Structure Plugin for JIRA

Documentation

Author: Igor Sereda

Version: 1

Date: Jun 20, 2017 12:08 AM

Table of Contents

1 Structure User's Guide 11

1.1 Basic Concepts 11
1.1.1 Default Structure 12
1.1.2 Favorite Structures 12

1.2 Structure Menu 13

1.3 JIRA Pages with Structure 15
1.3.1 Structure Board 15
1.3.2 Structure on the Issue Page 17
1.3.3 Structure Gadget 22
1.3.4 Structure on the Project Page 30
1.3.5 Structure on Agile Boards 31

1.4 Working with the Structure Widget 32
1.4.1 Structure Widget Overview 33
1.4.2 Navigating Structure 35
1.4.3 Main Structure Toolbar 38
1.4.4 Configuring View 40
1.4.5 Widget Columns 47
1.4.6 Searching and Filtering 126
1.4.7 Transformations 132
1.4.8 Two-Panel Mode 139
1.4.9 Changing Structure 141
1.4.10 Working with Issues 151
1.4.11 Viewing History of a Structure 166
1.4.12 Full Screen Mode 169
1.4.13 Printing Structure 171
1.4.14 Exporting Structure to XLS (Excel) 172
1.4.15 Real-Time Collaboration 175

1.5 Automation 176
1.5.1 Types of Generators 176
1.5.2 How to Add a Generator 176
1.5.3 How to Edit a Generator 177
1.5.4 How to Remove a Generator 178
1.5.5 Generators 178
1.5.6 Generators Options 188

1.6 Managing Structures 190
1.6.1 Locating a Structure 191
1.6.2 Structure Details 192
1.6.3 Creating New Structures 193
1.6.4 Structure Permissions 194
1.6.5 Customizing View Settings 196
1.6.6 Copying a Structure 198
1.6.7 Archiving a Structure 207
1.6.8 Deleting a Structure 208

1.7 Managing Views 209
1.7.1 Locating a View 210
1.7.2 Changing View Settings 211
1.7.3 View Sharing and Permissions 213
1.7.4 Associating Views with Structures 215
1.7.5 Copying a View 215
1.7.6 Deleting a View 215

1.8 Template Structures and Projects 216
1.8.1 Configuring Template Structures 216
1.8.2 Creating Issues and a Structure from Template 216
1.8.3 Template Projects 217

1.9 Sharing a Perspective 217

1.10 Synchronization 219
1.10.1 Importing Structure 220
1.10.2 Exporting Structure 221
1.10.3 Installing Synchronizer 223
1.10.4 Modifying Synchronizer 224
1.10.5 Removing Synchronizer 225
1.10.6 Turning Synchronizer On and Off 226
1.10.7 Running Resync 226
1.10.8 Synchronization and Permissions 228
1.10.9 Protection from Synchronizer Cycles 229
1.10.10 Bundled Synchronizers 229

1.11 Structure Activity Stream 249
1.11.1 Available Filters 249
1.11.2 Reading Activity Stream 250
1.11.3 Activity Streams Performance 251

1.12 Structured JQL 252

1.12.1 S-JQL Cookbook 252
1.12.2 S-JQL Reference 258
1.12.3 structure() JQL function 282

1.13 Keyboard Shortcuts 285
1.13.1 Keyboard Shortcuts (PC) 285
1.13.2 Keyboard Shortcuts (Mac) 287
1.13.3 Quick Action Lookup 289

1.14 Getting Help 290

2 Structure Administrator's Guide 292

2.1 Installing Structure 292
2.1.1 Migrating Data from Structure 2 to Structure 3 293
2.1.2 Memory Guidelines 294
2.1.3 Uninstalling and Reinstalling Structure 297
2.1.4 Upgrading and Downgrading 298

2.2 Setting Up Structure License 299
2.2.1 Setting Up Evaluation License 299
2.2.2 Licenses from ALM Works and from Atlassian 300
2.2.3 Purchasing a Commercial License 301
2.2.4 Migrating Licenses 302
2.2.5 Structure License Parameters 302
2.2.6 When Structure is Available for Free 303
2.2.7 License Maintenance and Expiration 303

2.3 Getting Started with Structure 304

2.4 Selecting Structure-Enabled Projects 305

2.5 Global Permissions 306
2.5.1 Who Has Access to the Structure 306
2.5.2 Restricting User Access to Structure 306
2.5.3 Changing Permission to Create New Structures 307
2.5.4 Changing Permission to Manage Synchronizers 308
2.5.5 Changing Permission to Access Automation 309

2.6 Changing Structure Defaults 310
2.6.1 Initial Configuration 310
2.6.2 Changing Default Structure 311
2.6.3 Changing Default View Settings 312

2.6.4 Changing Default Options for the Issue and Project Pages 312

2.7 Structure Backup, Restore and Migration 313
2.7.1 Using Structure Backup 314
2.7.2 Backing Up Structure 314
2.7.3 Restoring Structure from Backup 315
2.7.4 Migrating Structures 316

2.8 Automatic Structure Maintenance 318
2.8.1 Automatic Structure Maintenance 318
2.8.2 Maintenance Tasks 319
2.8.3 Running Maintenance Tasks Manually 321

2.9 Workflow Integration 321
2.9.1 Structure Workflow Validator 321
2.9.2 Structure Workflow Condition 323

2.10 Anonymous Usage Statistics 323
2.10.1 Viewing Current Statistics 324
2.10.2 Turning Anonymous Usage Statistics On and Off 324

2.11 Structure Files 324
2.11.1 $JIRA_HOME/structure 324

2.12 Turning Off Optional Features 325

2.13 Advanced Configuration with System Properties 326
2.13.1 Setting System Properties on Startup 326
2.13.2 Setting System Properties with Script Runner 326
2.13.3 Synchronizer Cycle Guard 327
2.13.4 Structure size limit 329

2.14 System Requirements 330
2.14.1 Atlassian Platform 330
2.14.2 Databases 330
2.14.3 Browsers 330
2.14.4 Server Requirements 331
2.14.5 Non-Conforming systems 331

2.15 Best Practices 331
2.15.1 Backup Strategy 332
2.15.2 Gradual Deployment 333

2.16 Dark Features 334
2.16.1 Alternative initial values for project/type when creating an issue in dialog 334

3 Structure Developer's Guide 335

3.1 Structure Developer Documentation 335

3.2 Structure Concepts, Developer's Perspective 336
3.2.1 1. Basic Concepts Overview 336
3.2.2 2. A Note on Extensibility 337

3.3 Accessing Structure from JIRA Plugin 337
3.3.1 Setting Up the Integration 337
3.3.2 Structure Services 342
3.3.3 Building Forest Specification 344
3.3.4 Reading Structure Content 345
3.3.5 Changing Structure Content 348
3.3.6 Loading Attribute Values 352
3.3.7 Creating and Adding Folders 354
3.3.8 Creating Dynamic Structures 355

3.4 Extending Structure Functionality 356
3.4.1 Creating a New Column Type 357
3.4.2 Creating a New Synchronizer 377
3.4.3 Loading Additional Web Resources For Structure Widget 378

3.5 Accessing Structure Data Remotely 379

3.6 Reference 380
3.6.1 Structure Developer Reference 380
3.6.2 Structure Java API Reference 380
3.6.3 Structure Plugin Module Types 382
3.6.4 Structure REST API Reference 389
3.6.5 Structure JavaScript API Reference 422
3.6.6 Web Resource Contexts 436

3.7 API Usage Samples 437
3.7.1 Download 437
3.7.2 Example List 438

3.8 Structure 3 API Changes 438
3.8.1 1. State of the API 438
3.8.2 2. Conceptual Changes 439
3.8.3 3. REST API 441
3.8.4 4. Java API 443

4 Structure FAQ 447

4.1 Frequently Asked Questions 447

4.2 Cannot Create an Issue With +Next Issue (+Sub-Issue) Because of the Required Fields 447

4.2.1 Question 447

4.2.2 Answer 447

4.3 Plugin Manager Says Structure Is Unlicensed 448

4.3.1 Question 448

4.3.2 Answer 448

4.4 No Check Mark Displayed for a Resolved Issue 448

4.4.1 Question 448

4.4.2 Answer 448

4.5 Structure plugin won't start 450

4.5.1 Question 450

4.5.2 Answer 450

4.6 After an Issue is Moved to Another Project, It Cannot Be Found in the Structure 452

4.6.1 Question 452

4.6.2 Answer 452

4.7 User Cannot Access Structure, Although Permissions Have Been Granted 453

4.7.1 Question 453

4.7.2 Answer 453

4.8 Issues Not Added to a Structure when Using Links Synchronizer or Import 453

4.8.1 Question 453

4.8.2 Answer 453

4.9 Where to find JIRA Server ID 454

4.10 Integration with JIRA Agile (Greenhopper) 454

4.10.1 Question 454

4.10.2 Answer 454

4.11 Using Subtasks and Structure 455

4.11.1 Question 455

4.11.2 Answer 455

4.12 Difference from Sub-tasks 455

4.12.1 Question 455
4.12.2 Answer 455

4.13 Some Link Synchronizer Operations Are Not Written to the History 456
4.13.1 Question 456
4.13.2 Answer 456

4.14 Performance Considerations 456

4.15 How to restore the structure using History 457

4.16 Can I export a structure to Microsoft Word so that it can be emailed as a document? 457

5 Structure Troubleshooting 459

5.1 Collecting Support Zip 459

5.2 HAR Network Report 459
5.2.1 Collecting HAR Report with Google Chrome 460

5.3 Troubleshooting Synchronizers 461
5.3.1 Structure Audit Log 461
5.3.2 Log Files 461

5.4 Structured JQL Troubleshooting 463

5.5 Collecting Performance Snapshots 463
5.5.1 1. Download and install Atlas-Yourkit plugin. 463
5.5.2 2. Load Profiling Agent 463
5.5.3 3. Capturing CPU Performance Snapshot 464
5.5.4 4. Capturing Memory Snapshot 464
5.5.5 5. Sending the Snapshots to Support Team 464
5.5.6 6. After Profiling Session 464
5.5.7 Performance Snapshot Without Yourkit Plugin 465

5.6 Sending Files to Support Team 470
5.6.1 1. Attach to the Support Request in ALM Works Service Desk (Preferred) 471
5.6.2 2. Send Files by E-mail 471
5.6.3 3. Upload Files Directly to Our Server 471

5.7 Alternative Structure Gadget for IE8 and IE9 471
5.7.1 Enable alternative gadgets 472

5.8 Troubleshooting Performance and Stability Issues 473

5.8.1 1. Thread Dumps 474
5.8.2 2. Verbose Logging 474
5.8.3 3. Support Zip 475
5.8.4 4. Browser Console Log 475
5.8.5 5. HAR Report 475
5.8.6 6. Screenshots or Video 476

Links to the available documentation collections:

Download documentation:

Name	Version	Date

1 Structure User's Guide

This section contains information for Structure users.

Contents:

1.1 Basic Concepts

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

Structure add-on	That's the product! We differentiate between Structure add-on and actual structures by writing plugin name with a capital letter.
a structure	A single hierarchical list of issues.
a structure contains an issue	A structure is initially empty - it does not contain any issues. Issues can be added to a structure manually (see page 142) or automatically with the help of Automation or Synchronizers (see page 219). 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. Structure widget is present at several places in JIRA (see page 15).
a view	Configuration of columns in the structure widget
a sub-item is under a parent item	When you place an item under another item in the hierarchy, you can say that it's a sub-item and the other one is its parent item. A sub-item may contain sub-items on its own (and so it's a sub-item and parent item at the same time). An item that's a sub-item in one structure may not be a sub-item in another structure.
children items	Sometimes we call sub-items "children items", but "sub-items" is the preferred term.

JIRA has sub-tasks	It's important to distinguish JIRA sub-tasks from Structure sub-items. We will use "sub-tasks" to refer to the issues of the special issue type that you can create in JIRA. They can be shown as sub-issues inside the structure, but since there are no restrictions on issue types in Structure, they can be placed anywhere just like other types of issues and other items.
item – sub-item relationship	Structure can represent any kind of relationship between a parent item and a sub-item. When speaking of issues, 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.

1.1.1 Default Structure

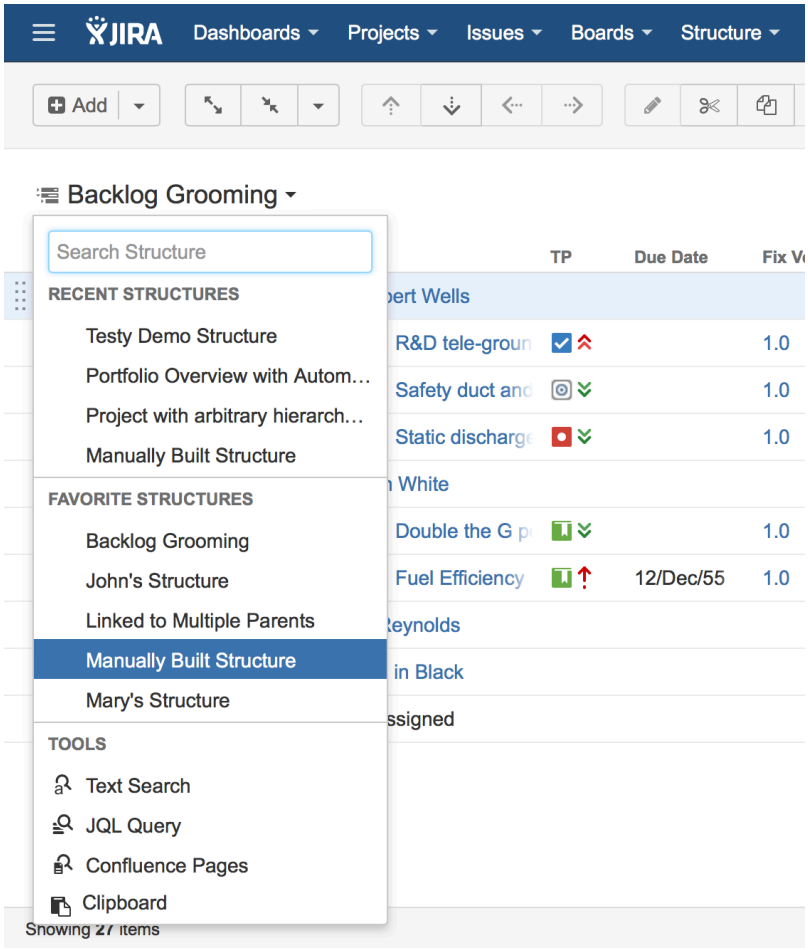
Default structure is displayed to the user when there's no specific structure selected - by default. JIRA administrator can change the system default structure (see page 311).

For the Issue Page (see page 21) and Project Page (see page 30) you can define which structure should be used as default. A **project-level default structure** can be specified for a project that is enabled for Structure on the Defaults (see page 311) page.

1.1.2 Favorite Structures

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

As you switch between structures in the Structure widget, you can see top 5 of the favorite structures ordered alphabetically by name.



To manage your favorite structures, use the Manage Structure (see page 190) page. To make a structure your favorite, click a white star (☆) near the name of that structure. The star will colored (★) to indicate that the structure was added to the list of your favorite structures.

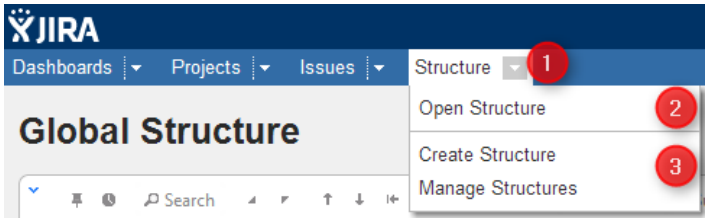
Structure Popularity

Structure **popularity** is the number of users who have marked this structure as favorite. Manage Structure (see page 190) page has **Popular** tab, which shows the most popular structures.

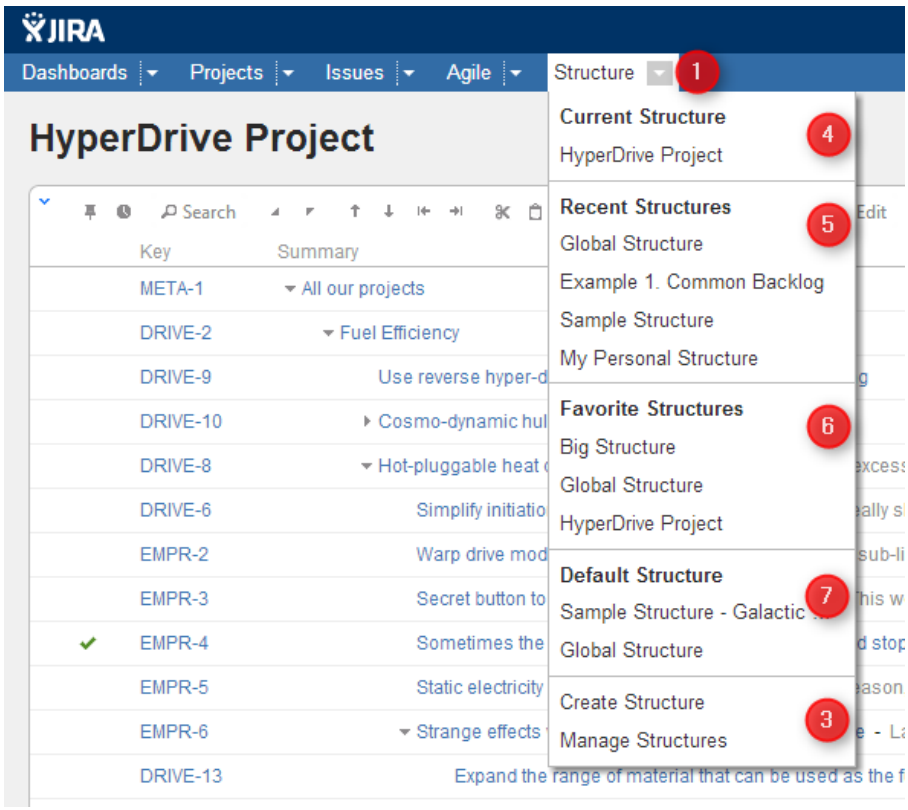
1.2 Structure Menu

Once you install Structure you will see a new item added to the top-level navigation bar.

If you don't have any structures in your JIRA yet, the menu will allow you to create a new structure or see the Getting Started page, which will help you understand the main Structure functionality:



If you already have several structures, or have access to more than one structure, the menu will look like this:



The menu has several sections:

1. **Current Structure.** Shows the last viewed structure. Click the structure name to open Structure Board with that structure.
2. **Recent Structures.** Shows structures that you've visited recently, or those which have been recently updated. Click a structure to open it.
3. **Favorite Structures.** Lists structures you have marked as your favorite. This section is absent if you don't have any favorite structures
4. **Default Structure.** Shows the system-wide Default Structure (see page 12). Also, if another default structure is defined (see page 311) for the current project, it is also shown in this section.

1.3 JIRA Pages with Structure

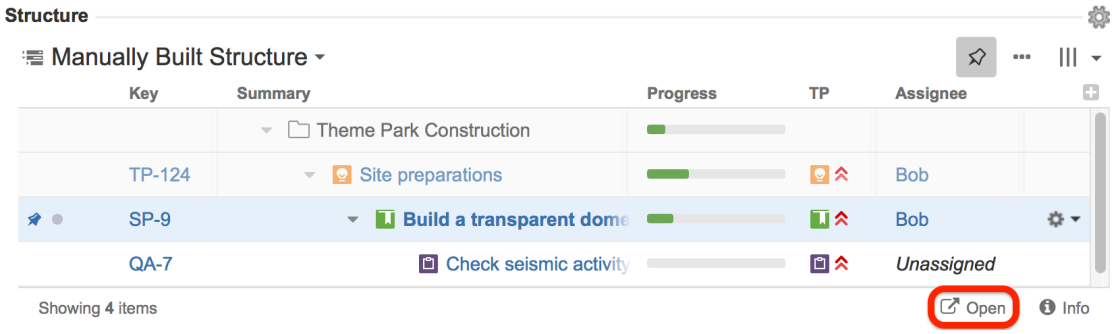
The structures you work with are displayed through the **structure widget**, which you can see in several places in JIRA:

- On a dedicated Structure Board
- On the issue page
- On the project, component and version pages
- On a dashboard

The widget displays a scrollable grid with the hierarchical list of issues and other items and lets you work with the structure itself and the items displayed there.

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:

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 (see page 32)

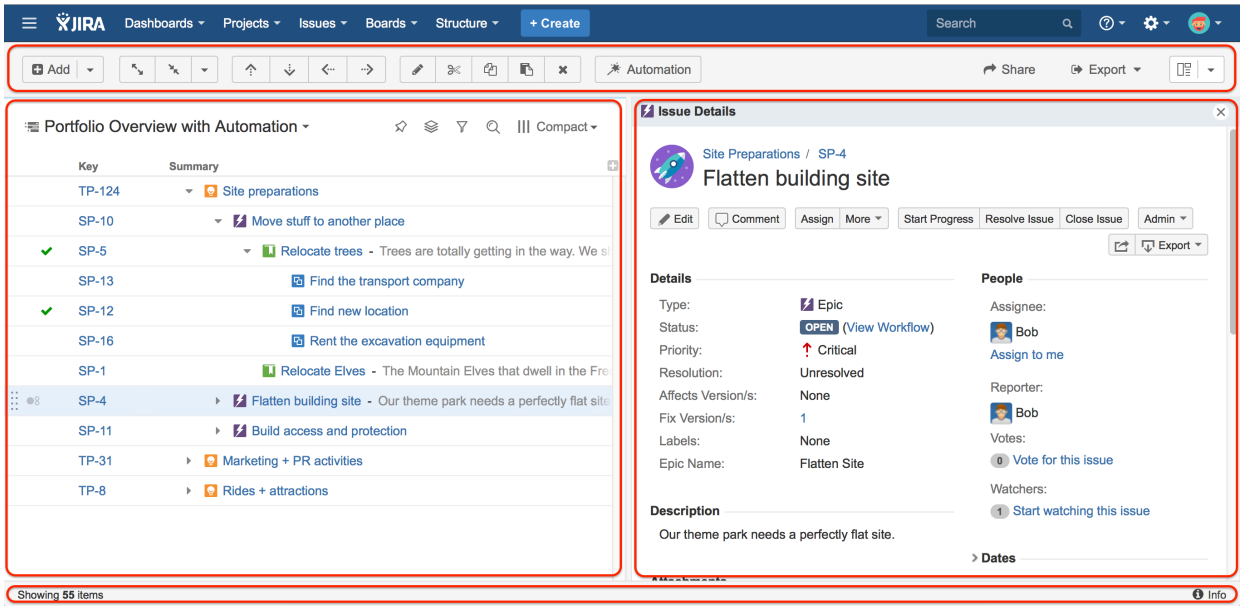
1.3.1 Structure Board

Structure Board is a full-screen view which gives you access to all the features available in Structure.

The main elements are:

- **Structure Toolbar** at the top, which gives you access to the main functions for manipulating structures
- **Working Panels**, which allow you to display structure widgets (see page 33), search results, clipboard contents and issue details

- **Status Bar** at the bottom, which shows the number of items currently displayed, links for the **Undo** operations and notifications



To open the Structure Board, click **Structure** in the top navigation menu in JIRA and select the specific structure you want to see.

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

You can make the structure board your JIRA Home page (see page 17).

