categories (as package names) and their current logging levels.
3. Locate and click the link that reads "Configure logging level for another package", and a dialog will be displayed. For troubleshooting bundled synchronizers, specify package name **`com.almworks.jira.structure.ext`**; choose the appropriate logging level, e.g. `DEBUG`.

Permanently change log level for structure synchronizers or set up separate log files for synchronizers

This way, you need to modify the `log4j.properties` file, which is located in the JIRA installation directory .

The package name that all bundled synchronizers log under is `com.almworks.jira.structure`. You can add the following lines to have debug messages from synchronizers show on the console and/or in the log file (depending on their respective log levels):

```
log4j.logger.com.almworks.jira.structure = DEBUG, console, filelog
log4j.additivity.com.almworks.jira.structure = false
```

Or, you can set up a separate log file for synchronizer actions:

```
log4j.appender.structure-sync=com.atlassian.jira.logging.JiraHomeAppender
log4j.appender.structure-sync.File=structure-sync.log
log4j.appender.structure-sync.Threshold=TRACE
log4j.appender.structure-sync.MaxFileSize=20480KB
log4j.appender.structure-sync.MaxBackupIndex=1
log4j.appender.structure-sync.layout=org.apache.log4j.PatternLayout
log4j.appender.structure-sync.layout.ConversionPattern=%d %t %p %X{jira.username} [%c{4}] %m%n

log4j.logger.com.almworks.jira.structure = DEBUG, structure-sync, console
log4j.additivity.com.almworks.jira.structure = false
```

Synchronization Undo Files

Structure keeps synchronization undo files for possible script processing in case massive changes are erroneously effected on issue data. The undo files contain actions that should be taken, in reverse order, to

bring issue database to the original / previous state. However, undo files do not contain changes for the structures - only for the issue data.

See Structure Files Location for information about where syncundo files are located.

In some cases undo files may be preferred to the logs as they more clearly indicate the changes.

6.4 Troubleshooting Database

Structure Plugin uses embedded Apache Derby database to store its data. The database files are located in JIRA home directory and are crucial for the plugin operation.

6.4.1 Database Check

If you suspect that Structure's database may have failed, please go through the following check list before calling in support. Usually it's one of those problems.

> For brevity, we'll use Unix-like path names with "/" as the separator. On Windows, the path names are delimited with "\". We'll also write `$JIRA_HOME` to refer to the JIRA home directory.

- Is there enough disk space on the file system that contains `$JIRA_HOME` and `$JIRA_HOME/structure/db`?
- Does the user account, under which JIRA process is running, have full access to all the content of `$JIRA_HOME/structure`?
    - Check file and directory ownership
    - Check file and directory permissions
    - JIRA must be able to read/write files, create files, delete files. Full access.
- Is file system that hosts `$JIRA_HOME/structure` mounted read-write?
    - Try creating a dummy file in `$JIRA_HOME/structure`.
- Do you not have any other applications that might access this JIRA's `$JIRA_HOME/structure` and lock it?
    - For example, aren't you running two JIRA instances on the same JIRA home?

6.4.2 Symptoms of Database Problems

Structure Plugin fails to start, gets Disabled

Observation: You start JIRA or enable Structure Plugin, but it immediately gets disabled.

Explanation: Structure shuts down and disables itself if it cannot properly start the database. Check the server logs for the following message:

```
Structure Plugin failed to start its database and will be DISABLED in a few seconds.
...
================================================================================
Structure Plugin failed to start and was automatically DISABLED.
Error message: ....

Please fix the problem and enable the plugin again through the Plugin Manager.
If the problem persists, please contact support.
================================================================================
```

The users cannot make any changes in Structure, getting errors

Observation: The structure is displayed correctly in the browser, but any attempt to change the hierarchy results in red error messages at the bottom of the screen, saying there was a problem on the server.

Explanation: It is possible that the database has shut down or become read-only due to one of the reasons mentioned above. Since Structure has some information in the server-side cache, it still can serve certain requests.

Check the logs for the following message:

```
================================================
Structure Plugin storage problem: ....
================================================
```

6.4.3 Proposed course of actions

1. Go through the checklist above to locate the problem with the database. If needed, request support from support@almworks.com
2. When the problem is cured, try disabling the plugin (if it's not disabled yet) and enabling it back again.
3. If that does not help, try restarting JIRA.

6.5 Sending Files to Support Team

When you need to send files to ALM Works support team, please use one of the following methods (listed in the order of preference).

6.5.1 Attach to the Support Request in ALM Works JIRA

File size limit: 50 MB

If the files pertain to a Support Request on https://jira.almworks.com, please use JIRA to upload and attach the files to the issue. Size limit is 50 MB per upload.

Please note that normally Support Request issues have "Protected" issue security level, which means that only the reporter and ALM Works have access to that issue (and attached files). However, if for some reason you don't see "Security Level: Protected" on the issue page, it means that the issue can be seen by anyone, and any attached files can be downloaded by anyone. Please keep this in mind – ALM Works is not responsible for public disclosure of any information entered into a publicly-visible issue, including information in the attachments.

If the issue is public and you'd like to change the security level to Protected, please comment on that issue and we'll do that shortly.

6.5.2 Send Files by E-mail

File size limit: 20 MB

You can send the files to support@almworks.com. Maximum total attachments size is 20 MB.

If you don't have a preceding e-mail communication with support about the problem in question, please add a short comment or a reference to the problem being diagnosed.