If you cannot see the structure you need in the top menu, there are several options:

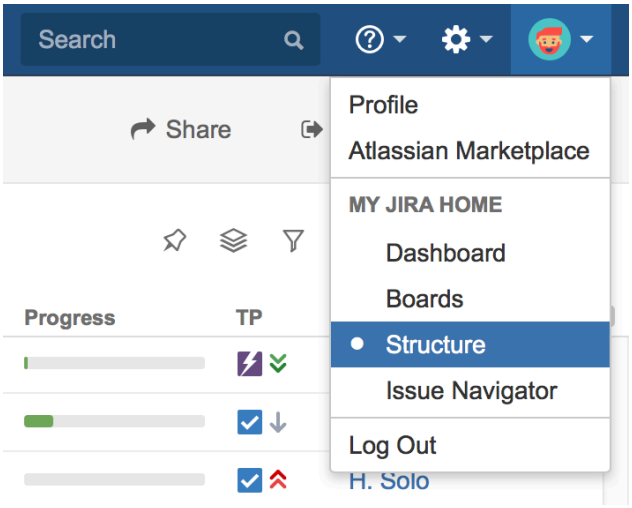
- Open any other structure, click the **Structure Name** and start typing the name of the structure you need. You'll see the results in the menu update as you type. Once you see your structure, click its name and it will be opened in the widget.
- Click **Structure** in the top navigation menu and select **Manage Structure** - this will take you to the Manage Structure page where you can browse for the structure you need, and then click the structure name to open it.
- If you know the ID of the structure you need opened, you can directly open a URL:

  `http://<i>your.jira.address</i>/secure/StructureBoard.jspa?s=<i>st`

Making Structure Board Your JIRA Home

If you want to go straight to the Structure Board (see page 15) when you log in to JIRA, you can make it your JIRA Home page. To do so:

1. Click your avatar in the top right corner of the JIRA page.
2. Select **Structure** in the **My JIRA Home** section.

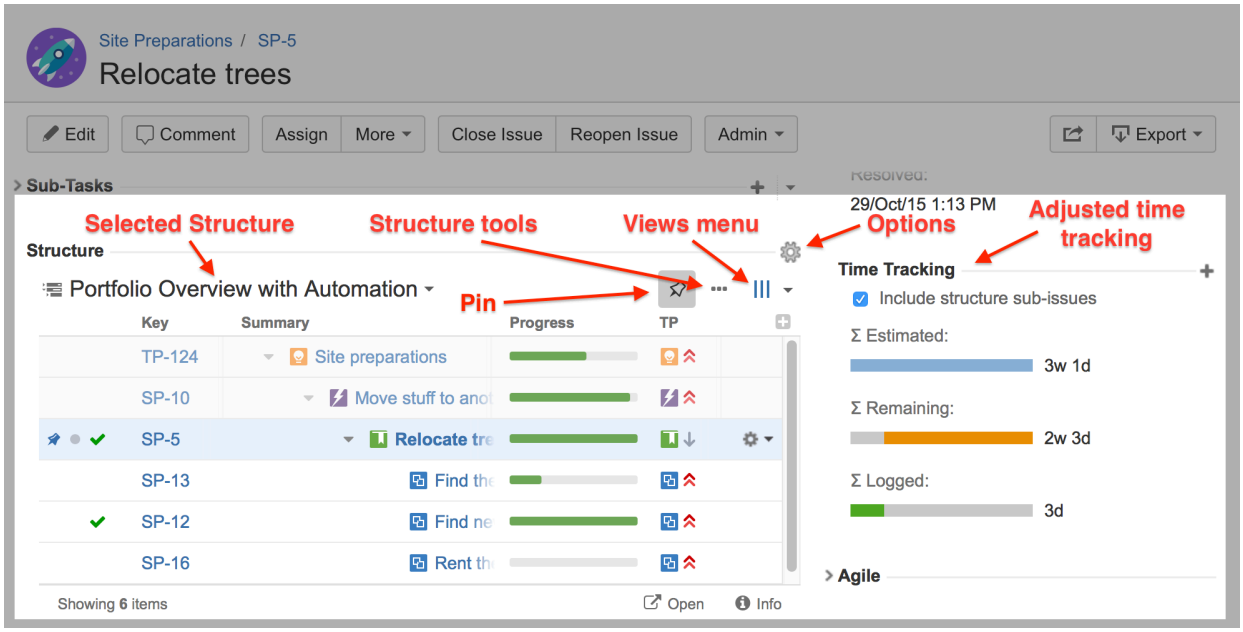


When used as a JIRA Home page, the Structure Board will show your most recently opened structure.

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.

1.3.2 Structure on the Issue Page

If an issues belongs to a project for which the Structure add-on is enabled (see page 305), the Structure widget is displayed on the issue details page. The widget is presented as a separate section, located right above the **Activity** section.



When you open an issue page with the structure section, the structure is chosen based on the Structure Options for the Issue Page (see page 21).

The issue itself is automatically located and Pinned (see page 130) in the structure. This means only the parent issues and sub-issues of the viewed issue are displayed. You can unpin the issue to see the whole hierarchy by clicking the **Pin** button on the toolbar or by using a keyboard shortcut.

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 a view (see page 40) to fit only the necessary information in a narrower space left for the Structure widget.

Structure widget can be hidden from the Issue Details page. Please refer to the Structure Administration (see page 325) article for details.

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

Collapsing/Showing 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 open does not belong to the selected structure. (This behavior can be adjusted in the Structure Options (see page 21).)

When the structure section is hidden, the issue hierarchy is not loaded from the server – it will be loaded 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.

Structure Selection

As you open the issue details for the first time, you will see one of the structures that contain this issue (this behaviour can be adjusted through the Options (see page 21)).

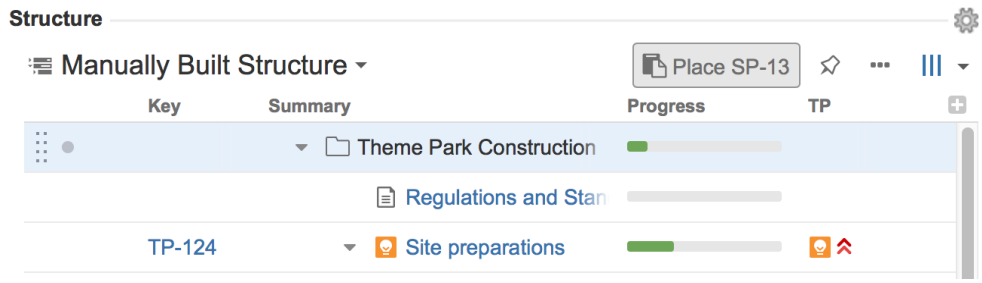
To switch to a different structure, simply click the name of the currently displayed structure and select the one you want to see. You will see the structures that contain this issue in the top section of the displayed menu.

> The structures, where this issue is added by some Generator will not be shown in the list of structures with this issue, as this would significantly affect the performance.

As you switch to another structure, this new structure is memorised and shown next time you open an issue.

Adding Issue to a Structure

If the issue you are viewing does not belong to the structure currently selected, you can add it to this structure. To do so unpin the issue by clicking the **Pin** button. Then in the structure that you'll see select an issue under which you want to add the issue you are viewing and click the **Place** button.



Now you can click the **Pin** button again to see only your issue and its parents and children.

If the issue is already in the structure, you can add another instance of it to the structure using the same approach. Unpin the issue, select a location for this new instance of the issue and click the **Place** button.

Structure Tools

Next to the **Pin** button you can see the **...** button, which allows you to access some of the basic structure functions such as Add New issue, Expand/Collapse, Edit, Copy, Cut, Paste and Remove. Using it you can work with the structure in a similar way you work with it on the Structure Board.

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 (see page 42) and select another view for the displayed structure.

The asterisk is shown next to the view name if it has been locally adjusted (see page 46).

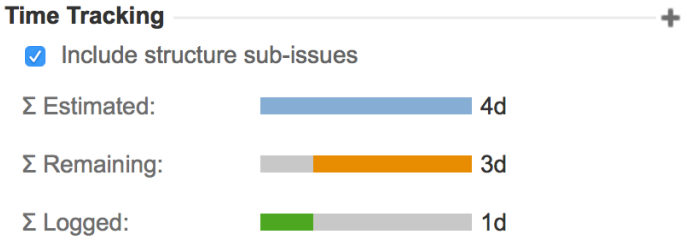
Click Options icon to open Structure Options for the Issue Page (see page 21).

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 by clearing the **Include structure sub-issues** check box. The standard JIRA time tracking will be shown (without Structure plugin). The browser will remember your preference and will show you original Time Tracking panel when you open other issues, until you select the **Include structure sub-issues** check box 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.

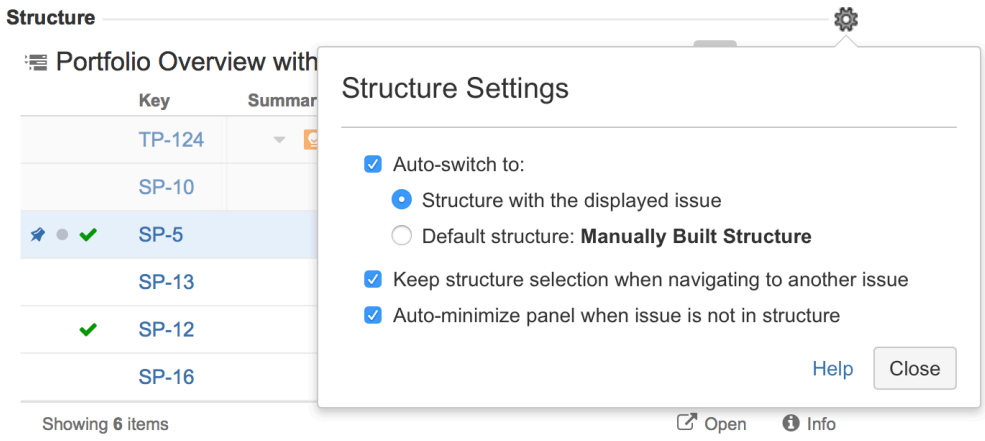


Activity tab

As you work with structures all changes are added to the JIRA Activity Stream. So in the **Activity** tab of the issue page you can see all such changes that affect the current issue. This may be useful 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 (see page 249) for more information.

Structure Options for the Issue Page

There are a few options that let you adjust 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 needs 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 opened last 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 (see page 12) 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 unintentional change of the viewed structure when you go through the structure's issues.

⚠ **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 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 (see page 310) page.

1.3.3 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.

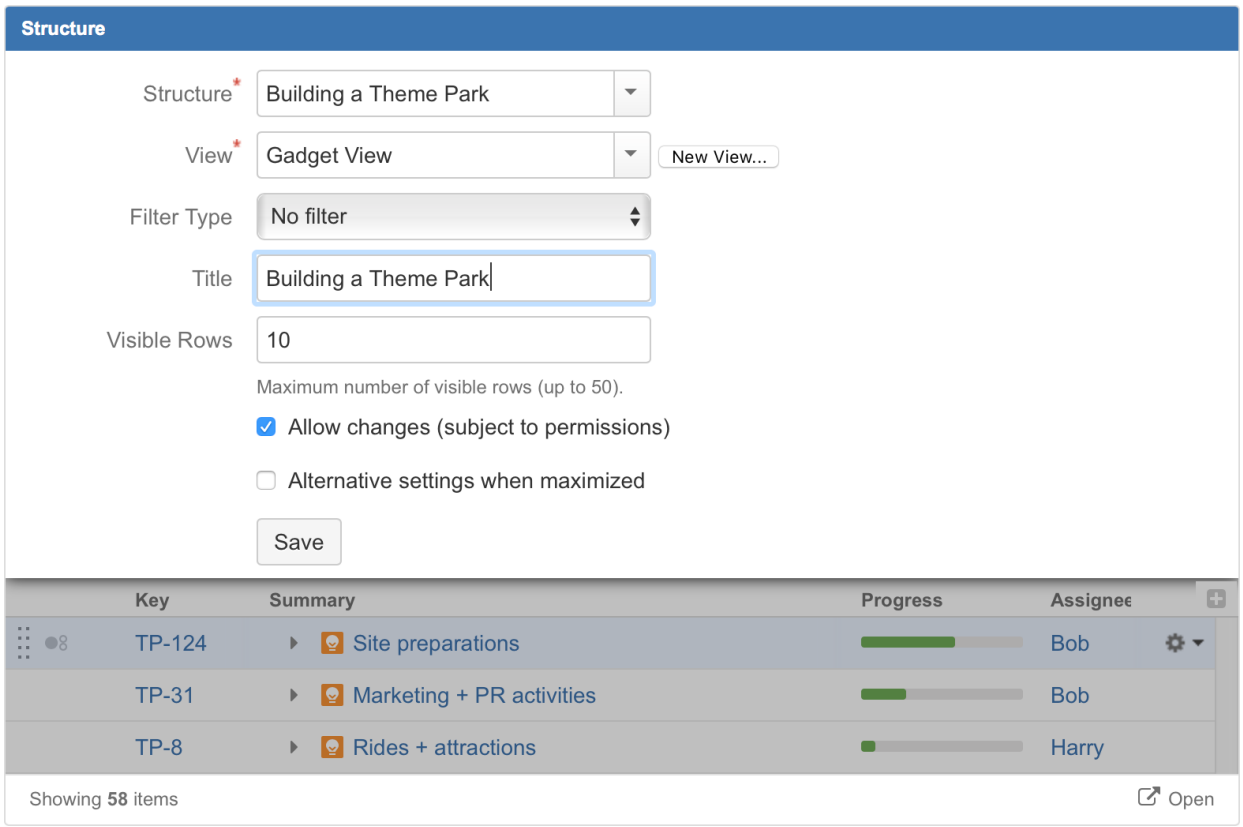
Adding Structure Gadget to Dashboard

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 (see page 40) determines which columns gadget displays. (You will be able to adjust the view later.)
3. Optionally, configure a **Filter**. The displayed structure will be filtered in the same way as in the Structure Board – see Filtering (see page 129). You can choose between a simple text filter, JQL, S-JQL, or a saved JQL filter – see Simple, JQL, and S-JQL Search (see page 127).
4. Optionally, define the **Title** for this gadget. By default it is the name of the selected structure.
5. 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.
6. 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.
7. Optionally, decide if you'd like the gadget to have different **View** and **Visible Rows** settings when maximized. Select the **Alternative settings when maximized** checkbox to configure these parameters for maximized gadget.
8. Click **Save**.

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

It may be useful to have different **View** and **Visible Rows** settings when gadget is maximized. In this case you can use the wide screen of the maximized gadget's window optimally and see more information for the same structure. Select **Alternative settings when maximized** to configure these parameters.

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 (see page 213) 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 (see page 43) 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 (see page 43) 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 (see page 213) – 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 (see page 213) – 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 (see page 43) 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 (see page 46) 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 (see page 163) 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

In the current version the Structure Gadget in Confluence is supported, but some issue may occur. This will be fixed in the future versions.

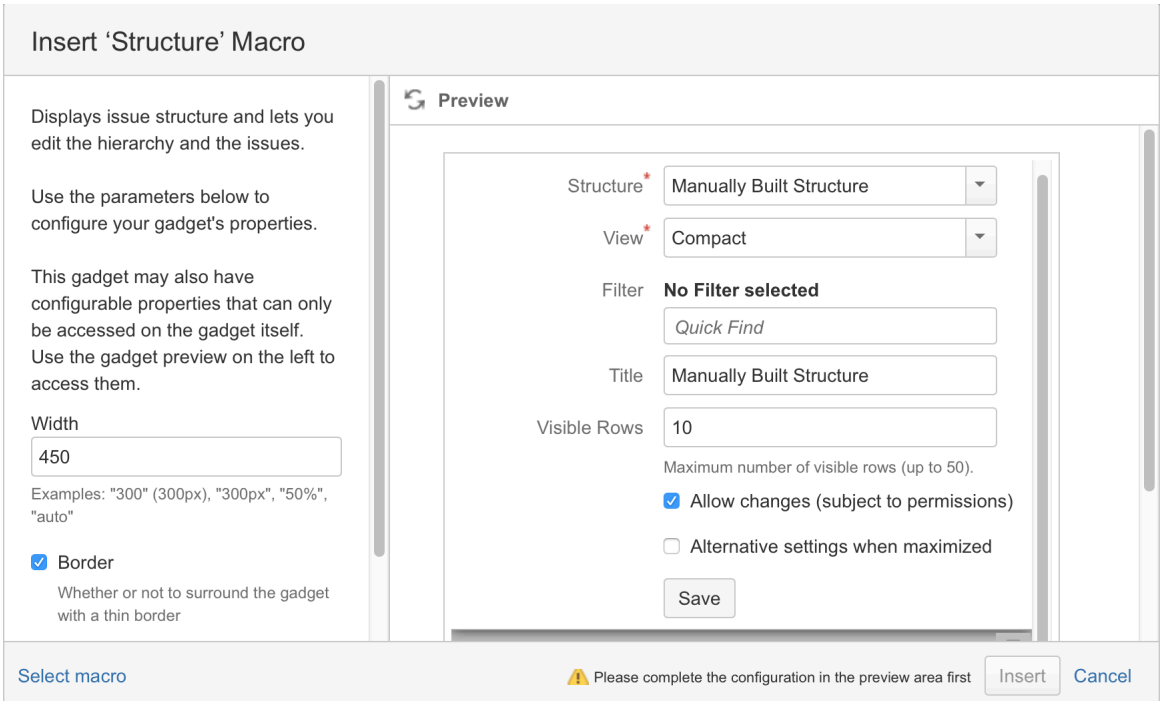
You can embed Structure Gadget (see page 22) in a Confluence page and view or edit structure in Confluence.

Before you can use Structure Gadget on a Confluence page, your Confluence administrator must add Structure Gadget to Confluence Configuration (see page 28). 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 later. For now, please use Printable Page (see page 171) to print a structure separately.

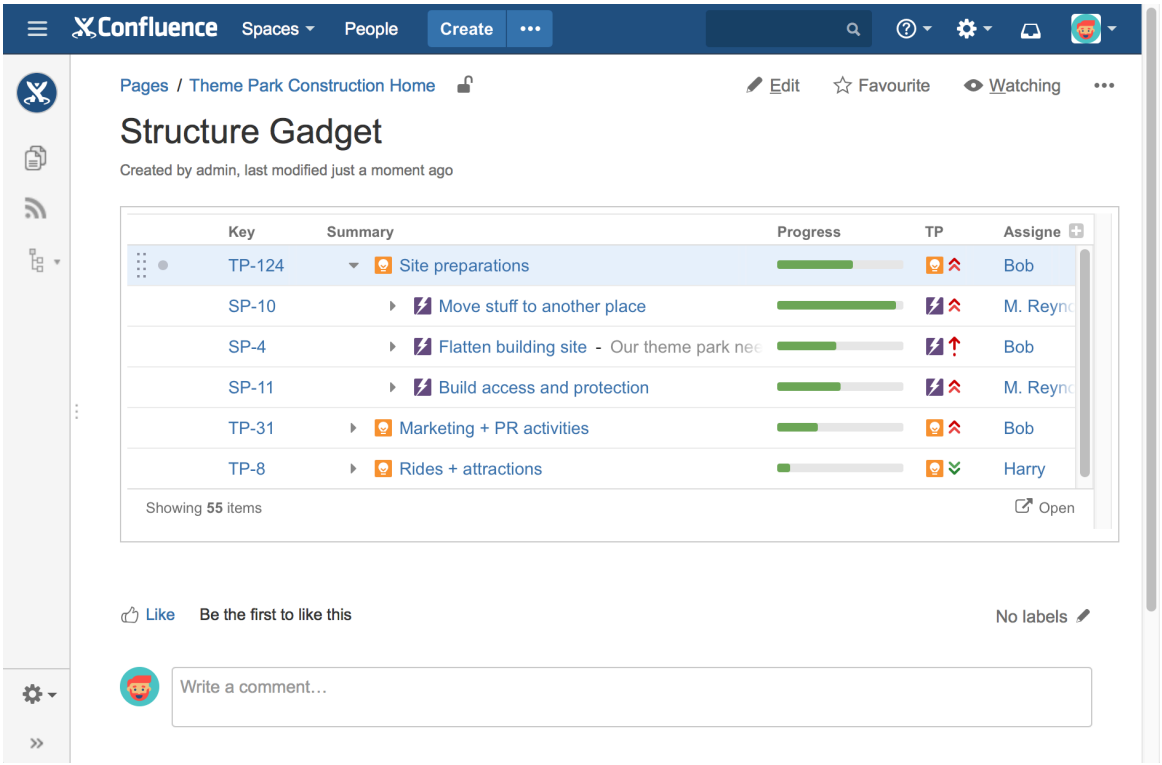
How to Add Structure Gadget

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 (see page 22) - 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.

If you have problems with Structure gadget in Confluence, check the browser's console. If you see errors saying that loading some of the resources is denied, then you hit a CORS problem in JIRA. To work around that problem, see Setting Up CORS Filter in JIRA (see page 29).

Main article: Adding JIRA Gadgets to a Confluence Page

Setting Up CORS Filter in JIRA

Sometimes Structure Gadget fails to load correctly in Confluence. You might see missing icons or the application can fail to work.

This may happen because of a known JIRA issue that prevents Structure gadget from loading resources from JIRA when it's being served in Confluence on another web domain.

To work around that problem, you can set up CORS filter in the Tomcat server that runs JIRA:

1. Copy cors-filter-2.4.jar, java-property-utils-1.9.1.jar from CORS docs to the **/lib** directory under JIRA's installation folder.
2. Edit file **JIRA_INSTALL_DIR/atlassian-jira/WEB-INF/web.xml** and add the following:

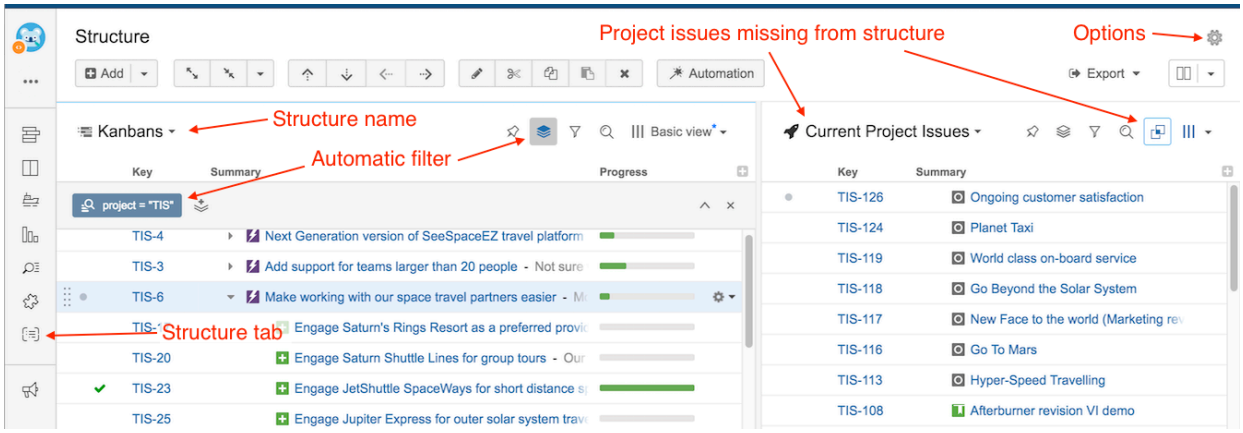
```
 <!-- ==================== CORS configuration
====================== -->
<filter>
    <filter-name>CORS</filter-name>
    <filter-class>com.thetransactioncompany.cors.CORSFilter<
/filter-class>
    <init-param>
        <param-name>cors.allowOrigin</param-name>
        <param-value>http://YOUR-CONFLUENCE-DOMAIN.com</param-
value>  <!-- use http: or https: depending on your
configuration -->
    </init-param>
```

```
</filter>
<filter-mapping>
    <filter-name>CORS</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Restart JIRA

1.3.4 Structure on the Project Page

Structure widget is displayed in a separate tab on the project page if the project is enabled for Structure (see page 305).



The widget on the project page is the standard, fully-functional Structure widget and has the same functionality you can find on the Structure Board (see page 15) (except for Perspectives (see page 217) and Issue Details panel), but it has several specific features not found on the Structure Board.

The current project defines the scope of the displayed data – it's used to filter the structure and show project issues in the second panel.

Layout

When you open Structure tab, Double Grid layout is selected automatically.

The primary (left) panel displays the most recently viewed structure or the default structure for the current project (as defined in the Options dialog). You can quickly switch to another structure by clicking the structure name and selecting the desired structure.

The secondary (right) panel shows issues from the current project that are not part of the selected structure. This allows to quickly place other issues in the project into the structure.

Automatic Filter

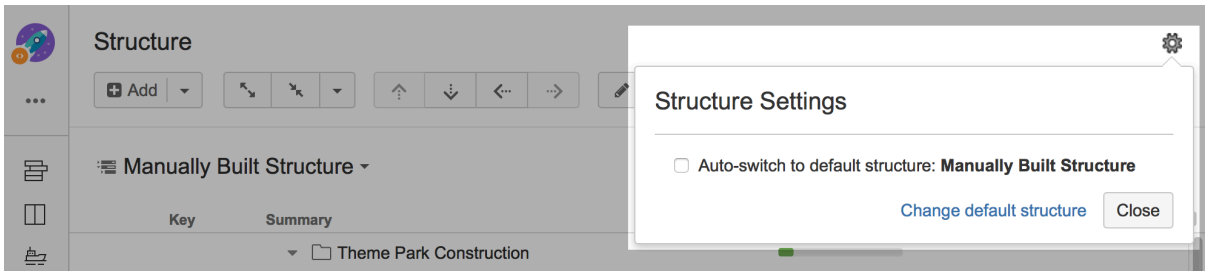
An automatic project filter is added to the primary panel. This is a non-removable transformation that hides all issues from other projects.

If you'd like to see the full structure without this filter, click *Open* link in the Structure widget's footer.

Project Page Options

You can make the widget open with the structure that is defined as a default structure for this specific project.

To do that, click the options gear button in the top right corner and select the **Auto-switch** option. The changes are saved to the server and applied immediately.

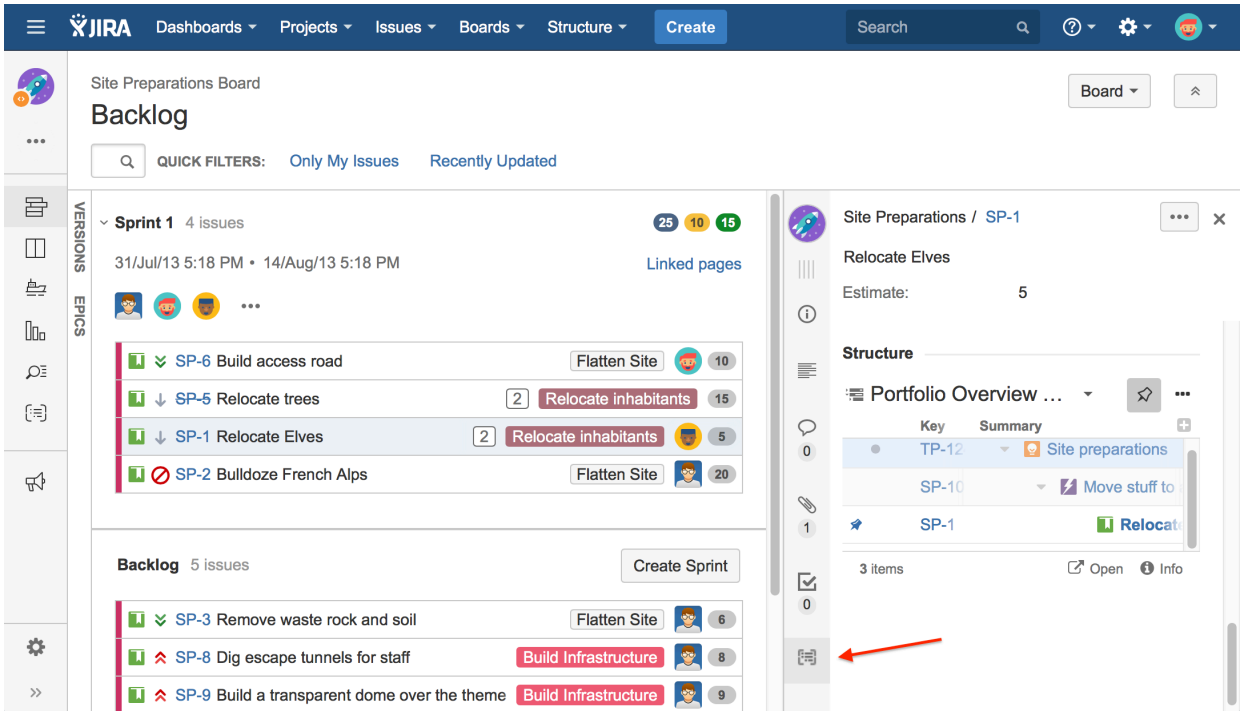


If you are the Project Administrator, the options dialog will also show the link to a page where you can change the default structure for your project.

The default value for this option can be configured by JIRA Administrator on the Structure Defaults (see page 310) page.

1.3.5 Structure on Agile Boards

If you are using JIRA Software (formerly JIRA Agile or GreenHopper), 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 Item Mode (see page 130) , 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 (see page 43) if needed. (And get a larger display!)

When you click 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 a different structure clicking the structure name.

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.

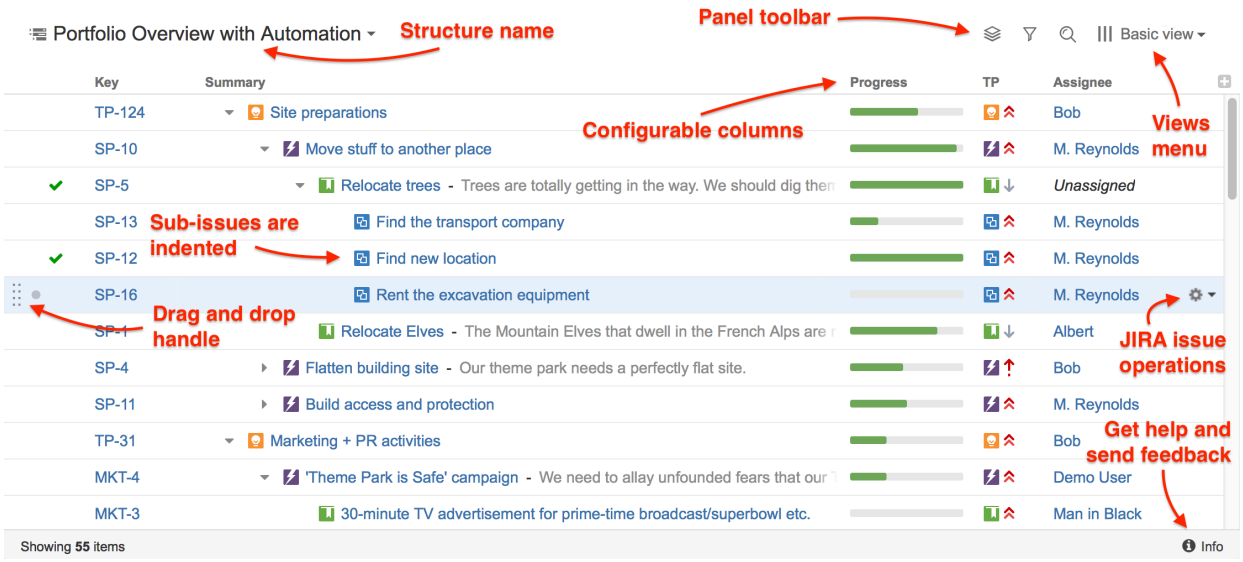
Only the users who have access to Structure (see page 306) will see the Structure tab on Agile boards.

1.4 Working with the Structure Widget

Structure widget is the main tool for working with a structure and its items. It's a component of the Structure plugin that is used on the Structure Board, on the Issue Page and in other places (see page 15) where structure is displayed.

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

1.4.1 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 (see page 15), Issue Page (see page 17) and in other places in JIRA (see page 15).

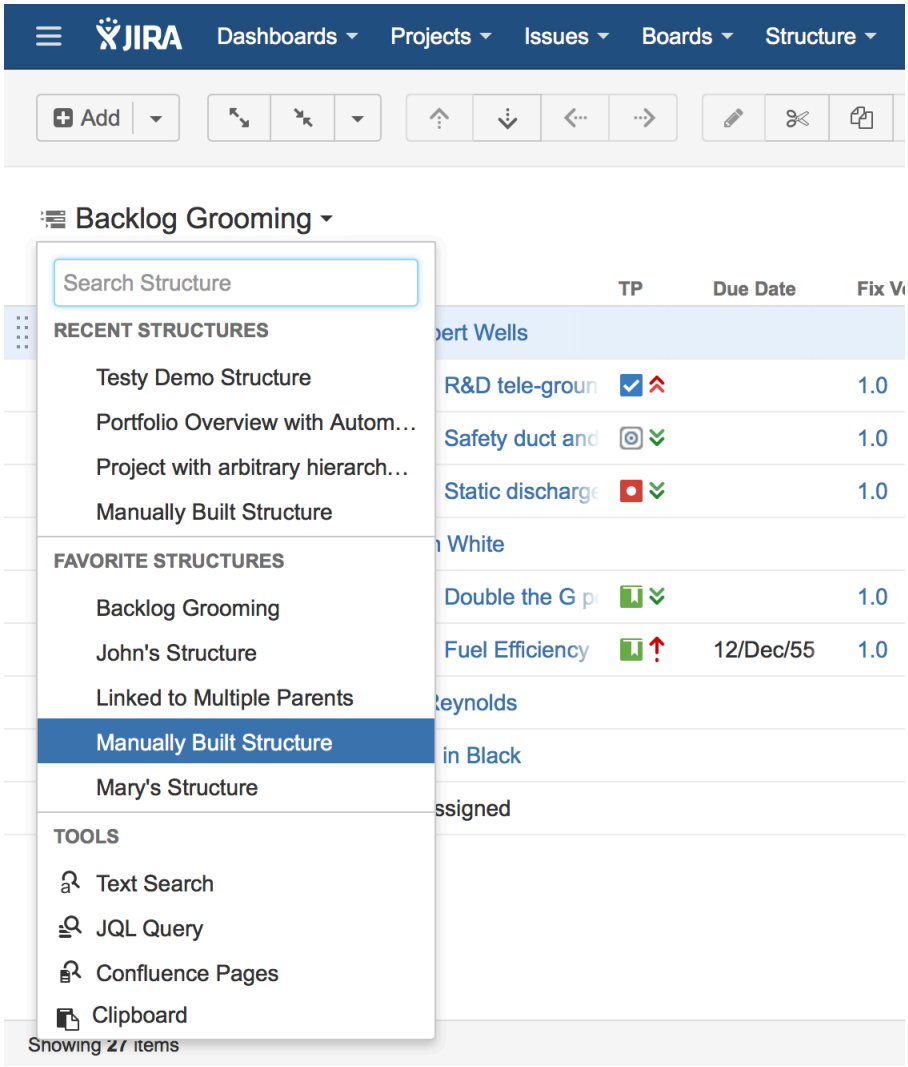
Structure lets you navigate the hierarchy and search for specific issues.

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 (see page 35)

Switching Between Structures

You can quickly switch between structures right in the structure widget on any of the pages with structure. Click the structure name to open the menu:



Find the structure you need in the list of recent or favorite structures and click it to open.

If your structure is not there, start typing the name of the structure you need and you will see the list of matching structures.

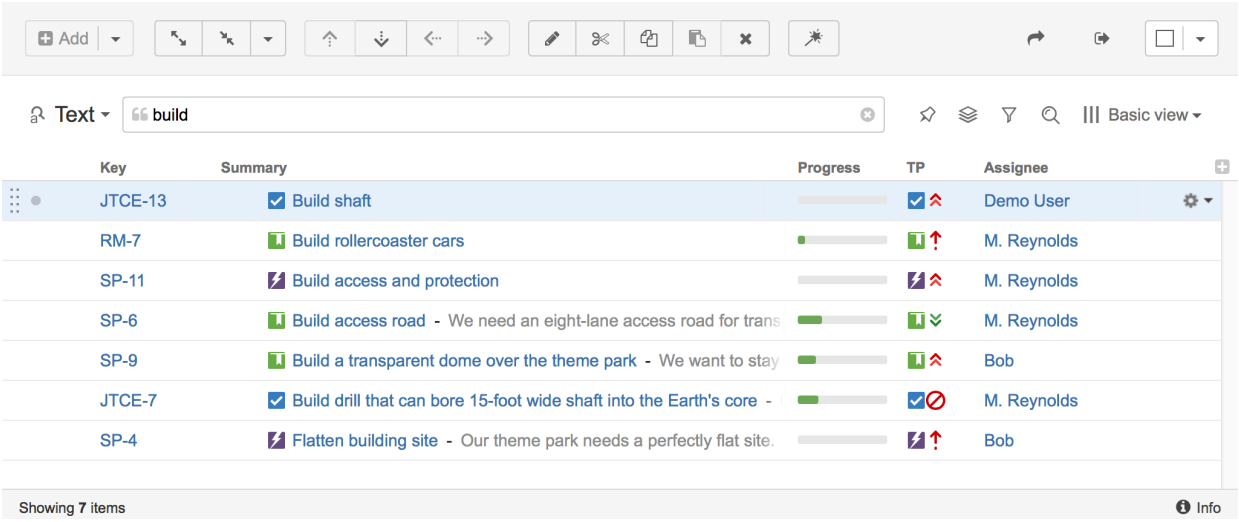
> On the Structure Board apart from the structures, you can also use the widget to run text and JQL searches, see the clipboard contents and, if you have Structure.Pages installed - search for Confluence pages. You can find all these options in the Tools section of the menu.

Using Structure Widget for Searching

On the Structure Board you can use the structure widget not only for showing structures, but also for finding existing issues using JQL or text search and displaying the clipboard contents.

If you have a structure open, to start searching click the structure name and select JQL or Text search.

Once the search is open, as you start typing, the results will be updated.



Just like with structures, you can select a specific **view** for your search results and then add and arrange columns as necessary.

The **structure panel toolbar** also works for search results the same way it works for structures. You can apply sorting, additional filtering and more complex transformations.

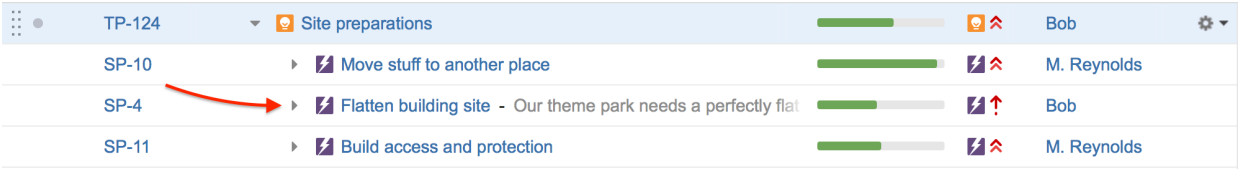
> ⚠ Search only looks for issues from structure-enabled projects (see page 305).

1.4.2 Navigating Structure

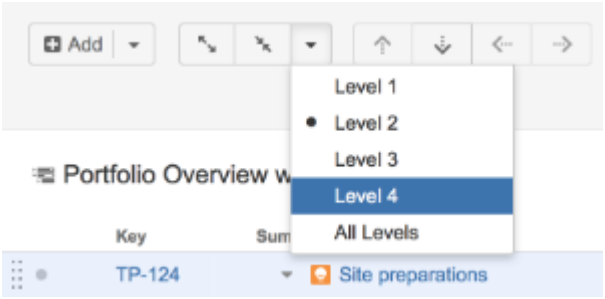
Navigating with Mouse

You can select items, scroll up and down, as you would do with any table. Clicking a link of an item will open the panel with item details or take you to the item page depending on the settings. So if you'd like to just select an item, click anywhere in the row except the underlined links. The row of an issue is also selected when you click the JIRA actions icon at the end of the row.

To show or hide sub-items of a parent item, click the **Expander** button near the item summary.



To expand or collapse the whole hierarchy, use **Expand All** or **Collapse All** buttons in the toolbar. You can also expand the structure to a certain level by clicking the drop-down menu next to these buttons and selecting the desired level of depth.



If there are many items in the structure, not everything is loaded from the server. As you scroll down or expand sub-items 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 items.

Navigating with Keyboard

You can use **arrow keys** to focus on the next or previous item in the list. Left and right arrows expand and collapse sub-items list.

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

Using **Ctrl+Arrow Key** moves the selected item 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 (see page 285) 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 Items

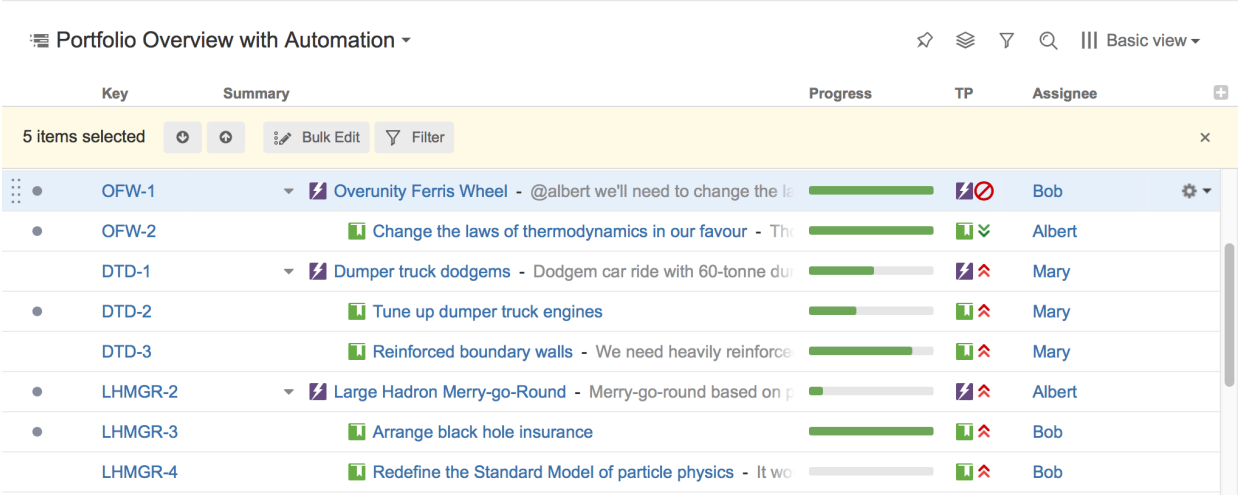
Structure allows you to select multiple items and do most of the operations with selected issues at once.Usually, you navigate structures and focus on a single item for further actions with keyboard arrows or mouse. The focused item is highlighted with the blue background, and the actions (like moving) apply to the highlighted item only.

Entering Multi-Select Mode

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

- Press *Space* to add currently focused item and move to the next issue.
- Click grey dot in the beginning of an item 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 items are marked with a filled circle and the additional panel appears at the top of the grid showing the number of selected items and several action buttons.



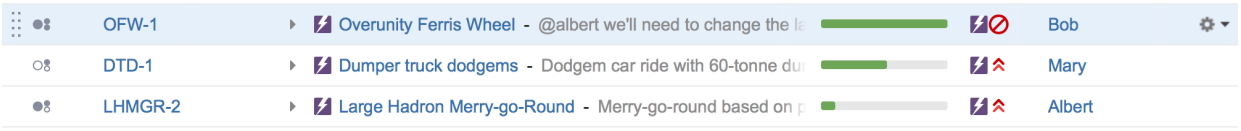
The selection panel offers the following features:

- Move focus from one selected item to another by clicking the up and down arrows.
- Bulk Edit the selected issues using the Bulk Edit button.
- Show only selected items and their parents by clicking the Filter button.
- Remove selection by clicking the close button in the right corner of the panel.

Special Selection Markers

If you collapse a list of sub-items, the selection marker of the parent item will show if it contains any selected sub-issues.

For example, if you collapse sub-issues of *OFW-1*, *DTD-1*, and *LHMGR-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:

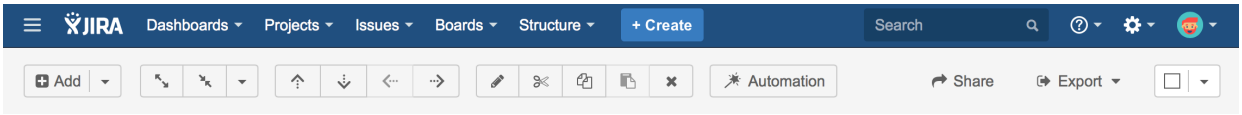
	DTD-1	DTD-1 itself **is not** selected, but **some** of its sub-items are selected
	not on the example	the item itself **is not** selected, but **all** of its sub-items are selected
	LHMGR-2	LHMGR-2 **is** selected, and **some** of its sub-items are selected
	OFW-1	OFW-1 **is** selected, and **all** its sub-items are also selected

Exiting Multi-Select Mode

Hit *Escape* key to clear multiple selection and exit multi-select mode (or press the close button in the selection panel). You can also hit Ctrl+A (Command+A) twice – first key stroke will select all items, second one will de-select all items.

1.4.3 Main Structure Toolbar

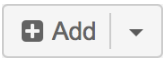
The structure toolbar provides access to the main functions of the structure widget.



As you move mouse pointer over the buttons in the toolbar, the active buttons are highlighted. In some situations some buttons may be disabled and they will remain light grey. For example, Paste action is not possible unless you have some issues in your clipboard, so the button will remain light grey and not clickable in that case.

Hold the mouse pointer over the toolbar button for a few moments and 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
	Create a new item and add it under the item currently selected in the structure. By default, you can add either new issues or new folders (Confluence pages are available if you have Structure.Pages installed). You can also click the drop-down menu next to the button itself to open the secondary panel, where you can search for existing issues.	Creating new items	**Enter**
	Expand/collapse the whole hierarchy. Expand to a certain level using the drop-down menu.	Navigating Structure (see page 35)	**++** / **--**
	Without changing the items's parent, move the item up/down and place it before/after the previous child - if possible.	Moving Issues Within Structure (see page 143)	**Ctrl+Up** / **Ctrl+Down**
	Unindent / Indent the item one level, if possible.	Moving Issues Within Structure (see page 143)	**Ctrl+Left** / **Ctrl+Right**
	Edit the currently selected issue / stop editing and save changes.	Editing Issues (see page 158)	**Tab**
	Cut the selected items to the clipboard (see page 141).	Issue Clipboard (see page 141)	**Ctrl+x** or **Command+x**

Button	Action	More Information	Keyboard Shortcut
	Save the selected items to the clipboard (see page 141) to copy them later to another place in the structure.	Issue Clipboard (see page 141)	**Ctrl+c** or **Command+c**
	Paste the items from the clipboard (see page 141) into the structure.	Issue Clipboard (see page 141)	**Ctrl+v** or **Command+v**
	Remove the currently selected issues from the structure.	Removing Issues (see page 144)	**Delete**
	Switch on/off Automation editing mode.	Automation	~
	Create a perspective (see page 217) link to share the current view.	Perspective (see page 217)	
	Open a printable page (see page 171) with the structure or export structure to Excel (see page 172).	Printing (see page 171) Exporting to Excel (see page 172)	
	Open the secondary panel (see page 139) and it's options.	Secondary Panel (see page 139)	

1.4.4 Configuring View

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

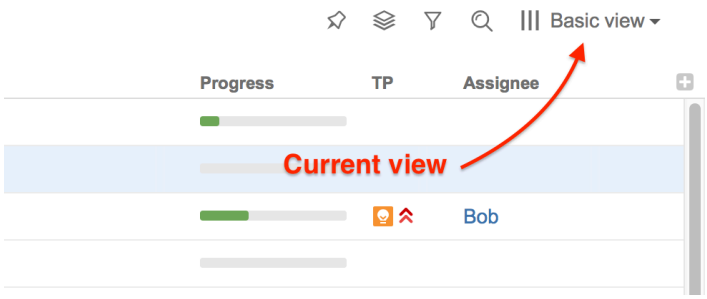
- You can configure how each item is represented by customizing columns (see page 43) or selecting a pre-defined View (see page 42).
- You can display only part of the whole structure that is relevant for some item by pinning that issue (see page 130).
- You can filter (see page 129) the displayed structure using text or JQL and display only the matching items 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 (see page 209).

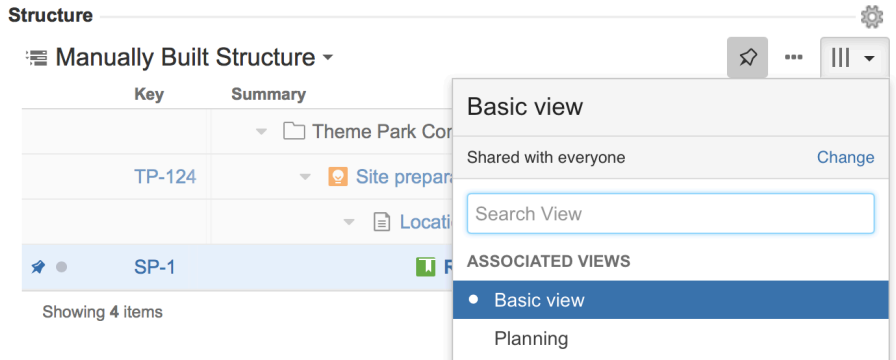
On the Structure Board, the current view is displayed in the top right corner:



On picture: Current view is called "Basic view".

If you modify the view, the small blue asterisk will be shown next to it until you either save this change or revert to the original settings: Basic view*

On other pages with structure the current view may be identified if you hover mouse over the Views icon, or click this button:



You can change which columns are displayed by switching to another view (see page 42) or by manually adding, removing or rearranging columns (see page 43).

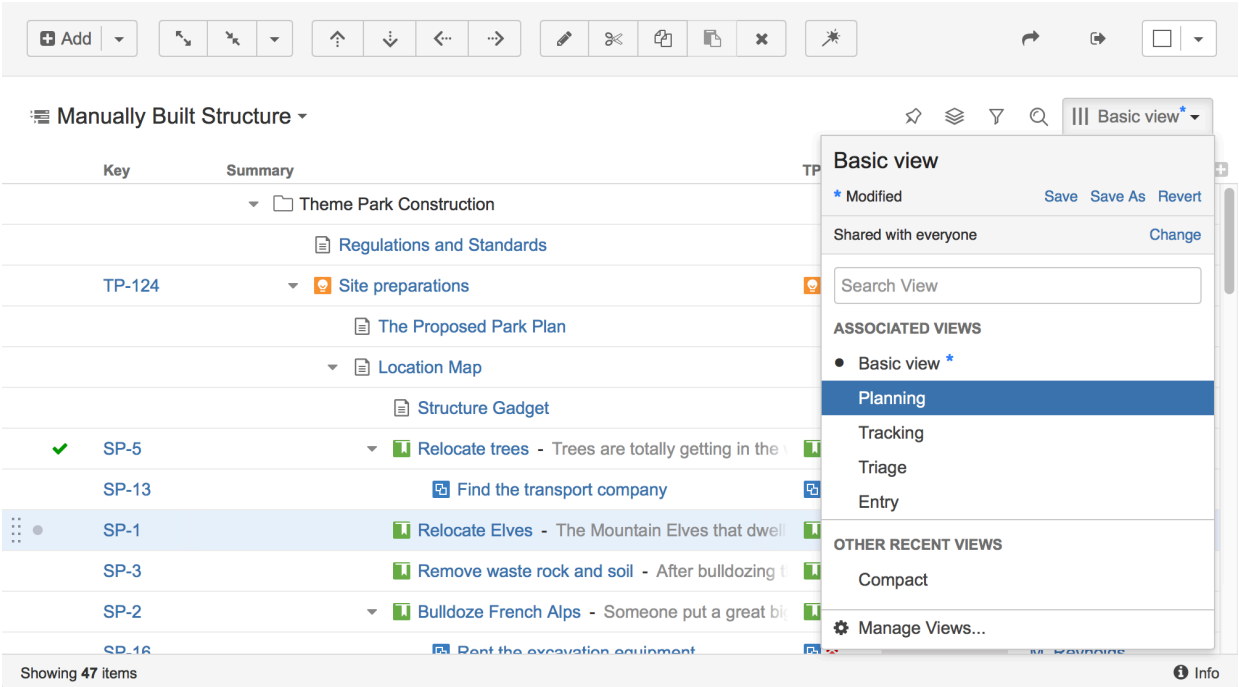
When you manually change column configuration, you create your local adjustments to the currently select view. You can then save the changes (if you have permissions to change the view) or save and share your customization as a new view – see Saving and Sharing Views (see page 46).

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 the Views icon.

Apart from the list of views the Views drop-down shows important information about the current view.



In the menu you can see the following:

1. Current view name. Hover mouse over the name to see the tooltip with the view description.
2. If the view was modified, you'll see the corresponding message and links, which allow you to save or revert the changes.
3. Permissions settings and a link to change them.
4. View search. Start typing the view name and results will be filtered as you type.

> Search looks for any views that match the entered name, not only those in this list.

5. 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 (see page 196).
6. List of views you have recently used (excluding the views shown in the section above).
7. Manage Views link opens View Management (see page 209) dialog.

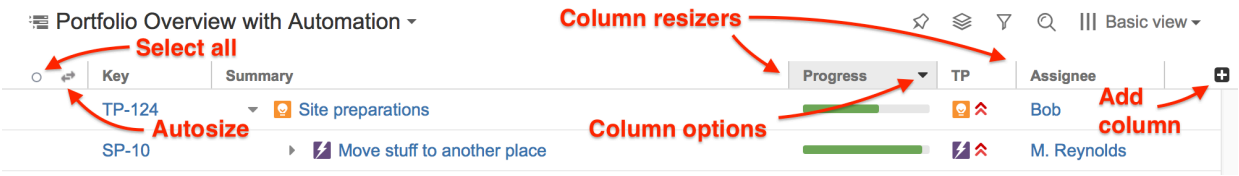
Switching View with Keyboard

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.

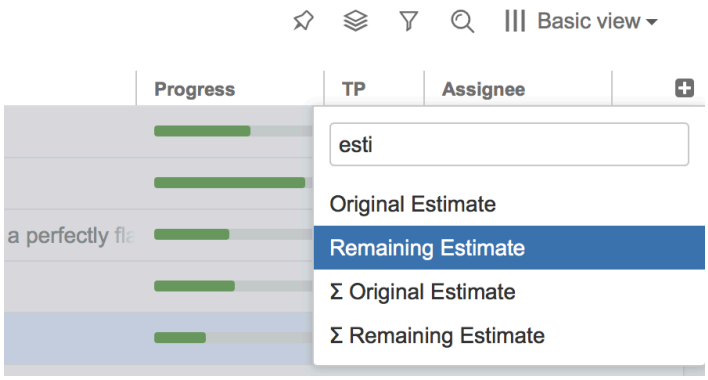


> When you add, configure, remove or rearrange columns, you make adjustments to the *View* that's being used to display the structure. Adjusted view is marked with a blue asterisk (*). The adjustments are stored in your browser and affect only yourself – to make the changes persistent and available to others, you need to save the view or create a new view (see page 46).

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.

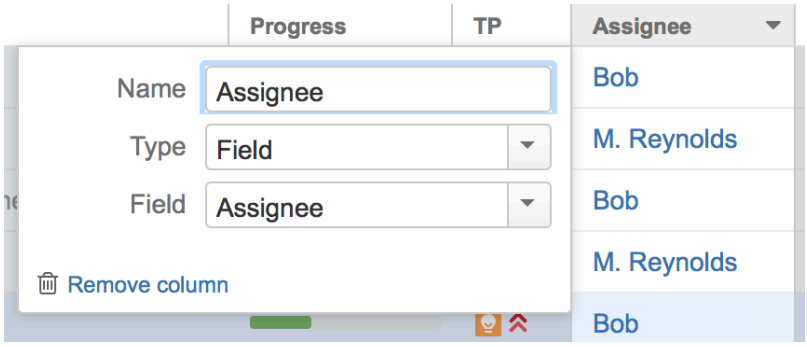


To abort adding new column, hit **Escape**.

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 (see page 47). For example, the Field (see page 48) 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 changes"** 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 (see page 48) and Special Columns (see page 65).