6.5.3 Upload Files via FTP

File size limit: 1 GB

Please use FTP only to upload large files.

1. Use any FTP client (`ftp` or `lftp` from the command line).
2. Connect to host `f.almworks.com`
3. Use login name `almftp` and password `almftp`
4. Upload files to the root folder.
5. After the upload is finished, please send us an e-mail with a comment about what you have uploaded.

The files you have uploaded are safe – they cannot be downloaded by anyone except ALM Works support.

You will not be able to list or download files from that FTP, and your FTP client may show errors about that. That's ok and should not prevent you from uploading files.

6.6 Structured JQL Troubleshooting

If a Structured JQL query doesn't work as expected, please try the following steps.

1. Double-check if the query itself correctly expresses what you are searching for. Feel free to ask a question on Atlassian Answers or write to support@almworks.com if you need help with S-JQL.
2. Probably, JIRA indexes that are used for searching have become corrupt. Please try to do a full reindex of JIRA — note that you should use **Lock JIRA and rebuild index** option, the other one is known to not help when indexes are corrupted.
3. If the query still returns strange results, please go to the Structured JQL Troubleshooting page and follow the instructions outlined there:

   ```
   <base URL>/secure/StructuredJqlTroubleshooting.jspa
   ```

   Here, `base URL` refers to the JIRA base URL.
   On this page, you will be able to run a Structured JQL query and collect extensive logs which we in ALM Works can inspect in order to track down the issue.

6.7 Collecting Support Zip

ALM Works support may ask you to collect a Support Zip during a support case investigation.

To collect Support Zip, you will need **System Administrator** permissions in your JIRA. You will also need a way to transfer files from the host that runs JIRA instance.

If you do not have the required access, please ask your JIRA administrator or your system administrator for assistance.

To collect a Support Zip:

1. Open **Administration | System | Logging and Profiling** page.
   a. Enter STRUCTURE TROUBLESHOOTING into the **Optional Message** field, turn on **Log Rollover** and press **Mark**.
   b. Scroll down and click **Configure logging level for another package**, enter package name *`com.almworks.jira.structure`* then select logging level DEBUG and click **Add**.
2. Reproduce the problem being investigated.
3. Open **Administration | System | Atlassian Support Tools**, switch to **Support Zip** tab. Select options **Application Properties**, **Thread Dump**, **JIRA Application Logs**, **Tomcat Logs**. Unselect all other options. Click **Create**.
4. Use access to the system that hosts JIRA to get the support zip file. If the file is larger than 100 MB, please create the support zip again but also turn on option **Limit File Sizes.**
5. Send the resulting ZIP file to ALM Works support by email or attach it to the support request in ALM Works JIRA.

6.8 Alternative Structure Gadget for IE8 and IE9

This article applies to JIRA 6.0 and later.

There is a known problem that Structure gadget (either added to a JIRA dashboard or a Confluence page) is not displayed properly when viewed in Internet Explorer 8 or 9. For that case, Structure is shipped with alternative gadgets which work in these and all modern browsers. This article describes how to enable and use the alternative gadgets.

Temporary Solution Warning

These gadgets are supplied as a temporary solution for Internet Explorer 8-9 users. **Once JIRA discontinues support of these browsers in one of its future versions, we will remove them in the corresponding Structure version** in favor of the all-purpose general gadget. When upgrading to that future version, you'll need to recreate all alternative gadgets with the general one.

6.8.1 Enable alternative gadgets

There are several kinds of alternative gadgets, one for each JIRA version. By default, all alternative gadgets are disabled. You will need to enable the one that works with the version of your JIRA.

To enable a gadget, please do the following:

1. Open **Administration** | **Add-ons | Manage Add-ons**.
2. Locate Structure plugin and expand its row.
3. Click the link that looks like the following: "179 of 182 modules enabled".
   Use the Search feature of your browser to locate the gadget by its name or unique ID. Determine the appropriate name by the following table:

JIRA version	Gadget name	Unique
6.0–6.0.8	`gadget:Structure (IE 8-9 Compatible, Works with JIRA 6.0.x)`	`struct`
6.1–6.1.7	`gadget:Structure (IE 8-9 Compatible, Works with JIRA 6.1.x)`	`struct`

   There is no gadget compatible with JIRA 6.2. We are looking into ways to provide it; to be notified of the progress on it, watch/vote this issue in our JIRA: HJ-1703-Make gadget accessible from IE 8-9 on JIRA 6.2/JIRA 6.3(Open)

4. Click the Enable button to the right of the module name. (Should you later need to disable the gadget, you'd need to click the Disable button.)

It is recommended to enable only the gadget appropriate for your JIRA version. A gadget designed for other JIRA version will not work in most cases — users will see empty space or a piece of code in place of the gadget. (All other JIRA functionality, including Dashboard, is not affected.)
This is necessary to consider when upgrading your JIRA.
So, for example, if you first enable the alternative gadget on JIRA 6.0, then when you later upgrade to JIRA 6.1, the gadget will stop working. You will need to enable the gadget for JIRA 6.1 and recreate all of the existing gadgets. Afterwards, it is recommended to disable the gadget for JIRA 6.0.

5. It is recommended to disable the general gadget, so that you don't accidentally use it. To do that, on the same page locate the gadget by name (`gadget:Structure`) and click the Disable button to the right of it.
6. Go to a JIRA dashboard and check that you can add the enabled gadget. The alternative gadgets are named "Structure (IE Compatible)".