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 is highlighted. When column size is close to what Structure considers ideal width (based on the displayed data), the resizer "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.
- **Double-click the resizer or the column header.** The column will automatically resize the default 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 (see page 48) 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.

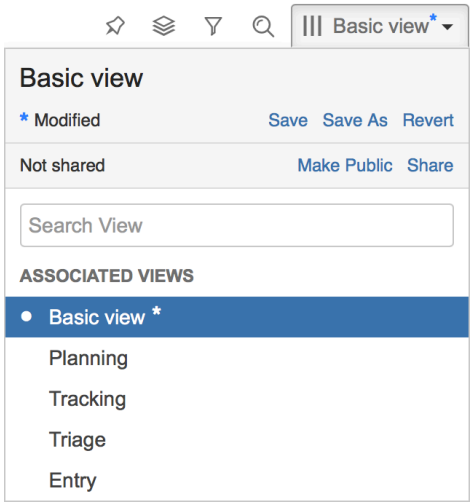
Saving and Sharing Views

When you have added, removed or rearranged columns (see page 43), you have adjusted the view that is used to display the structure. The view is marked as "adjusted" with the blue asterisk: Basic view*

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 (see page 213)). 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 (see page 213) for details.

1.4.5 Widget Columns

Structure widget provides a number of columns that display information about issues in the structure. You can customize (see page 43) the displayed columns by adding new columns, changing each column configuration, or switching to a new view (see page 42).

Out of the box, Structure provides the following columns:

Structure also contains extension API (see page 335), so the selection of available columns may be extended by a third-party plugin.

Issue Key Column

For issues the Issue Key column displays the issue key. For other types of items it remains empty.

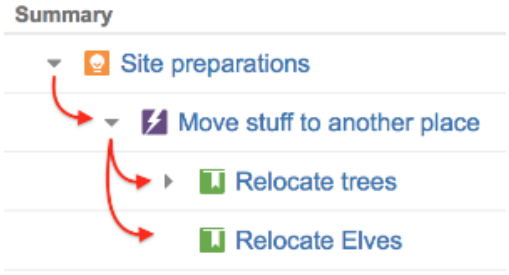
Compact View

If the project key is large, the issue key column may get too wide. You can configure Issue Key column to replace the project key with a small avatar icon of the project.

To enable compact view of the Issue Key column, open the column options and select **Compact View.**

Summary Column

For issues the Summary column displays the issue summary and, optionally, part of the issue description. For folders it shows the folder name. Sub-items have the text in the Summary column indented relative to their parent item.



Summary can be edited right in the structure widget (see page 158) and it's the only field required for creating new issues (see page 154).

To turn off descriptions in the Summary column, use the column configuration panel (see page).

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 (see page) 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.

Since each issue can be present multiple times in the structure, you can select if you want to count every instance of this issue in the totals or count it just once.

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

Editing Values

Most field columns are editable – you can edit field value (see page 158) 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 (see page) 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 (see page 43) menu. You can add any available preset configuration and then customize it using the column configuration panel (see page) (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:

Total Progress Calculation

When individual issue progress is calculated based on Status (see page 56), Percent Field (see page 59), or Resolution Only (see page 54), 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:

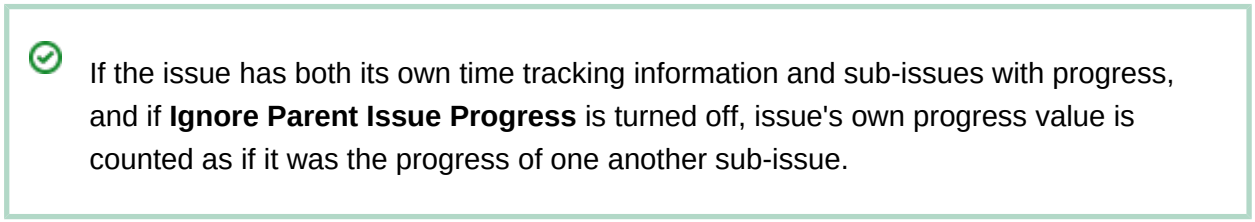
- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100%.
- Otherwise, if the issue has time tracking information, the progress is calculated proportionally to this issue completion%: `(Time Spent) / (Time Spent + Remaining Estimate)`

- Otherwise, the progress is 0%.

Calculating Progress for Issue with Sub-Issues

If the issue does have sub-issues:

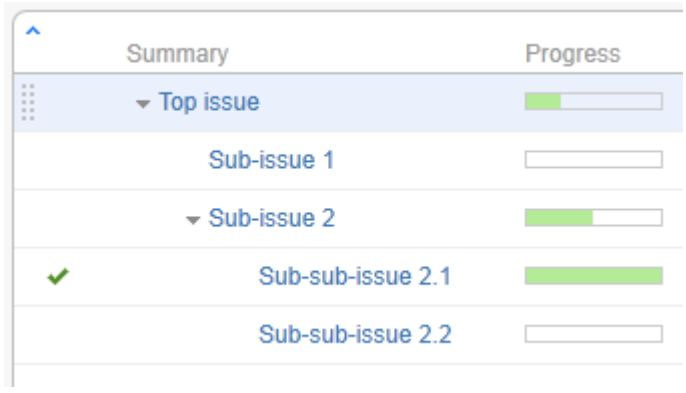
- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100% - regardless of the sub-issues progress.
- If the issue and its sub-issues do not have estimates or work logged (or if time tracking is turned off), the progress is calculated as the average from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

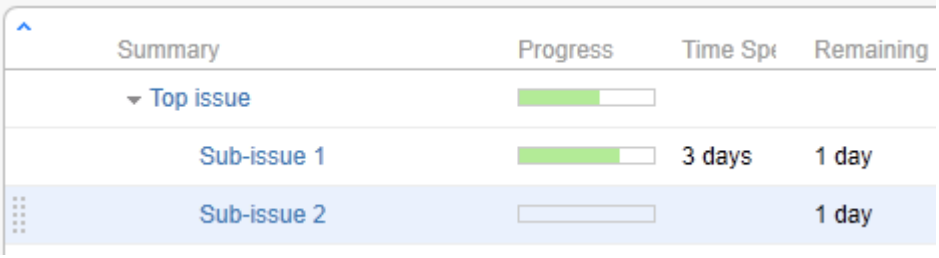
1. Example without time estimates



Issue	Explanation	Progress
Sub-sub-issue 2.1	This issue is resolved (indicated by the green mark) - so it is complete	100%

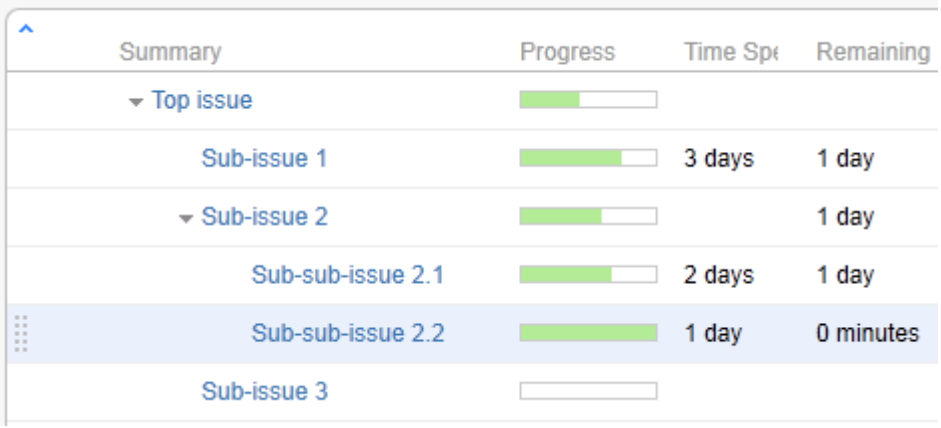
Issue	Explanation	Progress
Sub-issue 2	It has two sub-issues with 100% and 0% progress, the total progress is average value	50%
Top issue	It has two sub-issues: sub-issue 1 is 0% done and sub-issue 2 is 50% done, the mean value is 25%.	25%

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	44%

Issue	Explanation	Progress
	and 4.5 days remaining). That yields for 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%.

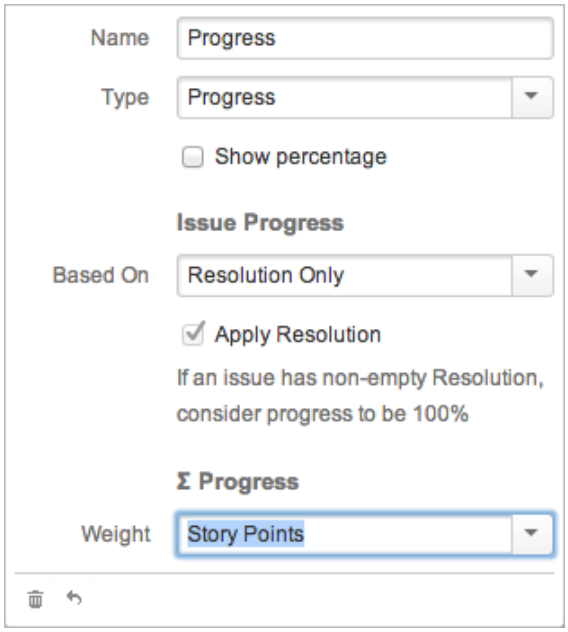
Calculating Progress for Issue with Sub-Issues

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		50%

Issue	Explanation	Progress
	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)	

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:

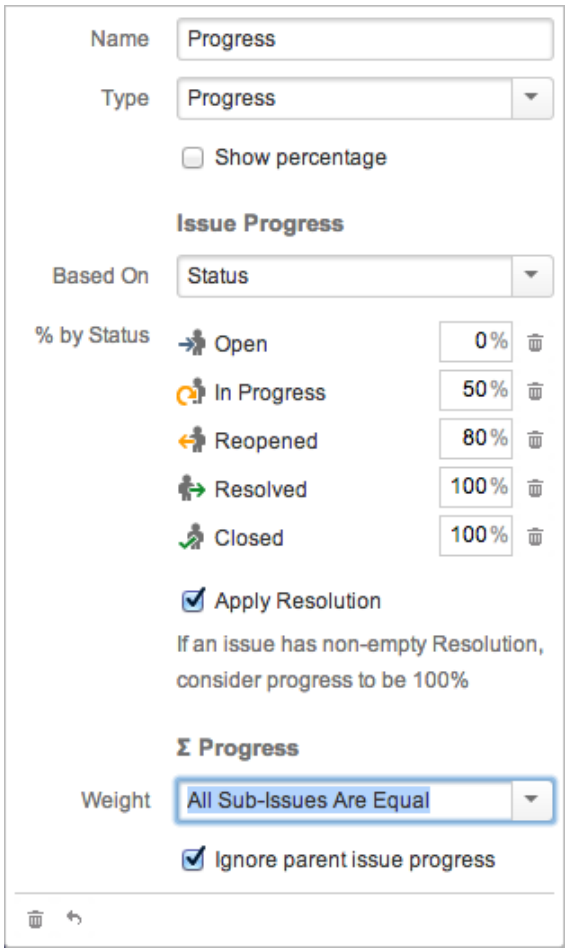
- If the issue's Resolution field is not empty, and **Apply Resolution** is turned on, the progress is 100% - regardless of the sub-issues progress.
- Otherwise, sub-issues progress is aggregated sum of progresses with chosen weights.

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.

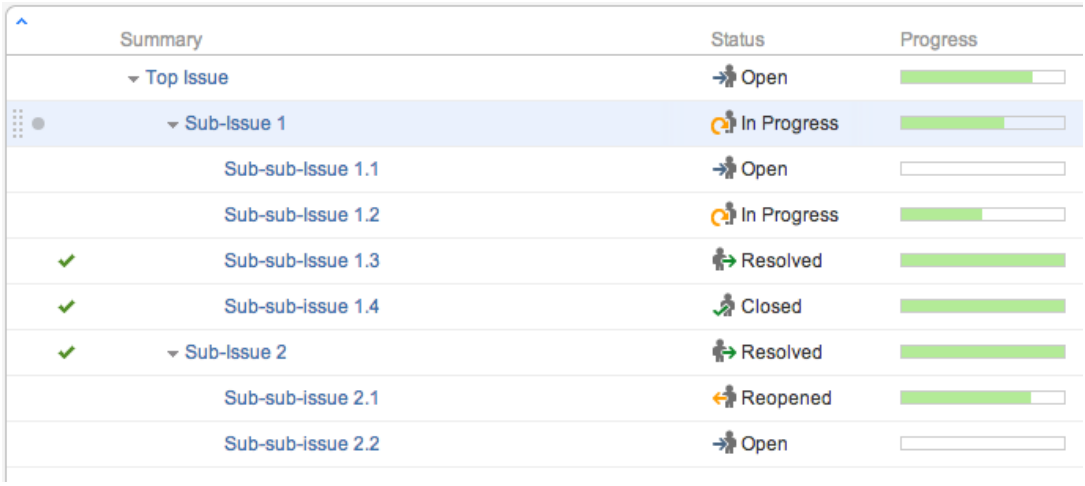
If some of statuses don't have any percentage configured, issue progress is considered undefined.

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%
		100%

Issue	Explanation	Progress
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	
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

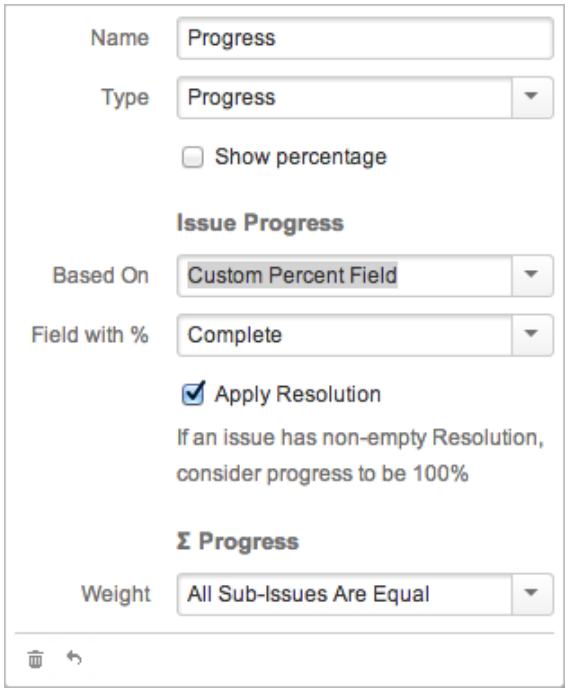
If the issue does have sub-issues:

- 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.

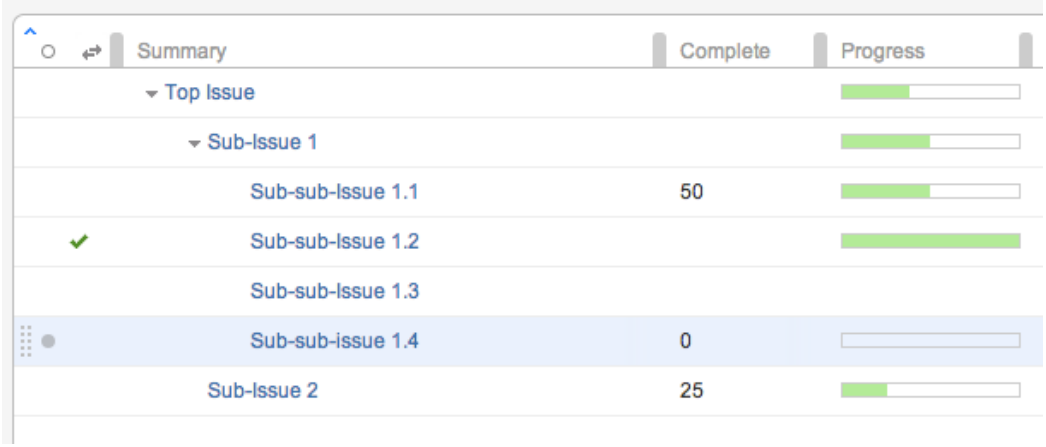
Examples

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



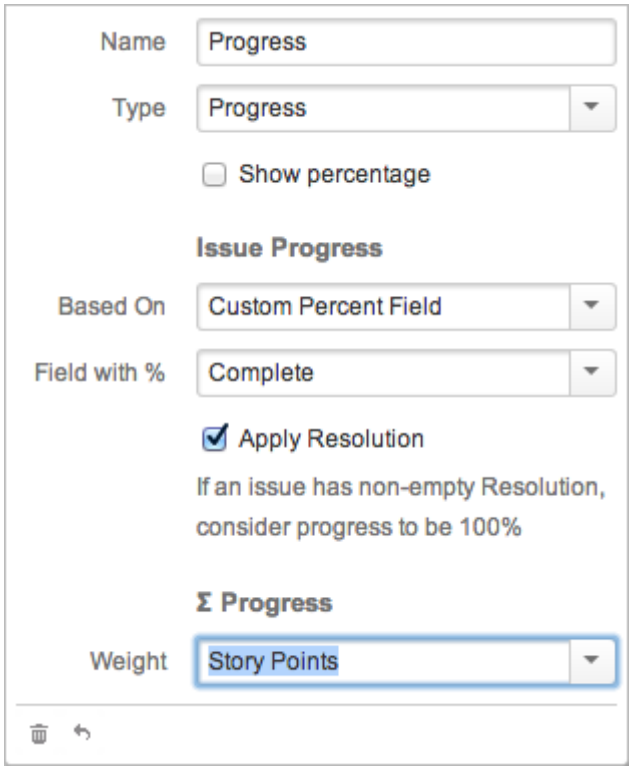
Sample Structure

Issue	Explanation	Progress
	This issue is 50% complete as specified by custom field	50%

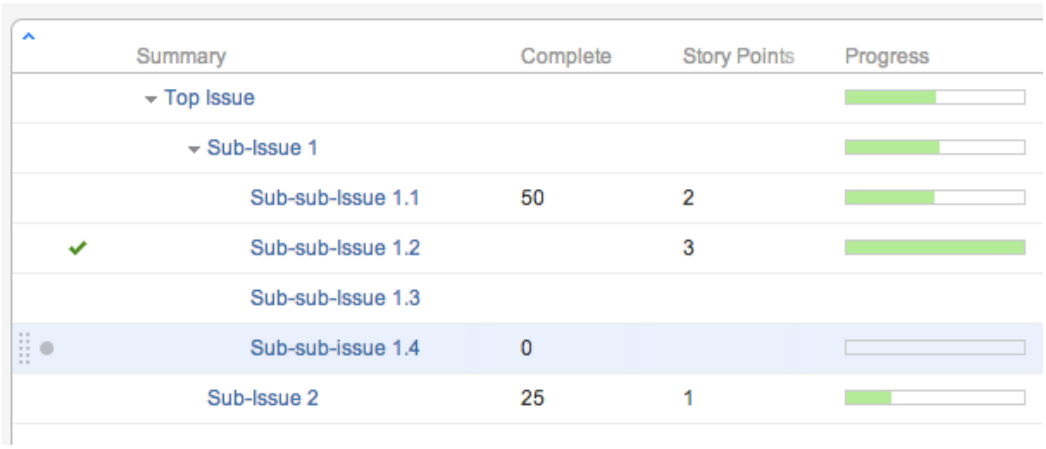
Issue	Explanation	Progress
Sub-sub-issue 1.1		
Sub-sub-issue 1.2	This issue is resolved (indicated by the green mark) - so it is complete, even if "Complete" field is empty	100%
Sub-sub-issue 1.3	This issue has no progress information (neither "Resolution" nor "Complete" fields), so progress is undefined 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



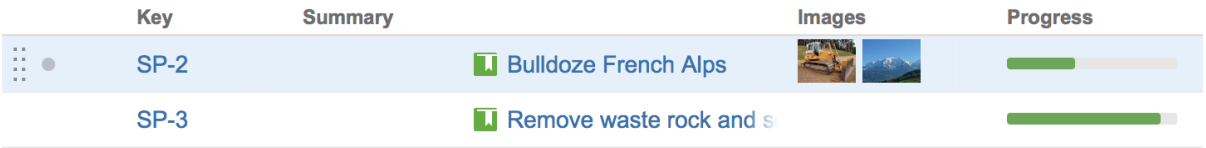
Sample Structure

Issue	Explanation	Progress
Sub-sub-issue 1.1	This issue is 50% complete as specified by custom field and has 2 story points	50%
		100%

Issue	Explanation	Progress
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	
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%
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.



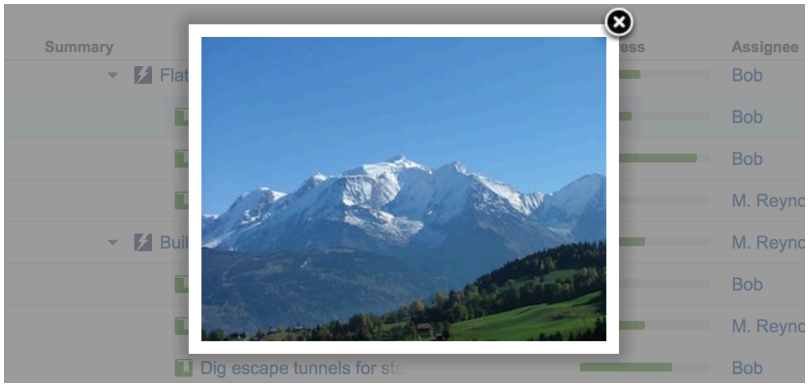
Viewing Full-Size Images

Using your mouse:

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

Work Logged Column

The Work Logged column displays the sum of time spent on an issue over a specific period of time and, optionally, by a specific user.

Work Logged column allows you to select one of the predefined periods using the column configuration panel.

Displaying Aggregate Values

Work Logged column also offers to display an aggregated value, calculated as the sum of time spent over sub-issues.

To display an aggregated value, use the column configuration panel and select **Sum over sub-issues**.

How is Work Logged Calculated?

Each time you log work on some issue you have to define "Time Spent" and "Date Started" values. The Work Logged column will summarize logged time spent over a selected period.

Note that the start of the selected period is calculated based on the column creator's time zone. This time zone can be configured on the user's profile page.

You can also create your own instance of Work Logged column to calculate the sum of time spent over a selected period in your current time zone if work logs are being created in different time zones.

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

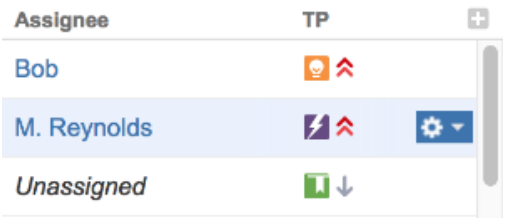
The flags are the small icons displayed at the left side of the table to mark specific issue states.

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 (see page) button.
🔒	**Read-only** flag means that the current user does not have **Edit Issue** permission on this issue, so you cannot edit this issue (see page 158). Additionally, if the structure is configured to require Edit Issue permission on Parent Issue (see page 196), you cannot change or rearrange the immediate children of this issue.

JIRA Actions Column

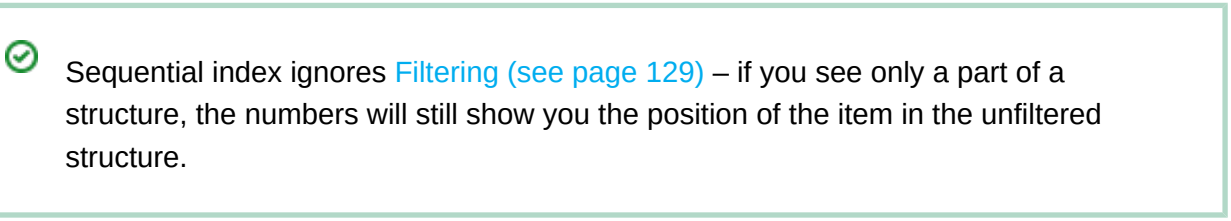
The JIRA actions column displays the gear button that calls out the menu with available JIRA actions for the issue.This column works like the similar column on the JIRA's Issue Navigator page and lets you log work, apply workflow actions and use other JIRA actions (see page 165) available for the issue.



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.

Sequential Index Column

Sequential index column displays the hierarchical number (for example, "1.2.15") based on the position of an item in the structure.



Sequential index ignores Filtering (see page 129) – if you see only a part of a structure, the numbers will still show you the position of the item in the unfiltered structure.

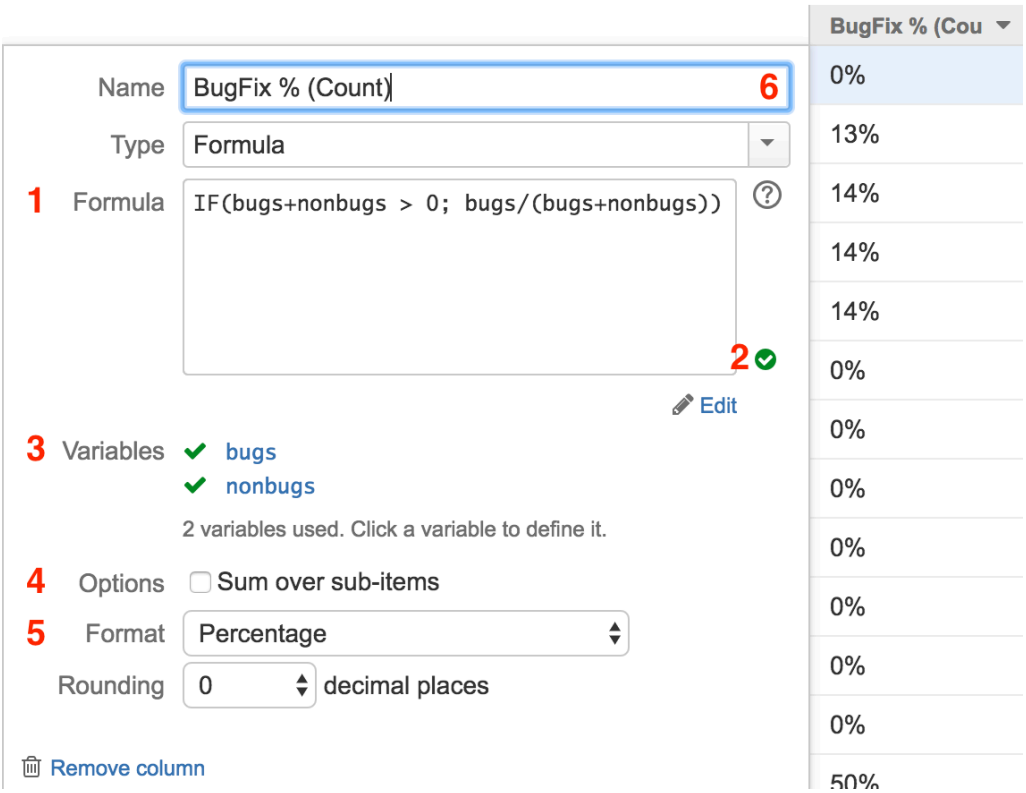
Formula Column

Formula column displays a value that is calculated from issue fields or other attributes using a custom formula. You can use one of the predefined formulas or write your own using a simple expression language.

Configuring Formula Column

Use the following steps to define a new formula.

1. Write Formula



The formula is a simple expression that uses variables (for example, "priority" or "duedate"), arithmetic operations and functions. Variables are linked to issue fields or other attributes and the formula is calculated for every item in the structure. The first thing you need to do is express exactly what you'd like to calculate, using the variables you will need.

See Expr Language (see page 78) for an in-depth description of the language and examples.

2. Verify Formula's Correctness

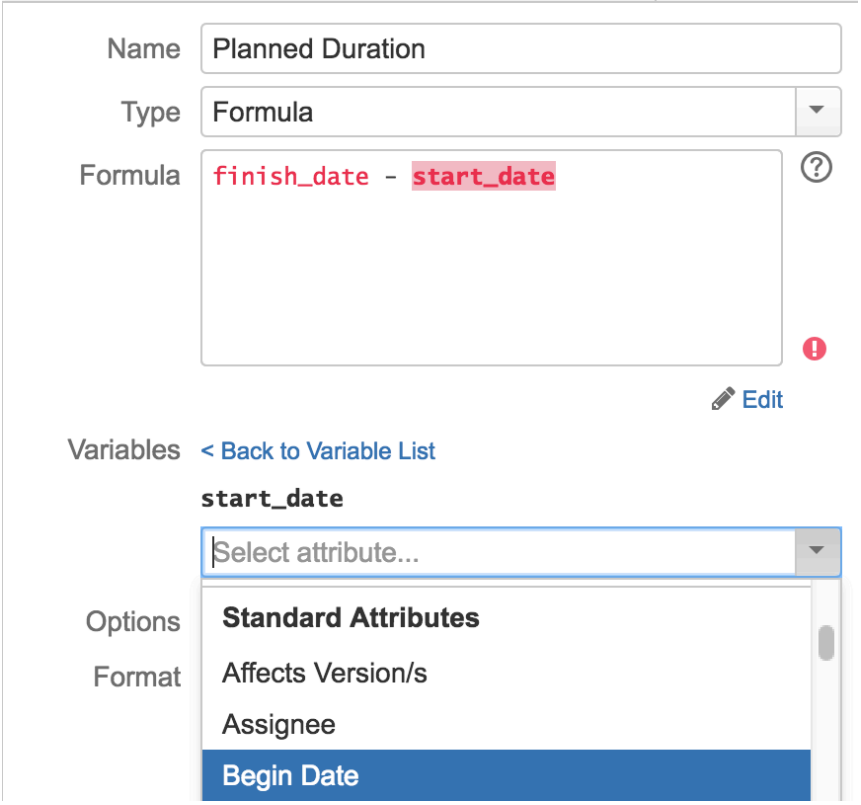
When you stop writing the formula and click **Save**, it is verified and a green mark is displayed if the formula is ready to be used. If it's not, the problematic parts are highlighted in the formula editor with red color.

There might be several problems with your formula:

- A syntax error happens when the formula cannot be parsed, for example, if you forget a closing parenthesis. The editor will highlight the part that failed to get parsed.
- A function resolution error happens when you use an unknown function (probably you have mistyped a known function's name). These functions are shown in red.

- A variable resolution error happens when you have used a variable but it hasn't been defined yet. This error is perfectly ok and you should proceed to the next step, defining the variables.

3. Define Variables



Normally, the formula would involve some variables. (Otherwise the result will be the same for each row in the structure.) These variables need to be mapped to *attributes*, which could be issue fields, progress, a hierarchical total or something else. The configuration dialog displays the list of all variables used in the formula.

If you use a well-know field name for a variable, such as **Priority** or **Assignee**, Structure will automatically assign it to the respective attribute. If the variable name is unfamiliar to Structure, it will remain unassigned and will be clearly marked with red color and icon.

To assign a variable to an attribute, click it in the variable list or in the formula. An attribute selector opens up, where you can select from the list of standard attributes, use a customizable attribute or pick an attribute that is already being used by some of the columns on the screen.

The following names are automatically recognized by Structure:

- Names of the standard JIRA fields, such as **Summary** or **Priority**.

- Names of the custom fields, with all non-letters removed and all spaces converted to underscores, for example, **Story_Points**.
- Names starting with **Total_** or with **Sum_** and having a well-known name afterwards – such as **Sum_Story_Points** or **Total_Estimate**. These are converted to a Sum attribute of the given value. (Without duplicate removal option.)

Even if Structure automatically assigns an attribute to a variable, you might want to double-check it by clicking the variable name.

Please note that if you're selecting an attribute defined in another column (from the "**Used in Columns**" section of the attribute selection drop-down), you **copy** the attribute definition from another column at that moment, rather than creating a link to that column. That means that even if the other column is removed or if its content is changed, the Formula configuration will keep using the attribute as it was defined at the moment you configured the variable.

4. Optionally, Turn On Aggregation

Select **Sum Over Sub-Items** to have each row display a total value, calculated as a sum of the value for the row and values for all sub-items. The usual options to exclude values from duplicates (multiple rows with the same item) and to operate on a filtered structure apply.

Note that sometimes it doesn't make sense to calculate a sum. For example, percent values usually cannot be added together.

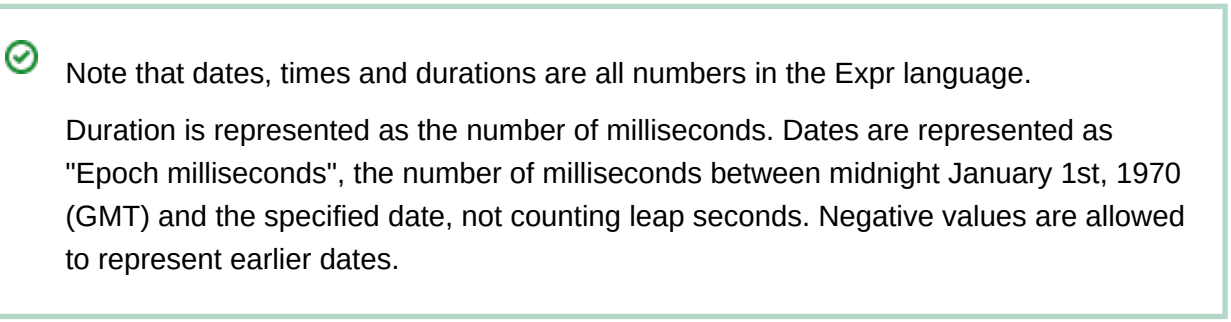
Structure has no knowledge about the meaning of the values, so it will offer to sum them up anyway. It's up to the user to make sure that the calculated value makes sense.

5. Optionally, Select a Number Format

If the result of your formula is a number, you might want to customize the way it is displayed. The following options are available:

- **General** is the usual way numbers are displayed, as-is.
- **Number** format lets you specify the number of decimal places that will always be shown. The value will be rounded up to the least meaningful digit in this format.
- **Percentage** is similar to number, however, the value is treated as a ratio (0.0 = 0%, 1.0 = 100%) and the percent sign is added.

- **Date/Time** option is used to show dates and times and allows to pick a format.
- **Duration** option is used to display a duration values (such as those stored in **Time Spent** field) and format them nicely as number of days/hours/minutes.



Note that dates, times and durations are all numbers in the Expr language.

Duration is represented as the number of milliseconds. Dates are represented as "Epoch milliseconds", the number of milliseconds between midnight January 1st, 1970 (GMT) and the specified date, not counting leap seconds. Negative values are allowed to represent earlier dates.

5.1 About Work Time Option

If option **Work Time** is selected, Structure will use JIRA's time tracking settings to convert the number of hours to the number of days and weeks. By default, JIRA is set up with 8 hours in a day and 5 days in a week. If that option is not selected, hours are converted to days and weeks on a calendar basis.

Whether you need to use that option depends on where the value is coming from.

If you subtract dates (for example, if you want to calculate the number of days the ticket remains open: `now() - created`), then you'd probably want to keep Work Time option off and see calendar duration.

If you operate with the values retrieved from issue's Original Estimate, Remaining Estimate and Time Spent fields (for example, to calculate overspending: `time_spent + remaining_estimate - original_estimate`), then you'd probably use Work Time option.

6. Give Column a Meaningful Name

Structure will try to provide a name based on the formula, but in most cases you'd want to name it according to the meaning of the calculated value.

Sharing Formula Columns

Formula columns are just like any columns, so you can make them a part of a View or create a perspective URL that would open the same configuration, including the formula column.

Sorting by Calculated Value

You can sort by the value calculated in the column by clicking the column header.

See Also

- Advanced Formula Configuration (see page 71)
- Bundled Formulas (see page 75)
- Expr Language (see page 78)

Advanced Formula Configuration

A number of interesting metrics would have a formula that involves sums of other formulas. For example, to calculate the % of bugs in a sub-tree, we'll need to divide the total number of bugs (calculated as a formula) by the total number of issues (also calculated as a formula).

It is possible to do that with the Formula column, but it's a bit tricky. You will need to use temporary additional Formula columns. The following instruction walks you through the creation of the BugFix % column, mentioned above, so that you can solve similar tasks.

> We're aiming to make this instruction obsolete in a future version, where we'll introduce hierarchy aggregates right into the Expr language.

Creating and Advanced Formula Column

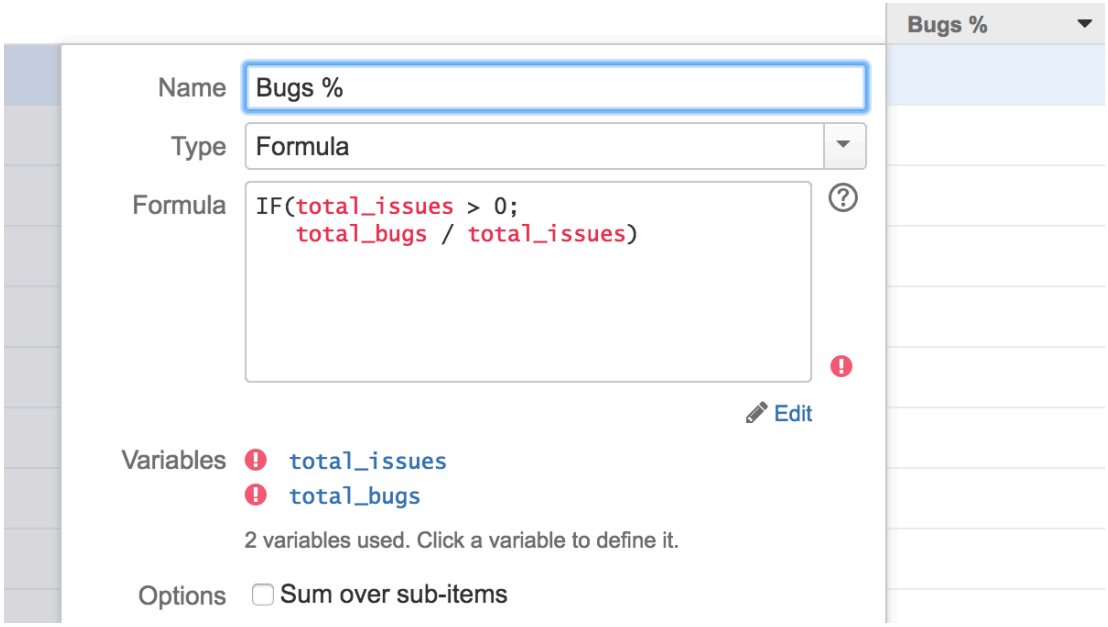
In this walk-though, we will create a column that shows the % of bugs in a subtree. The formula is going to be something like `(total number of bugs) / (total number of issues)`.

1. Main Formula

Create a new Formula column and define the main formula. We're using **total_bugs** and **total_issues** as variable names, but they really could be anything.

A few things to note:

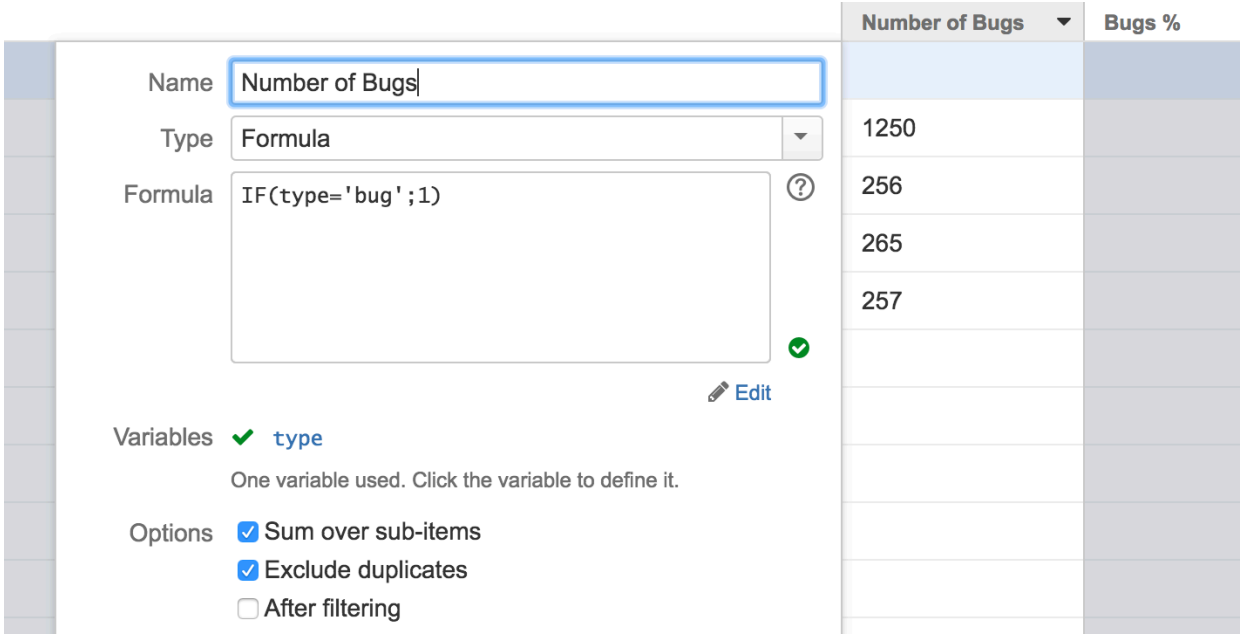
- We don't have total_bugs and total_issues defined at the moment, so they are marked red. That's OK.
- We use `IF()` function to avoid dividing by zero.
- We don't use Sum option because this is a ration. The summing is going to happen when total_bugs and total_issues are calculated.



2. Temporary Column with Total Number of Bugs

Now let's define a new column that shows the value of **total_bugs**. That is quite easy with the formula column.

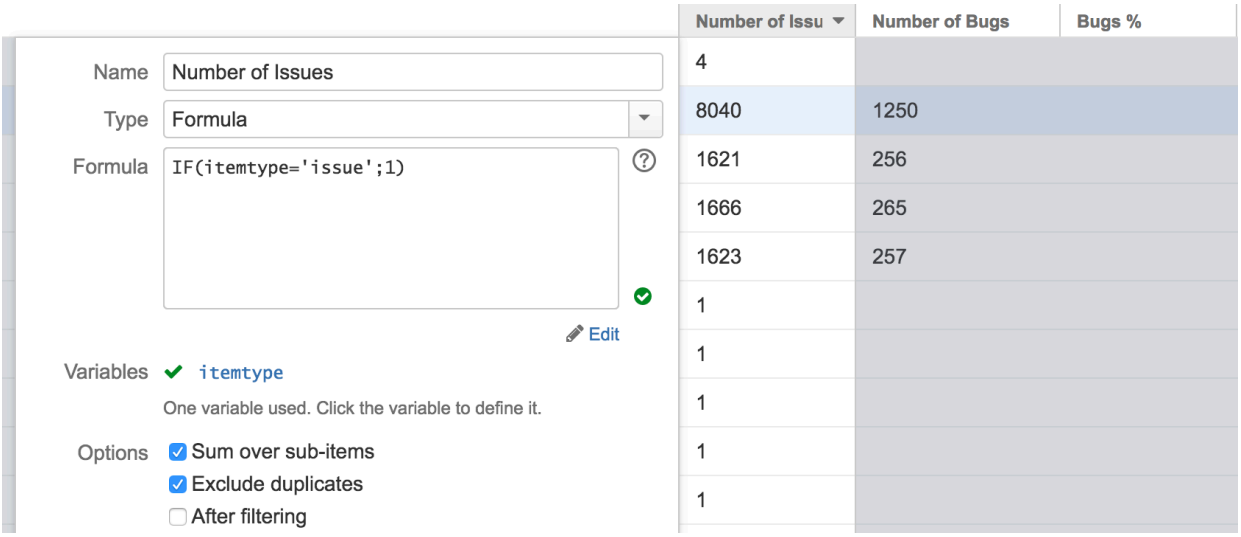
- For an individual row, we use `IF()` function to check if the issue type is "Bug", and produce **1** as the result in that case.
- The values are summed up, with duplicate items excluded.
- **type** variable is automatically mapped to the Issue Type attribute



3. Temporary Column with Total Number of Issues

In the manner we create another temporary column that shows total number of issues.

- Note that we still have to use `IF(),` because items in the sub-tree could be folders or of other non-issue types.

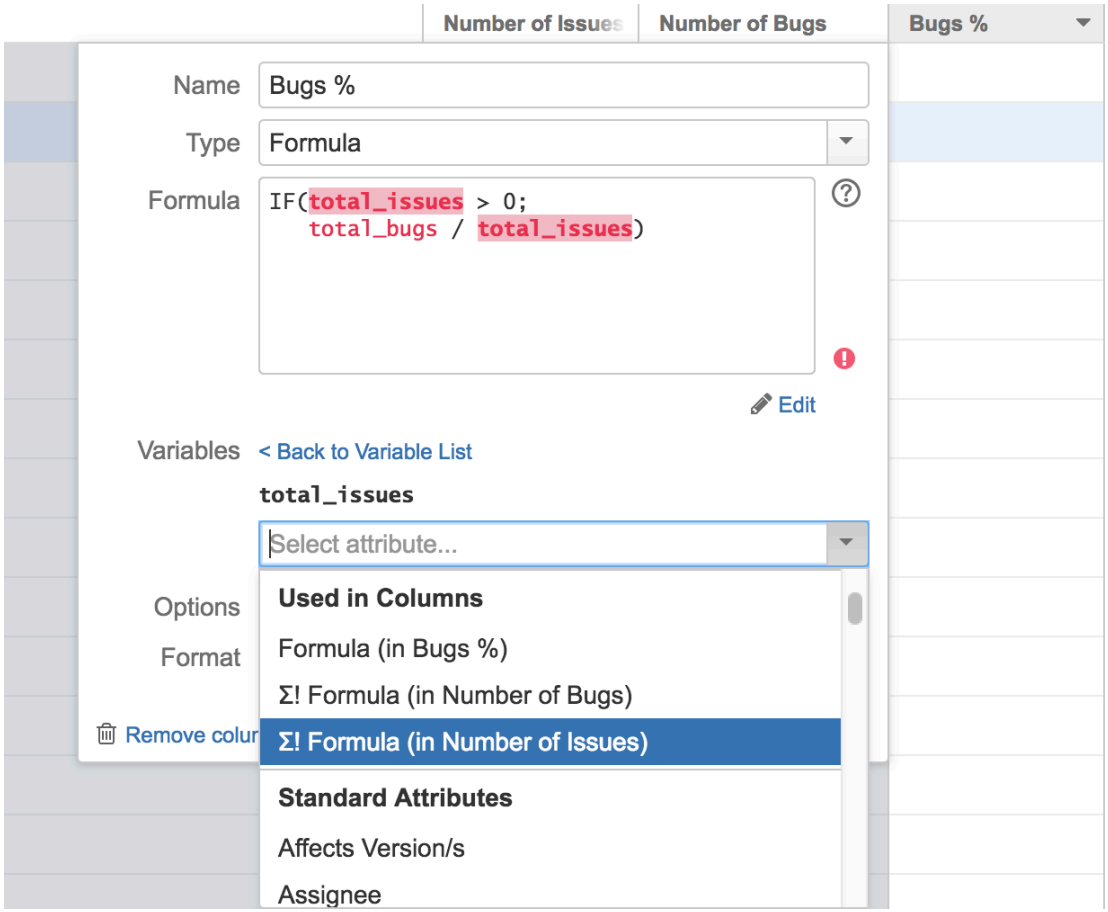


4. Define Main Formula Variables

Now it's time to map the main formula's variables to the values calculated in the temporary columns.

Go back to the main column configuration and click **total_issues** variable. In the attribute selector, scroll down to **Used in Columns** section and select "Formula" entry with the name of the temporary column in parentheses.

Repeat the same for **total_bugs**.



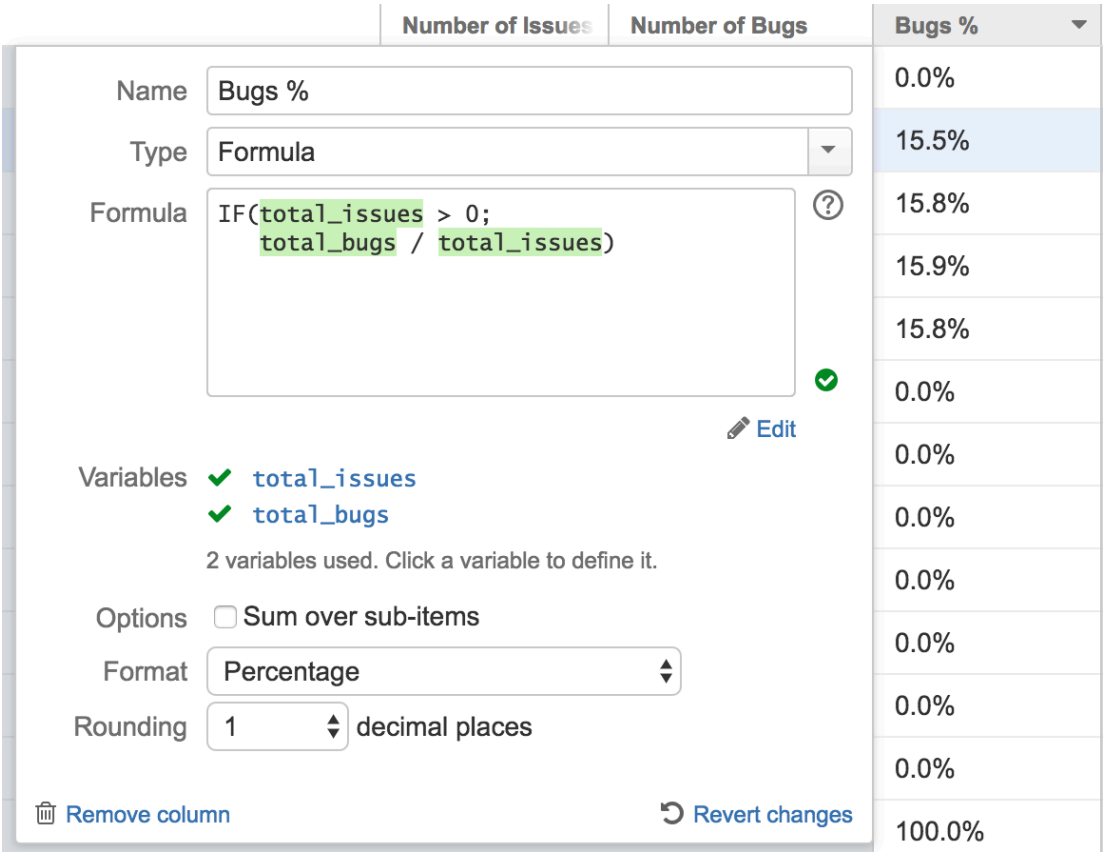
Please note that if you're selecting an attribute defined in another column (from the "**Used in Columns**" section of the attribute selection drop-down), you **copy** the attribute definition from another column at that moment, rather than creating a link to that column. That means that even if the other column is removed or if its content is changed, the Formula configuration will keep using the attribute as it was defined at the moment you configured the variable.

So if you need to change the formula in the temporary column, you can do it, but then you'll need to open the main formula configuration again and re-select the attribute for the variable.

5. Final Steps

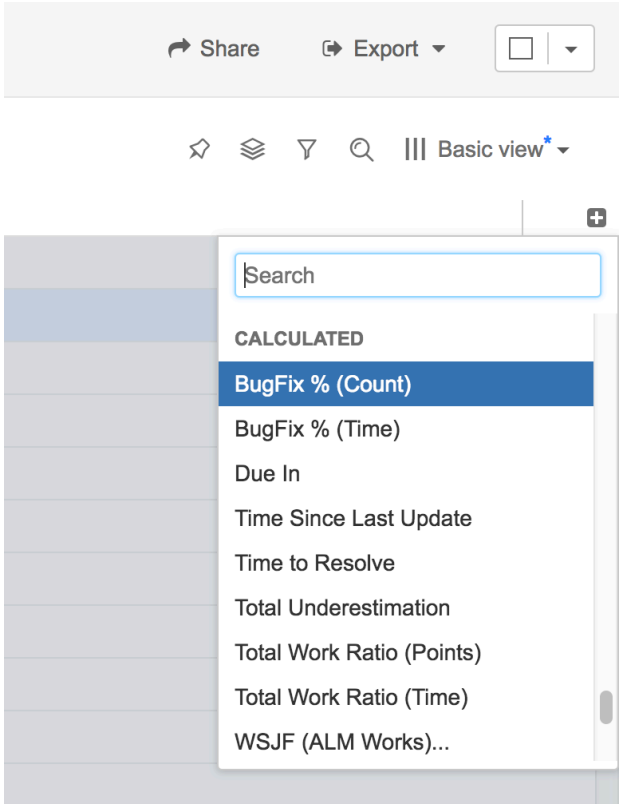
Select Percent as the number format and it's all done!

As a final clean-up, you can remove the temporary columns from the grid. Our Bugs % column has all the information it needs to do the calculation.



Bundled Formulas

Structure add-on comes with several bundled formulas. You can quickly add them to the view by opening **Add Column** dialog and scrolling down to the **Calculated** section.



Here's the description of these predefined values.

Column Name	Description
BugFix % (Count)	Displays the percentage of bugs among all sub-issues. Bugs are identified by having issue type "Bug".
BugFix % (Time)	Displays the percentage of time scheduled and spent on bugs, compared to the time scheduled and spent on all sub-issues. Uses JIRA time tracking fields.
Due In	Displays amount of calendar time left before deadline. Deadline is taken from Due Date field.
Time Since Last Update	Displays amount of calendar time that has passed since the last update of the issue.
Time to Resolve	For resolved issues, displays the amount of calendar time that had passed since issue creation to its resolution.

Column Name	Description
Total Underestimation	Displays the percentage by which total actual time expenditure exceeded total original estimates. Uses total Time Spent and Remaining Estimate fields to calculate the actual time.
Total Work Ratio (Points)	Ratio of total work done to the total amount of work. Amount of work is counted in Story Points, and "done" means a non-empty Resolution field. Totals include values for all sub-issues.
Total Work Ratio (Time)	Ratio of total work done to the total amount of work. Amount of work is based on the sum of Time Spent and Remaining Estimate values. Totals include values for all sub-issues.
WSJF (Basic)	Weighted Shortest Job First metric, based on basic attributes available in any JIRA – Priority, Votes, Watchers, Due Date, Story Points and Remaining Estimate.
WSJF (SAFe)	Weighted Shortest Job First metric, based on recommendations from Scaled Agile Inc. Requires setting up the following numerical fields: • Job Size • User/Business Value • Time Criticality • Risk Reduction • Opportunity Enablement If you have such fields but they are not numeric (for example, a select list), edit the formula and replace the usage of a variable with a CASE() function, where you can assign individual numerical weights to each option.
WSJF (ALM Works)	Weighted Shortest Job First metric, according to categories used at ALM Works: • Benefit • Pain • Marketability • Impact

Column Name	Description
	• Cost • Risk • Clarity. Requires setting up such fields with the following values: Nil, Low, Medium and High.
Assignee Cost	Calculates the dollar amount of the task based by the time (Time Spent + Remaining Estimate) multiplied by the per hour rate for the current Assignee. The rate is taken from the "Hourly Rate" additional property for the user who is the assignee. Shows the total amount.

Expr Language

Expr Language (pronounced like "expert" without "t") is a simple language that lets you specify an "expression", or a formula, which will be calculated for an issue or another item. When you use is in a Formula Column (see page 66), the expression is calculated for each visible row in the displayed structure or query result.

You can see examples of formulas by adding predefined columns in Structure (from the **Calculated** section), and then opening column options (see page 43) panel. The language itself and its grammar is quite obvious and is similar to arithmetic expressions with a number of functions.

If you're familiar with writing formulas in **Microsoft Excel**, you'll recognize a few things in Expr. In particular, if you know some functions that Excel provides, they have a good chance to be supported by Structure as well.

Language Components

An expression may contains one or more of the following:

- Variables, which are bound to specific values when calculating expression for a specific item.
- Functions, which may take some arguments, and which produce the result at the moment of calculation.
- Numbers and text strings.
- Arithmetic, logical operations and parentheses.

Variables

Variables are user-defined names, containing letters (English only), numbers, dot (".") or underscore ("_") characters. The first character should be a letter or an underscore.

Examples:

- `Priority`
- `remaining_estimate`
- `abc11`
- `sprint.name`

When writing an expression, you'd usually associate a name with some value that an issue or another item has. After the expression is written, Formula Column (see page 66) lets you associate the variables used with specific attributes.

A few things to note about variables:

- You can use any names, from a simplistic "x" to a "VeryComplicatedCustomFieldName".
- But, if Formula Column recognizes the variable name to be similar to a field name, it may automatically assign an attribute to the variable. For example, "Priority" variable will be automatically mapped to the similarly named field.
- But, it is possible (although very unreasonable!) to edit the association and assign a variable with a well-known name to something else. ⚠ Please keep that in mind if you need to troubleshoot a formula and double-check the variables.

> Variable names are case-insensitive. `Priority`, `priority` and `pRiOrItY` will refer to the same variable.

Functions

A function calculates some value based on its arguments and, sometimes, some external aspect. A function is written as the function name, followed by parentheses, which might contain arguments.

Examples:

- `SUM(-original_estimate; remaining_estimate; time_spent)`
- `CASE(priority, 'High*', 5, 1)`
- `TODAY()`

There are a number of standard functions available with Structure 4.0 – see Expr Function Reference (see page 82) for details.

A function may take zero, one or more arguments. Some functions take variable number of arguments. Each argument can be another Expr expression and include calls to other functions.

Function arguments may be separated by comma (,) or semicolon (;). But in every function call you need to use either all commas or all semicolons.

Function names are case-insensitive, like the variables. You can write `TODAY()` or `Today()`.

Numbers and Text Strings

Numbers

You can use numbers in your formula. The numbers are always written as a sequence of digits with optionally a dot (".") and a fractional part. Locale-specific, percents, currency or scientific formats are not supported.

Recognized as a number	Not recognized as a number
`0`	`0,0`
`1234567890123456`	`1 100 025`
`11.25`	`1.234e+04`
`.111`	`($100)`

You can write a number that is written with a locale-specific decimal and thousands separator as a text value and it will be automatically converted to a number if needed. For example:

- `"1 122,25" * 2  2244.5`

Text Strings

Text strings are a sequence of characters enclosed either in single (') or double quotes ("). Examples:

- `'a text in single quotes may contain " (a double quote)'`

- `"a text in double quotes may contain ' (a single quote)"`
- `""`

Everything within a text string is retained verbatim to participate in the expression evaluation, except for the following:

- A sequence of two backslashes (`\\`) is converted to a single backslash (`\`).
- A sequence of a backslash and a single quote (`\'`) is converted to a single quote character (`'`) in the text values enclosed in single quotes.
- A sequence of a backslash and a double quote (`\"`) is converted to a double quote character (`"`) in the text values enclosed in double quotes.

Operations

Expr provides basic arithmetic operations, comparisons and logical operations.

The operations follow the general precedence rules for arithmetics, so `A + B * C` is calculated correctly. Comparison operations are done after the arithmetic operations and logical operations are done after comparisons. For detailed specification, see Expr Language Reference (see page 102).

Operations	Comments
`+ - * /`	Convert the value into a number.
`= !=`	Equality and non-equality: if either part of the comparison is a number, the other part is also converted into a number. If both values are strings, then string comparison is used. String comparison ignores leading and trailing whitespace and is case-insensitive (according to JIRA's system locale).
`< <= > >=`	Numerical comparisons – both values are converted to numbers.
`AND`, `OR`, `NOT`	Logical operations.
`( )`	Parentheses can be used to group the results of operations prior to passing to other operations.

See Also

- Expr Function Reference (see page 82)

- Expr Language Reference (see page 102)

Expr Function Reference

All standard Expr functions are listed on this page, grouped by category.

A function may take zero, one or more arguments. Some functions can take unlimited number of arguments.

When a function expects a text or a numeric value as an argument and the actual type of value is different, the function will try to convert the value to the required type. If the conversion is not possible and the value is not empty (for example, it's impossible to convert "ABC" to a number), the result will be an error.

A variable used in a formula may have `undefined` value. Usually it means that the value for an issue is not set – for example, Resolution field will produce `undefined` value until the issue is resolved. When a function that manipulates values receives `undefined` value at its primary argument, the return value will also typically be undefined.

Functions

Conditional Functions

CASE

```
CASE(Value; Match1; Result1; Match2; Result2; ...; DefaultOpt)
```

Checks if the `Value` matches agains several checks and returns a corresponding result.

- `Value` – value to check.
- `Match1`, `Match2`, ..., `MatchN` – text patterns to check against. The first matching pattern will define the result. A pattern can be an exact value, a wildcard expression or a regular expression. See Expr Pattern Matching (see page 117) for details.
- `Result1`, `Result2`, ..., `ResultN` – values to return from the function, each value corresponds to the preceding `Match` parameter.
- `DefaultOpt` – optional default value, to be returned if none of the patterns match. If not specified, `undefined` is returned.

This function is typically used to map text values to numbers.

If the `Value` is `undefined`, the function immediately returns `DefaultOpt` result (or `undefined` if there's no default). There's no sense in using `undefined` as one of the matches.

Examples:

- `CASE(Priority; "Highest"; 10; "High"; 5; "Medium"; 3; 1)`
- `CASE(Version; "V1*"; 1; "V2*"; 2)`

CHOOSE

`CHOOSE(Index; Value1; Value2; ...)`

Based on the value of `Index`, returns the corresponding value from the argument list.

- `Index` – numeric index, with 1 corresponding to `Value1`, 2 corresponding to `Value2` and so on.
- `Value1`, `Value2`, ..., `ValueN` – the values to pick from.

Examples:

- `CHOOSE(1; "A"; "B"; "C")  "A"`
- `CHOOSE(2; "A"; "B"; "C")  "B"`

DEFINED

`DEFINED(Value)`

Checks if the value is defined. Returns false (0) if `Value` is `undefined` and true (1) otherwise.

Example:

- `IF(DEFINED(Resolution); ...)`

DEFAULT

`DEFAULT(Value; DefaultValue)`

Substitutes `DefaultValue` in case `Value` is `undefined`.

Examples:

- `DEFAULT(100; 500)  100`
- `DEFAULT(undefined; 500)  500`

IF

`IF(Condition1; Result1; Condition2; Result2; ...; DefaultOpt)`

Checks one or several conditions, returns the result associated with the first true condition.

- `Condition1`, `Condition2`, ..., `Condition3` – the conditions to check. The values are evaluated using "truthfulness check" – the first value that is "truthy", that is, not undefined, not zero and not an empty string, will define the returned value.
- `Result1`, `Result2`, ..., `ResultN` – results to be returned, each result corresponding to the preceding check.
- `DefaultOpt` – optional default value to be returned if none of the conditions are true. If omitted, `undefined` is returned.

Examples:

- `IF(Estimate > 0; Duration / Estimate; 0)`
- `IF(N = 0; "No apples"; N = 1; "One apple"; CONCAT(N; " apples"))`

IFERR

`IFERR(Value; FallbackValue)`

Checks if calculating `Value` produced an error and substitutes `FallbackValue` instead of error value.

Normally, if an error occurs while calculating a formula, it is propagated upwards and the result of the whole expression will be an error. This function helps circumvent that.

Example:

- `IFERR(100 / 0; 100)  100`

ISERR

`ISERR(Value; ErrorCodeOpt)`

Checks if calculating value produced an error. Returns true (1) if there was an error. If `ErrorCodeOpt` is specified, returns true only if the error was of the specified error code.

- `Value` – value to check.
- `ErrorCodeOpt` – optional error code. See Expr Error Codes (see page 118) for a list.

Examples:

- `ISERR("Ham")  0`
- `ISERR(1 / 0)  1`
- `ISERR(1 / 0, 4)  1`

Numeric Functions

ABS

`ABS(Value)`

Calculates absolute value of a number.

Examples:

- `ABS(5)  5`
- `ABS(-4)  4`

CEILING

`CEILING(Value; N)`

Rounds value up to the N^{th} decimal place.

- `Value` – a number to round.
- `N` – how many decimal places to round up to, can be negative to round up to tens, hundreds, etc. Default value: `0` (round to an integer).

Examples:

- `CEILING(1.678)  2`
- `CEILING(12.34; 1)  12.4`
- `CEILING(12.34; -1)  20`
- `CEILING(-3.14)  -3`

FLOOR

`FLOOR(Value; N)`

Rounds value down to the Nth decimal place.

- `Value` – a number to round.
- `N` – how many decimal places to round down to, can be negative to round up to tens, hundreds, etc. Default value: `0` (round to an integer).

Examples:

- `FLOOR(1.678)  1`
- `FLOOR(12.34; 1)  12.3`
- `FLOOR(12.34; -1)  10`
- `FLOOR(-3.14)  -4`

MAX

`MAX(Value; ...)`

Selects numerically largest value from all values passed as arguments. `Undefined` values are skipped. Text values that cannot be converted to a number will also be skipped.

Examples:

- `MAX(Due_Date; Updated_Date)`
- `MAX(0; -10; undefined; 10)  10`

MIN

`MIN(Value; ...)`

Selects numerically smallest value from all values passed as arguments. `Undefined` values are skipped. Text values that cannot be converted to a number will also be skipped.

Example:

- `MIN(0; -10; undefined; 10)  -10`

MOD

`MOD(A; N)`

Returns remainder from dividing A by N.

- `A` – the dividend, must be an integer.
- `N` – the divisor, must be an integer.

Example:

- `MOD(17; 5)  2`

POW

`POW(B; E)`

Produces B to the power of E (B^E). Both values can be fractional.

- `B` – base
- `E` – exponent

Example:

- `POW(27; 1/3)  3`

ROUND

`ROUND(Value; N)`

Produces a rounded value up to the N^{th} decimal place.

- `Value` – a number to round.
- `N` – how many decimal places to round to, can be negative to round up to tens, hundreds, etc. Default value: `0` (round to an integer).

Examples:

- `ROUND(1.678)  2`
- `ROUND(12.34; 1)  12.3`
- `ROUND(12.34; -1)  10`

SIGN

`SIGN(Value)`

Returns sign of the Value.

Examples:

- `SIGN(123)  1`
- `SIGN(0)  0`
- `SIGN(-123)  -1`

SQR

`SQR(Value)`

Returns the passed numerical value, squared.

Example:

- `SQR(5)  25`

SQRT

`SQRT(Value)`

Returns the square root of the passed numerical value.

Example:

- `SQRT(25)  5`

Text Functions

Text function let you manipulate character strings.

If a function expects a string but encounters a number, it converts it to a string using mathematical notation ("." decimal separator, no thousands separator).

CONCAT

`CONCAT(Value; ...)`

Concatenates strings together. Accepts any number of arguments. Ignores undefined values.

Example:

- `CONCAT(Reporter; ' => '; Assignee)`

EXACT

`EXACT(A; B)`

Checks if text value A is exactly the same as text value B.

This comparison is case sensitive, which is different from comparing A with B using equals sign or text matching. Undefined values will be equal to each other and to empty strings.

Examples:

- `EXACT("Fox"; "fox")  0`
- `EXACT("Fox"; "Fox")  1`
- `EXACT(""; undefined)  1`

LEFT

`LEFT(Value; N)`

Returns up to N leftmost characters from string value.

- `Value` – string to get characters from.
- `N` – the number of characters to get. If Value contains fewer characters, all of them are returned.

Example:

- `LEFT("abc"; 2)  "ab"`

LEN

`LEN(Value)`

Returns the number of characters in a string value. If the value is not a string, it is converted to string first.

Example:

- `LEN("abc")  3`

LOWER

`LOWER(Value)`

Converts the string to lowercase. The locale of the current user is applied.

Example:

- `LOWER("HAM")  "ham"`

MATCH

`MATCH(Value; Pattern)`

Checks if the Value matches the Pattern. Returns `true` (1) or `false` (0).

- `Value` – the value to check.
- `Pattern` – pattern to check against. Can be an exact value, a wildcard expression or a regular expression. See Expr Pattern Matching (see page 117) for details.

Examples:

- `MATCH("Apples"; "Oranges")  0`
- `MATCH(" Blocker "; "blocker")  1`
- `MATCH("Hamster"; "ham*")  1`
- `MATCH("The Flight of the Bumblebee"; "/.light.*beer?/")  1`

MID

`MID(Value; Index; Count)`

Retrieves a part of the text.

- `Value` – the string value to get a substring from.
- `Index` – the starting index of the part to retrieve, 1-based (first character is at index 1).
- `Count` – the number of characters to retrieve.

Example:

- `MID("A quick brown fox"; 3; 5)  "quick"`

REPEAT

`REPEAT(Value; N)`

Produces a text that is a repetition of the string value N times.

- `Value` – a string value to repeat.
- `N` – the number of repetitions.

Examples:

- `REPEAT("ha"; 3)  "hahaha"`
- `REPEAT(123, 3)  "123123123"`

REPLACE

`REPLACE(Value; Pattern; Replacement)`

Replaces all occurrences of Pattern with Replacement and returns the new string.

- `Value` – the value to manipulate.
- `Pattern` – pattern to find. Can be an exact value, a wildcard expression or a regular expression. See Expr Pattern Matching (see page 117) for details.
- `Replacement` – an optional string to use instead of the matched parts. If omitted, the matched parts are removed.

Examples:

- `REPLACE("I like cats"; "CAT"; "DOG")  "I like DOGs"`
- `REPLACE("Can you read this?"; "/[aeuio]/")  "Cn rd ths?"`

REPLACE_AT

`REPLACE_AT(Value; Index; Count; Replacement)`

Replaces a specific part of the Value with Replacement string and returns the value.

- `Value` – the string to manipulate.
- `Index` – the starting index of the part to replace, 1-based (first character is 1, second is 2, etc.)
- `Count` – the number of characters to replace. When Count is 0, the Replacement string gets inserted at the Index position.
- `Replacement` – optional string to use instead of the replaced part. If omitted, the part will be deleted.

When the values of Index and Count are out of range, they are brought to the nearest sensible value.

Examples:

- `REPLACE_AT("A"; 1; 1; "B")  "B"`
- `REPLACE_AT("What does the fox say?"; 6; 4; "did")  "What did the fox say?"`
- `REPLACE_AT("A step for mankind"; 3; 0; "small ")  "A small step for mankind"`
- `REPLACE_AT("A step for mankind"; 7; 1000)  "A step"`

RIGHT

`RIGHT(Value; N)`

Returns up to N rightmost characters from string value.

- `Value` – string to get characters from.
- `N` – the number of characters to get. If Value contains fewer characters, all of them are returned.

Example:

- `RIGHT("abc"; 2)  "bc"`

SEARCH

`SEARCH(Pattern; Value; Index)`

Finds the first occurrence of a pattern in the value. Returns the index of the matched part (1-based), or `undefined` if not found.

- `Pattern` – the string or pattern to look for. Can be an exact value, a wildcard expression or a regular expression. See Expr Pattern Matching (see page 117) for details.
- `Value` – the string to search in.
- `Index` – optional parameter that provides an index to start searching at.

Examples:

- `SEARCH("ham"; "The Ham is for the Hamster"; 6)  20`
- `SEARCH("Jedi*"; "Return of the Jedi")  15`
- `SEARCH("/^Jedi/"; "Not the Jedi you're looking for")  undefined`

SUBSTRING

`SUBSTRING(Value; From; To)`

Returns a substring, indicated by a starting index and ending index. Note that the indexes are 0-based, unlike in some other functions.

- `Value` – the string to take the part from.

- `From` – starting index, inclusive, 0 means the first character, `LEN(Value)-1` means the last character.
- `To` – optional ending index, exclusive - the character at this index will not be included. If omitted, the substring will include all characters up to the end of the `Value`.

Examples:

- `SUBSTRING("Batman"; 0; 3)  "Bat"`
- `SUBSTRING("Batman"; 3)  "man"`

TRIM

`TRIM(Value)`

Removes leading and trailing whitespace from the text.

Example:

- `TRIM(" Batman ")  "Batman"`

UPPER

`UPPER(Value)`

Converts the string to uppercase. The locale of the current user is applied.

Example:

- `UPPER("ham")  "HAM"`

Date and Time Functions

Date/time functions operate with numeric representation of time. A moment in time is represented as a number of milliseconds since midnight, January 1st 1970, GMT. Negative values are allowed.

To display a result of a date/time calculation in a readable way, you need to either configure Formula Column (see page 66) to use a date/time format, or use one of the conversion functions to turn the value into a human-readable text.

Many of the date / time functions depend on the current user's time zone.

DATE

`DATE(Text; LocaleOpt; TimeZoneOpt)`

Converts text representation of a date to number. The resulting timestamp will correspond to midnight of the specified date at the specified timezone.

- `Text` – the text value to convert.
- `LocaleOpt` – optional locale identifier, such as "fr_FR". If not specified, user's locale is used.
- `TimeZoneOpt` – optional time zone identifier, such as "America/New_York".

The conversion uses tries the standard formats for representing dates:

- Format "yyyy-MM-dd", like "2017-04-15".
- Standard formats for the specified locale.
- JIRA formats, as specified in the JIRA's system settings.

If conversion is unsuccessful, returns an error.

Examples:

- `DATE("2016-01-01")`
- `DATE("31/Dec/16")`
- `DATE("12/31/2016", "en_US", "America/New_York")`

DATE_ADD

`DATE_ADD(DateTime, Number, Unit)`

Adds the specified number of seconds, minutes, hours, days, months or years to the date or date/time value.

- `DateTime` – date or date/time value.
- `Number` – the number of units of time to add.
- `Unit` – a text value specifying the unit of time: `"seconds"`, `"minutes"`, `"hours"`, `"days"`, `"months"`, `"years"`

Examples:

- `DATE_ADD(DATE("2016-01-31"), 1, "day")  DATE("2016-02-01")`
- `DATE_ADD(DATE("2016-01-31"), 1, "month")  DATE("2016-02-29")`
- `DATE_ADD(DATE("2016-02-29"), 1, "year")  DATE("2017-02-28")`
- `DATE_ADD(DATETIME("2016-01-31 10:30:00"), 3, "hours")  DATETIME ("2016-01-31 13:30:00")`
- `DATE_ADD(DATETIME("2016-01-31 23:59:59"), 2, "minutes")  DATETIME ("2016-02-01 00:01:59")`

DATE_SET

`DATE_SET(DateTime, Number, Unit)`

Sets the specified part of the date or date/time to the specific value. Note that unlike `DATE_ADD` and `DATE_SUBTRACT`, you can specify additional units like "day_of_week".

- `DateTime` – date or date/time value.
- `Number` – the number to be set as the unit value in this date/time.

- `Unit` – a text value specifying the unit of time: `"second"`, `"minute"`, `"hour"`, `"day"`, `"month"`, `"year"`, `"day_of_week"`.

Examples:

- `DATE_SET(DATE("2016-01-31"), 2017, "year")  DATE("2017-01-31")`
- `DATE_SET(DATE("2016-01-31"), 2, "month")  DATE("2016-02-29")`
- `DATE_SET(DATETIME("2016-02-29 15:30"), 10, "day")  DATETIME ("2016-02-10 15:30")`
- `DATE_SET(DATE("2017-04-01"), 7, "day_of_week")  DATE("2017-04-02")`
- `DATE_SET(DATETIME("2016-01-31 10:30:00"), 0, "hour")  DATETIME ("2016-01-31 00:30:00")`

DATE_SUBTRACT

`DATE_SUBTRACT(DateTime, Number, Unit)`

Subtracts the specified number of seconds, minutes, hours, days, months or years from the date or date/time value.

- `DateTime` – date or date/time value.
- `Number` – the number of units of time to subtract.
- `Unit` – a text value specifying the unit of time: `"seconds"`, `"minutes"`, `"hours"`, `"days"`, `"months"`, `"years"`

Examples:

- `DATE_SUBTRACT(DATE("2016-02-01"), 1, "day")  DATE("2016-01-31")`
- `DATE_SUBTRACT(DATE("2016-02-29"), 1, "month")  DATE("2016-01-29")`
- `DATE_SUBTRACT(DATE("2017-02-28"), 1, "year")  DATE("2016-02-28")`
- `DATE_SUBTRACT(DATETIME("2016-01-31 10:30:00"), 3, "hours") DATETIME("2016-01-31 07:30:00")`
- `DATE_SUBTRACT(DATETIME("2016-02-01 00:01:59"), 2, "minutes") DATETIME("2016-01-31 23:59:59")`

DAY

`DAY(DateTime)`

Returns the day of the month for the give date or date/time value. The result is calculated using current user's time zone.

Example:

- `DAY(DATE("2017-04-15"))  15`

DAYS_BETWEEN

`DAYS_BETWEEN(DateTime1, DateTime2)`

Calculates the number of full days (24 hour periods) between two date or date/time values. Returns negative value if `DateTime2` occurs earlier than `DateTime1`.

Examples:

- `DAYS_BETWEEN(DATE("2017-01-01"), DATE("2017-02-01"))  31`
- `DAYS_BETWEEN(DATE("2017-01-01"), DATE("2017-01-01"))  0`
- `DAYS_BETWEEN(DATE("2017-01-01"), DATE("2016-01-01"))  -366`
- `DAYS_BETWEEN(DATETIME("2017-01-01 00:00"), DATETIME("2017-01-01 23:59"))  0`
- `DAYS_BETWEEN(DATETIME("2017-01-01 23:59"), DATETIME("2017-01-02 23:58"))  0`
- `DAYS_BETWEEN(DATETIME("2017-01-01 23:59"), DATETIME("2017-01-02 23:59"))  1`

DATETIME

`DATETIME(Text; LocaleOpt; TimeZoneOpt)`

Converts text representation of a date and time to number. The resulting timestamp will correspond to the specified date and time at the specified timezone. If seconds are omitted, they will be set to zero.

- `Text` – the text value to convert.
- `LocaleOpt` – optional locale identifier, such as "fr_FR". If not specified, user's locale is used.
- `TimeZoneOpt` – optional time zone identifier, such as "America/New_York".

The conversion uses tries the standard formats for representing dates:

- Format "yyyy-MM-dd HH:mm:ss" and the same without seconds, like "2017-04-15 15:00" or "2017-12-31 23:59:59" (using 24-hour clock).
- Standard formats for the specified locale.
- JIRA formats, as specified in the JIRA's system settings.

If conversion is unsuccessful, returns an error.

Examples:

- `DATETIME("2016-01-01 00:01")`
- `DATETIME("31/Dec/16 3:15 pm")`

- `DATETIME("12/31/2016 3:15 PM", "en_US", "America/New_York")`

END_OF_MONTH

`END_OF_MONTH(DateTime)`

Sets the day in the date/time value to the end of the month. Does not change the time value.

Example:

- `END_OF_MONTH(DATE("2017-04-15"))  DATE("2017-04-30")`

FORMAT_DATETIME

`FORMAT_DATETIME(DateTime, Format, LocaleOpt, TimeZoneOpt)`

Advanced function to convert a date/time value into a text. Accepts arbitrary format string and optional locale and time zone settings. Does not depend on the current user's locale nor time zone.

- `DateTime` – the value to convert.
- `Format` – the format string. For all the options, please see Java documentation for SimpleDateFormat.
- `LocaleOpt` – the optional locale identifier. If omitted or `undefined`, will use JIRA's system locale. (Not the user's locale!)
- `TimeZoneOpt` – the optional time zone identifier. If omitted or `undefined`, will use JIRA's system time zone. (Not the user's time zone!)

Examples:

- `FORMAT_DATETIME(DATE("2017-04-15"), "EEE, MMM d, `yy", "fr_FR") "sam., avr. 15, `17"`
- `FORMAT_DATETIME(DATETIME("2016-12-31 23:59"), "yyyy-MM-dd'T'HH:mm:ss")  "2016-12-31T23:59:00"`

HOUR

`HOUR(DateTime)`

Returns the hour in the specified date/time value (from 0 to 23).

Example:

- `HOUR(DATETIME("2017-01-01 20:15"))  20`

HOURS_BETWEEN

`HOURS_BETWEEN(DateTime1, DateTime2)`

Calculates the number of full hours between two date/time values. Returns negative value if `DateTime2` occurs earlier than `DateTime1`.

Examples:

- `HOURS_BETWEEN(DATE("2017-01-01"), DATE("2017-01-02"))  24`
- `HOURS_BETWEEN(DATETIME("2017-01-01 15:00"), DATETIME("2017-01-01 16:30"))  1`
- `HOURS_BETWEEN(DATETIME("2017-01-01 23:59"), DATETIME("2017-01-02 00:58"))  0`
- `HOURS_BETWEEN(DATETIME("2017-01-01 23:59"), DATETIME("2017-01-02 00:59"))  1`

MAKE_DATE

`MAKE_DATE(Year, Month, Day)`

Creates a date value based on the numbers defining year, month and day. The time is set to midnight in the user's time zone.

Example:

- `MAKE_DATE(2017, 12, 31)`

MAKE_DATETIME

`MAKE_DATETIME(Year, Month, Day, Hour, Minute, Second)`

Creates a date/time value based on the numbers defining year, month, day, hour, minute and second. The current user's time zone is used. The valid values for `Hour` are 0–23.

Example:

- `MAKE_DATETIME(2017, 12, 31, 23, 59, 59)`

MINUTE

`MINUTE(DateTime)`

Returns the minute in the specified date/time value (from 0 to 59).

Example:

- `MINUTE(DATETIME("2017-01-01 20:15"))  15`

MONTH

`MONTH(DateTime)`

Returns the month in the specified date/time value (from 1 to 12).

Example:

- `MONTH(DATE("2017-04-15"))  4`

MONTHS_BETWEEN

`MONTHS_BETWEEN(DateTime1, DateTime2)`

Calculates the number of months between two date or date/time values. Returns negative value if `DateTime2` occurs earlier than `DateTime1`.

Examples:

- `MONTHS_BETWEEN(DATE("2017-01-01"), DATE("2018-01-01"))  12`
- `MONTHS_BETWEEN(DATE("2017-01-31"), DATE("2017-02-28"))  0`
- `MONTHS_BETWEEN(DATE("2017-02-28"), DATE("2017-04-28"))  2`
- `MONTHS_BETWEEN(DATE("2017-01-01"), DATE("2016-12-01"))  -1`

NOW

`NOW()`

Returns the current date and time.

Example:

- `NOW()`

PARSE_DATETIME

`PARSE_DATETIME(Text, Format, LocaleOpt, TimeZoneOpt)`

Advanced function to convert a text into a date or date/time value. Accepts arbitrary format string and optional locale and time zone settings. Does not depend on the current user's locale nor time zone.

- `Text` – the value to convert.
- `Format` – the format string. For all the options, please see Java documentation for SimpleDateFormat.
- `LocaleOpt` – the optional locale identifier. If omitted or `undefined`, will use JIRA's system locale. (Not the user's locale!)
- `TimeZoneOpt` – the optional time zone identifier. If omitted or `undefined`, will use JIRA's system time zone. (Not the user's time zone!)

Examples:

- `PARSE_DATETIME("sam., avr. 15, `17", "EEE, MMM d, `yy", "fr_FR")  DATE("2017-04-15")`
- `PARSE_DATETIME("2016-12-31T23:59:00", "yyyy-MM-dd'T'HH:mm:ss")  DATETIME("2016-12-31 23:59")`

SECOND

`SECOND(DateTime)`

Returns the second in the specified date/time value.

Example:

- `SECOND(DATETIME("2017-04-15 15:30:59"))  59`

START_OF_MONTH

`START_OF_MONTH(DateTime)`

Sets the day in the date/time value to the first day of month.

Example:

- `START_OF_MONTH(DATE("2017-04-15"))  DATE("2017-04-01")`

TODAY

`TODAY()`

Returns the current date with time set to midnight according to the current user's time zone.

Example:

- `TODAY()`

TRUNCATE_TIME

`TRUNCATE_TIME(DateTime)`

Removes time value from the date/time, setting it to midnight in the current user's time zone.

Example:

- `TRUNCATE_TIME(DATETIME("2017-01-01 15:15"))  DATE("2017-01-01")`

TRUNCATE_TO_HOURS

`TRUNCATE_TO_HOURS(DateTime)`

Removes minutes, seconds and milliseconds from the date/time, setting it to the last even hour in the current user's time zone.

Example:

- `TRUNCATE_TO_HOURS(DATETIME("2017-01-01 15:15"))  DATE("2017-01-01 15:00")`

TRUNCATE_TO_MINUTES

`TRUNCATE_TO_MINUTES(DateTime)`

Removes seconds and milliseconds from the date/time, setting it to the last even minute.

Example:

- `TRUNCATE_TO_MINUTES(DATETIME("2017-01-01 15:15:15"))  DATE("2017-01-01 15:15:00")`

TRUNCATE_TO_SECONDS

`TRUNCATE_TO_SECONDS(DateTime)`

Removes milliseconds from the date/time.

Example:

- `TRUNCATE_TO_SECONDS(NOW())`

WEEKDAY

`WEEKDAY(DateTime)`

Returns the number of the day in the week, following ISO-8601 standard (1 – Monday, 7 – Sunday).

Example:

- `WEEKDAY(DATE("2017-04-23"))  7`

YEAR

`YEAR(DateTime)`

Returns the year in a date or date/time value as a number.

Example:

- `YEAR(DATE("2017-04-23"))  2017`

YEARS_BETWEEN

`YEARS_BETWEEN(DateTime1, DateTime2)`

Calculates the number of years between two date or date/time values. Returns negative value if `DateTime2` occurs earlier than `DateTime1`.

Examples:

- `YEARS_BETWEEN(DATE("2017-01-01"), DATE("2018-01-01"))  1`
- `YEARS_BETWEEN(DATE("1703-05-27"), DATE("2017-04-23"))  313`
- `YEARS_BETWEEN(DATE("2017-06-01"), DATE("2018-05-31"))  0`

Duration Functions

Duration is represented as a number of milliseconds. To create a value or to make sense of a value, you need the following function to convert a string to a duration and vice versa.

Note that you can add duration to a date or date/time value and treat the result as a new date /time, but only if the duration is a "calendar duration", not "work duration". Work duration is when JIRA's settings, like the number of work hours per day, are used – the value `"2d"` really becomes 16 hours in JIRA's default setup, and if you add 16 hours to a date, you won't get a date 2 days later.

CALENDAR_DAYS

`CALENDAR_DAYS(Duration)`

Returns a number of calendar days represented by the duration value as a decimal number. May return fractional number of days.

Examples:

- `CALENDAR_DAYS(DURATION("10d"))  10`

- `CALENDAR_DAYS(DURATION("12h"))  0.5`

CALENDAR_HOURS

`CALENDAR_HOURS(Duration)`

Returns a number of hours represented by the duration value as a decimal number. May return fractional number of hours.

Examples:

- `CALENDAR_HOURS(DURATION("10d"))  240`
- `CALENDAR_HOURS(DURATION("12h 45m"))  12.75`

CALENDAR_MINUTES

`CALENDAR_MINUTES(Duration)`

Returns a number of minutes represented by the duration value as a decimal number. May return fractional number of minutes.

Example:

- `CALENDAR_MINUTES(DURATION("3h"))  180`

CALENDAR_SECONDS

`CALENDAR_SECONDS(Duration)`

Returns a number of seconds represented by the duration value as a decimal number. May return fractional number of seconds.

Example:

- `CALENDAR_SECONDS(DURATION("1h"))  3600`

DURATION

`DURATION(Text)`

Converts a text representation of a calendar duration to a number. The format is provided by JIRA – the text may be several numbers, each number followed by a symbol to specify the time unit: `w` for weeks, d for days, h for hours and `m` for minutes.

Note that this function ignores JIRA's settings for work time, so `DURATION("1w") = DURATION("7d")` and `DURATION("1d") = DURATION("24h")`.

Examples:

- `DURATION("1w 2d 3h 4m")`
- `DURATION("3d")`

FORMAT_DURATION

`FORMAT_DURATION(Duration)`

Converts duration value to the JIRA's format with numbers followed by symbols specifying the time unit.

Example:

- `FORMAT_DURATION(DURATION("1w 1d"))  "1w 1d"`

JIRA_DAYS

`JIRA_DAYS(Duration)`

Returns a number of work days in the specified duration according to JIRA's settings. (By default, one day is 8 hours.) May return fractional number.

Example:

- `JIRA_DAYS(DURATION("24h"))  3`
- `JIRA_DAYS(DURATION("12h"))  1.5`

JIRA_DURATION

`JIRA_DURATION(Text)`

Converts a text representation of a JIRA work duration to a number. The format is provided by JIRA – the text may be several numbers, each number followed by a symbol to specify the time unit: `w` for weeks, `d` for days, `h` for hours and `m` for minutes.

The specified time is work time, according to JIRA's settings. With the default JIRA settings, `JIRA_DURATION("1w") = JIRA_DURATION("5d")` and `JIRA_DURATION("1d") = JIRA_DURATION("8h")`.

Examples:

- `JIRA_DURATION("1w 2d 3h 4m")`
- `JIRA_DURATION("3d")`

JIRA_WEEKS

`JIRA_WEEKS(Duration)`

Returns a number of work weeks in the specified duration according to JIRA's settings. (By default, one week is 5 work days.) May return fractional number.

Example:

- `JIRA_WEEKS(JIRA_DURATION("10d"))  2`
- `JIRA_WEEKS(DURATION("5d"))  3`

Miscellaneous Functions

ME

`ME()`

Returns the user key of the current user.

Example:

- `IF(ME() = "admin"; "You're admin!")`

NUMBER

`NUMBER(Value)`

Converts value to number. This function is rarely needed because conversion to number happens automatically when needed.

Example:

- `NUMBER("1.234")  1.234`

TEXT

`TEXT(Value)`

Converts value to text. This function is rarely needed because conversion to text happens automatically when needed.

Example:

- `TEXT(1.234)  "1.234"`

Expr Language Reference

Expr language defines expressions, which are evaluated in the context of an item in some structure. This article describes the syntax of the language and the rules that govern the evaluation.

Conventions

- Similarity to Excel formula language was a design goal, so if you're unsure how Expr behaves, think Excel.
- The language is case-insensitive.
- Whitespace is not meaningful. It is only required to separate word operators and identifiers, in all other cases there can be arbitrary number of whitespace symbols.
- Currently language constructs support only English letters and a few punctuation symbols. However, values can contain any Unicode symbols.

Values

All expressions, when evaluated, produce either a value or an error. All values in Expr are either numbers, text, or a special value called *undefined*. The simplest expression thus is the literal representation of some fixed value. The forms of these literal representations are described below.

Undefined

Undefined value is represented by the word `undefined`.

Undefined value is used when the variable value is not specified. For example, variable `Assignee` has value undefined if the issue is unassigned.

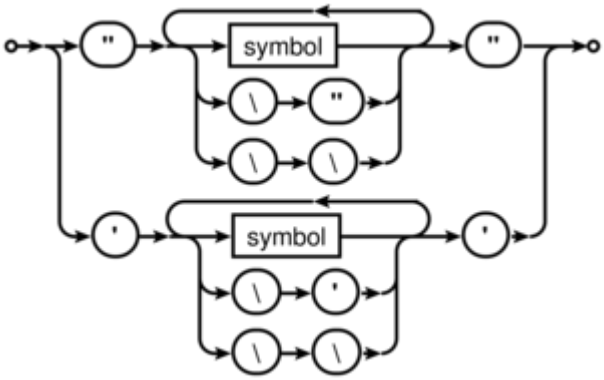
Functions can return this value when the result of function is not specified. For example, function `IF` returns its second argument if the first argument evaluates to a truthy value (see below on that). Otherwise, it returns third argument, but if it wasn't specified, it returns `undefined`.

Text

A text value consists of 0 or more Unicode symbols. Its literal representation consists of the value enclosed in single quotes (') or double quotes ("). Example: `"Major"` represents text value *Major*. Similarly, `'Major'` represents the same text value.

If the text value itself contains quotes, you'll need to insert a backslash (\) before them. Example: `"Charlie \"Bird\" Parker"` represents text value *Charlie "Bird" Parker*. Alternatively, you can use another kind of quotes to enclose the literal representation: `'Charlie "Bird" Parker'`.

If you need to use the backslash at the end of text value, you'll need to insert another backslash before it. Example: `"C:\Users\John\\"` represents text value *C:\Users\John*.



Numbers

Aside from representing some quantity, a number value can also represent points in time and duration of time. Then you can use Format settings in the Formula Column (see page 66) to properly display them as dates or durations.

There are two forms of literal representations of numbers:

- a whole number: `42`
- a fractional number: `0.239`

Note that only dot (.) can be used as a decimal separator. Comma (,) is used to delimit function arguments. Thus, `MAX(X, 0,618)` will be understood as the maximum of three quantities: *X*, 0, and 618.

Group separators are not supported, so `100 000` is not a literal representation of number `100000`.

> **Technical note**: internally, numbers are represented as decimal floating-point numbers with 16 digits of precision and half-even rounding. Most of the operations are carried out in this form, however, some of the more sophisticated functions, such as `SQRT`, might first convert the numbers into binary floating-point, calculate the result, and then convert it back into decimal floating-point.

Text to number conversion

Some functions expect their arguments to be number values. In case an argument is a text value, we try to interpret it as a number. This can be useful if the value comes from a variable that represents a text custom field, which contains numbers — e.g., imported from some external system.

If conversion is successful, that number is used as the value for that argument. If conversion is not successful, functions can either produce an error, ignore that argument, or substitute some default — it depends on the function; see Expr Function Reference (see page 82) for details.

The first step is to accommodate for variations in number formatting. Conversion supports these formatting symbols:

- decimal fraction separators:

comma	`,`
dot	`.`

- digit group separators:

comma	`,`
dot	`.`
apostrophe	`'`
space	

Conversion expects that the text contains 0 or 1 decimal mark, and 0 or more group separators of the same kind. If text contains any other formatting symbols, conversion fails. Decimal mark must come after all group separators, otherwise conversion fails.

If text contains only one formatting symbol, and it's a dot (`.`), it is always treated as decimal mark. If text contains only one formatting symbol, and it's a comma (`,`), then it is treated as decimal mark if comma is decimal separator mark in JIRA default language; otherwise, it is treated as group separator.

After determining decimal mark and group separator symbols, conversion removes all group separator symbols and replaces decimal mark with dot. Note that if text contains several whole numbers separated by spaces, conversion will think that its' one number, for example, `"10 11 12"` will become 101112. Similarly, `"10,11,12"` will become 101112.

The final step of conversion is to recognize the resulting text as either Expr's literal number representation or scientific or engineering notation. Examples:

```
0.239
-1.32e5
12e-3
```

Falsy and truthy values

A value is *falsy* if it is:

- `undefined`,
- number 0,
- empty text value (`""` or `''`), or a text value that contains only space characters.

All other values are *truthy.* By convention, when predefined functions or logical operators need to construct a truthy value, they use number `1`.

Variables and functions

Other kind of expressions are variables and function calls.

Identifiers

An identifier consists of letters (English only: a-z, A-Z), digits (0-9), dot (`.`) or underscore (_) characters. The first character must be a letter or an underscore.

Variables

Variables are represented by identifiers. Each variable is resolved to a value once during expression evaluation. If the variable cannot be resolved, its value is `undefined`.

Conceptually, you can think of variable as the cell of some column for the item, in the context of which the expression is evaluated. As such, it might or might not have a value, and that value can be either textual or numeric. Variables are defined in the Formula Column (see page 66) settings.

Function calls

A function consumes zero or more values, and can produce a value. A function call consists of a function name (an identifier), followed by its arguments enclosed in parentheses. An argument can be any expression. Different arguments are separated by commas (`,`) or semicolons (`;`) — for one function call, all separators must be the same.

Function call can evaluate only some or even none of the arguments, depending on the function. This is useful for functions that perform choices, such as `IF`: the argument that wasn't chosen is not evaluated, so the whole expression doesn't produce an error when it produces an error.

Single-argument operators

Expression with single-argument (or *unary*) operator has the following syntax: `<op> <expression>`.

`<expression>` can be any Expr language expression in parentheses. If it is a literal value representation, a variable, or a function call, parentheses are optional.

If `<expression>` evaluation produces error, the operator also produces error.

NOT

Instead of `NOT`, exclamation mark (`!`) can also be used.

The operator produces `0` if `<expression>` evaluates to a truthy value, and `1` otherwise.

+ -

The operator first attempts to convert the value of `<expression>` to number. If conversion succeeds, `+` produces this number, and `-` produces the negated number. If conversion fails, and the value of `<expression>` is falsy, produces `undefined`. Otherwise, produces error.

Logical and arithmetic operators

Two or more expressions can be combined using operators: `<expression1> <operator> <expression2>`. If any subexpression produces error, the operator produces the same error.

Logical operators

OR (||, |)

AND (&&, &)

`OR` examines each expression from left to right, and produces the value of the first expression that evaluates to a truthy value. If no expression evaluates to a truthy value, returns `undefined`. All expressions that come after the first that evaluates to a truthy value are not evaluated. This prevents from unnecessary computations, and protects from producing error if any of the subsequent expression produces an error.

`AND` works in the same way. The only difference is that it looking for the first *falsy* value.

Examples (assuming the default variable assignment):

- `assignee || "UNASSIGNED"` will produce either issue's assignee user key or text value `"UNDEFINED"` if the issue is unassigned.
- `!assignee && status = "OPEN"` will produce `1` if the issue is unassigned and in status `OPEN`, and `0` otherwise.

Comparison operators

All of these operators produce `0` or `1`. These operators can work only on two arguments. They start with evaluating both expressions. All comparison operators have the same precedence.

Equality: = (==) .

If both values are numbers, returns `1` if they are equal.

If both values are text, returns `1` if they are equal, ignoring differences in letter forms and leading and trailing whitespace (thus `" cote " = "côte"`).

If both values are undefined, returns `1`.

In all other cases returns `0`.

If one value is a number and the other value can be converted to a number, both values are treated as numbers. However, if both values are text, they will be treated as text, even if both can be converted to a number. You can use NUMBER (see page 102) function to force a value to be numeric.

- `3.4 = 3.40  1`
- `3.4 = "3.40"  1`
- `"3.4" = "3.40"  0`
- `NUMBER("3.4") = "3.40"  1`

Inequality: <> (!=)

Works in the same way as equality operator, but returns `0` where it returns `1` and vice versa.

Ordering

< (less than)

> (greater than)

<= (less than or equal)

>= (greater than or equal)

All operators work on numbers, producing the result of their comparison.

If either of the values is text, attempts to convert it to number. If conversion fails, operators behave as if the corresponding value was undefined.

If any value is undefined, strict operators (<, >) produce `0`. Non-strict (<=, >=) produce `0`, unless *both* values are undefined (because they are equal).

Arithmetic operators

Arithmetic operators are: addition (+), subtraction (-), multiplication (*) and division (/).

These operators convert their arguments to numbers. A non-empty non-number argument would produce an error. Falsy non-number values are treated as zero.

Examples:

- `"" + 1  1`
- `"foo" + 1  error`
- `"" * 1  0`
- `"foo" * 1  error`
- `"" - 1  -1`
- `1/0  error`

Precedence of operators

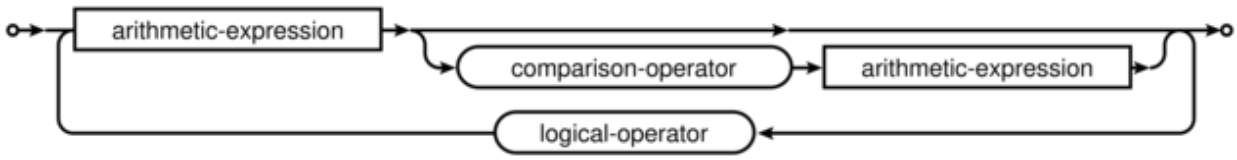
Precedence defines which operators evaluate first: if operator A has lesser precedence than B, then in expression `<expression1> A <expression2> B <expression3>` first B is evaluated, then A.

Single-argument operators are always evaluated first. Other operators in Expr language have the following precedence:

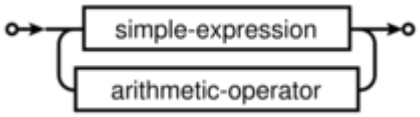
1 (lowest)	`OR`
2	`AND`
3	`= <> < > <= >=`
4	`+ -`
5 (highest)	`* /`

Railroad diagrams

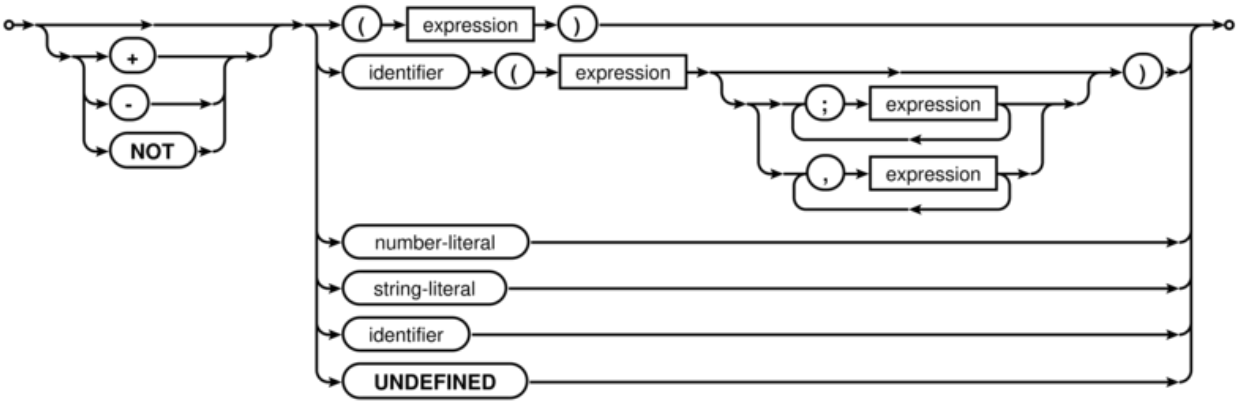
expression



arithmetic-expression



simple-expression



Expr Language Reference - v 4.2

DRAFT - update when releasing version 4.2

Expr language defines expressions, which are evaluated in the context of an item in some structure. This article describes the syntax of the language and the rules that govern the evaluation.

Conventions

- Similarity to Excel formula language was a design goal, so if you're unsure how Expr behaves, think Excel.
- The language is case-insensitive.
- Whitespace is not meaningful. It is only required to separate word operators and identifiers, in all other cases there can be arbitrary number of whitespace symbols.
- Currently language constructs support only English letters and a few punctuation symbols. However, values can contain any Unicode symbols.

Values

All expressions, when evaluated, produce either a value or an error. All values in Expr are either numbers, text, or a special value called *undefined*. The simplest expression thus is the literal representation of some fixed value. The forms of these literal representations are described below.

Undefined

Undefined value is represented by the word `undefined`.

Undefined value is used when the variable value is not specified. For example, variable `Assignee` has value undefined if the issue is unassigned.

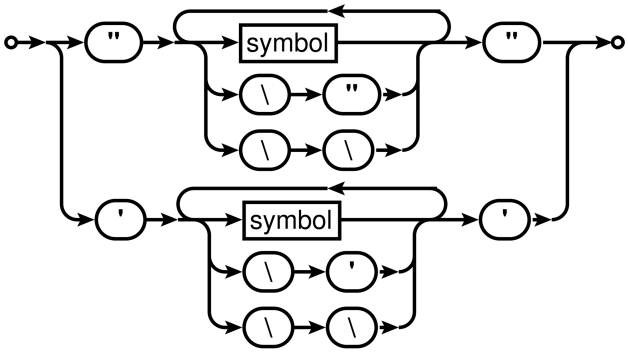
Functions can return this value when the result of function is not specified. For example, function `IF` returns its second argument if the first argument evaluates to a truthy value (see below on that). Otherwise, it returns third argument, but if it wasn't specified, it returns `undefined`.

Text

A text value consists of 0 or more Unicode symbols. Its literal representation consists of the value enclosed in single quotes (') or double quotes ("). Example: `"Major"` represents text value *Major*. Similarly, `'Major'` represents the same text value.

If the text value itself contains quotes, you'll need to insert a backslash (\) before them. Example: `"Charlie \"Bird\" Parker"` represents text value *Charlie "Bird" Parker*. Alternatively, you can use another kind of quotes to enclose the literal representation: `'Charlie "Bird" Parker'`.

If you need to use the backslash at the end of text value, you'll need to insert another backslash before it. Example: `"C:\Users\John\\"` represents text value *C:\Users\John*.



Numbers

Aside from representing some quantity, a number value can also represent points in time and duration of time. Then you can use Format settings in the Formula Column (see page 66) to properly display them as dates or durations.

There are two forms of literal representations of numbers:

- a whole number: `42`
- a fractional number: `0.239`

Note that only dot (.) can be used as a decimal separator. Comma (,) is used to delimit function arguments. Thus, `MAX(X, 0,618)` will be understood as the maximum of three quantities: *X*, 0, and 618.

Group separators are not supported, so `100 000` is not a literal representation of number 100000.

> **Technical note**: internally, numbers are represented as decimal floating-point numbers with 16 digits of precision and half-even rounding. Most of the operations are carried out in this form, however, some of the more sophisticated functions, such as SQRT, might first convert the numbers into binary floating-point, calculate the result, and then convert it back into decimal floating-point.

Text to number conversion

Some functions expect their arguments to be number values. In case an argument is a text value, we try to interpret it as a number. This can be useful if the value comes from a variable that represents a text custom field, which contains numbers — e.g., imported from some external system.

If conversion is successful, that number is used as the value for that argument. If conversion is not successful, functions can either produce an error, ignore that argument, or substitute some default — it depends on the function; see Expr Function Reference (see page 82) for details.

The first step is to accommodate for variations in number formatting. Conversion supports these formatting symbols:

- decimal fraction separators:

comma	,
dot	.

- digit group separators:

comma	,
dot	.
apostrophe	'
space	

Conversion expects that the text contains 0 or 1 decimal mark, and 0 or more group separators of the same kind. If text contains any other formatting symbols, conversion fails. Decimal mark must come after all group separators, otherwise conversion fails.

If text contains only one formatting symbol, and it's a dot (`.`), it is always treated as decimal mark. If text contains only one formatting symbol, and it's a comma (`,`), then it is treated as decimal mark if comma is decimal separator mark in JIRA default language; otherwise, it is treated as group separator.

After determining decimal mark and group separator symbols, conversion removes all group separator symbols and replaces decimal mark with dot. Note that if text contains several whole numbers separated by spaces, conversion will think that its' one number, for example, `"10 11 12"` will become 101112. Similarly, `"10,11,12"` will become 101112.

The final step of conversion is to recognize the resulting text as either Expr's literal number representation or scientific or engineering notation. Examples:

```
0.239
-1.32e5
12e-3
```

Falsy and truthy values

A value is *falsy* if it is:

- `undefined`,
- number 0,
- empty text value (`""` or `''`), or a text value that contains only space characters.

All other values are *truthy.* By convention, when predefined functions or logical operators need to construct a truthy value, they use number `1`.

Variables and functions

Other kind of expressions are variables and function calls.

Identifiers

An identifier consists of letters (English only: a-z, A-Z), digits (0-9), dot (`.`) or underscore (_) characters. The first character must be a letter or an underscore.

Variables

Variables are represented by identifiers. Each variable is resolved to a value once during expression evaluation. If the variable cannot be resolved, its value is `undefined`.

Conceptually, you can think of variable as the cell of some column for the item, in the context of which the expression is evaluated. As such, it might or might not have a value, and that value can be either textual or numeric.

Variables can be *free* (external) and *local*.

Free Variables

Free variables are defined in the Formula Column (see page 66) settings and associated with some attribute that later produces a value for each item the calculation is done for. There's no need to declare free variables – you just use it in the expression, and expression editor picks it up automatically and suggests you to assign an attribute to it.

Local Variables

Local variables are defined in the expression using "**with**" keyword:

```
with local_variable_name = <expression> :
<expression_with_local_variable>
```

Note the colon (":") that separates the expression assigned to the variable and the expression where variable is used.

A few facts about local variables:

- `<expression_with_local_variable>` may start with another local variable definition, so you can introduce many local variables. The expression assigned to a variable may refer local variables defined previously in the enclosing expressions.
- Local variables can "shadow" previously defined local and free variables with the same name. If you write `with priority = 10: <expression>`, then when calculating `<expression>`, the value of `priority` will be 10, even if there was a free variable attached to the issue's priority in the enclosing scope.
- The `with...` construct is itself an expression, so you can use it, enclosed in parentheses, anywhere in an expression. The name defined in this expression is not visible outside `with...` expression.

Function calls

A function consumes zero or more values, and can produce a value. A function call consists of a function name (an identifier), followed by its arguments enclosed in parentheses. An argument can be any expression. Different arguments are separated by commas (`,`) or semicolons (`;`) — for one function call, all separators must be the same.

Function call can evaluate only some or even none of the arguments, depending on the function. This is useful for functions that perform choices, such as `IF`: the argument that wasn't chosen is not evaluated, so the whole expression doesn't produce an error when it produces an error.

Single-argument operators

Expression with single-argument (or *unary*) operator has the following syntax: `<op> <expression>`.

`<expression>` can be any Expr language expression in parentheses. If it is a literal value representation, a variable, or a function call, parentheses are optional.

If `<expression>` evaluation produces error, the operator also produces error.

NOT

Instead of `NOT`, exclamation mark (`!`) can also be used.

The operator produces `0` if `<expression>` evaluates to a truthy value, and `1` otherwise.

+ -

The operator first attempts to convert the value of `<expression>` to number. If conversion succeeds, `+` produces this number, and `-` produces the negated number. If conversion fails, and the value of `<expression>` is falsy, produces `undefined`. Otherwise, produces error.

Logical and arithmetic operators

Two or more expressions can be combined using operators: `<expression1> <operator> <expression2>`. If any subexpression produces error, the operator produces the same error.

Logical operators

OR (||, |)

AND (&&, &)

`OR` examines each expression from left to right, and produces the value of the first expression that evaluates to a truthy value. If no expression evaluates to a truthy value, returns `undefined` . All expressions that come after the first that evaluates to a truthy value are not evaluated. This prevents from unnecessary computations, and protects from producing error if any of the subsequent expression produces an error.

`AND` works in the same way. The only difference is that it looking for the first *falsy* value.

Examples (assuming the default variable assignment):

- `assignee || "UNASSIGNED"` will produce either issue's assignee user key or text value `"UNDEFINED"` if the issue is unassigned.
- `!assignee && status = "OPEN"` will produce `1` if the issue is unassigned and in status `OPEN`, and `0` otherwise.

Comparison operators

All of these operators produce `0` or `1`. These operators can work only on two arguments. They start with evaluating both expressions. All comparison operators have the same precedence.

Equality: = (==) .

If both values are numbers, returns `1` if they are equal.

If both values are text, returns `1` if they are equal, ignoring differences in letter forms and leading and trailing whitespace (thus `" cote " = "côte"`).

If both values are undefined, returns `1`.

In all other cases returns `0`.

If one value is a number and the other value can be converted to a number, both values are treated as numbers. However, if both values are text, they will be treated as text, even if both can be converted to a number. You can use NUMBER (see page 102) function to force a value to be numeric.

- `3.4 = 3.40  1`
- `3.4 = "3.40"  1`
- `"3.4" = "3.40"  0`
- `NUMBER("3.4") = "3.40"  1`

Inequality: <> (!=)

Works in the same way as equality operator, but returns `0` where it returns `1` and vice versa.

Ordering

< (less than)

> (greater than)

<= (less than or equal)

>= (greater than or equal)

All operators work on numbers, producing the result of their comparison.

If either of the values is text, attempts to convert it to number. If conversion fails, operators behave as if the corresponding value was undefined.

If any value is undefined, strict operators (<, >) produce `0.` Non-strict (<=, >=) produce `0`, unless *both* values are undefined (because they are equal).

Arithmetic operators

Arithmetic operators are: addition (+), subtraction (-), multiplication (*) and division (/).

These operators convert their arguments to numbers. A non-empty non-number argument would produce an error. Falsy non-number values are treated as zero.

Examples:

- `"" + 1  1`
- `"foo" + 1  error`
- `"" * 1  0`
- `"foo" * 1  error`
- `"" - 1  -1`
- `1/0  error`

Precedence of operators

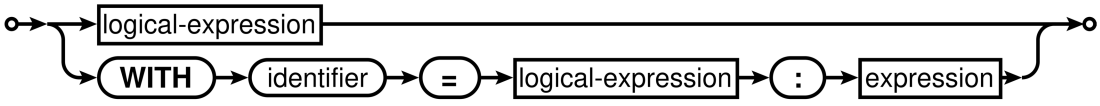
Precedence defines which operators evaluate first: if operator A has lesser precedence than B, then in expression `<expression1> A <expression2> B <expression3>` first B is evaluated, then A.

Single-argument operators are always evaluated first. Other operators in Expr language have the following precedence:

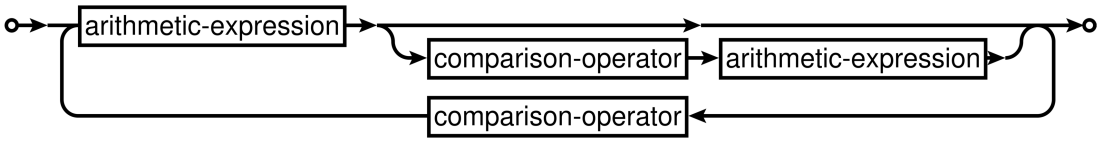
1 (lowest)	`OR`
2	`AND`
3	`= <> < > <= >=`
4	`+ -`
5 (highest)	`* /`

Railroad diagrams

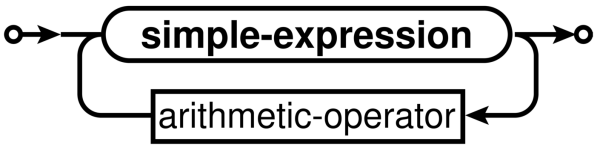
expression



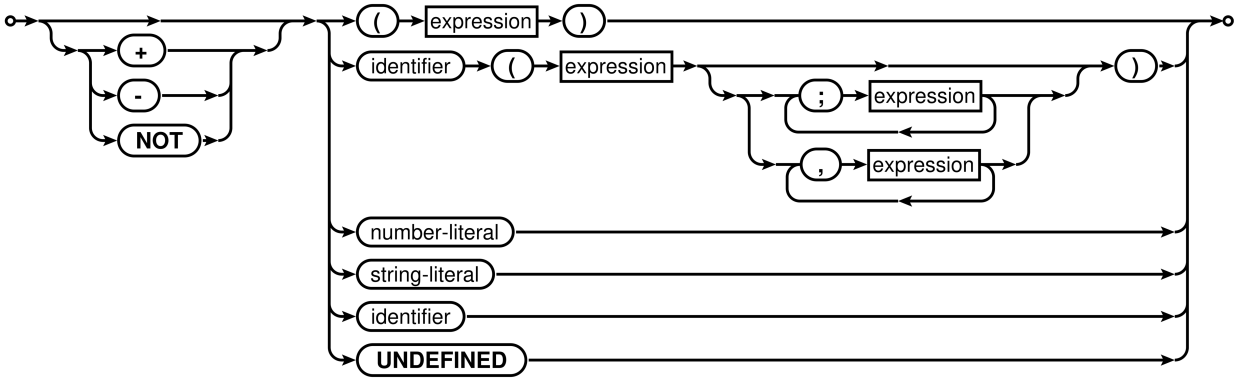
logical-expression



arithmetic-expression



simple-expression



Expr Pattern Matching

Expr language provides you with a couple of functions that make it easier to check a text value against a certain pattern. Functions MATCH (see page 88), CASE (see page 82), REPLACE (see page 89) and SEARCH (see page 90) use pattern matching for your convenience.

Matching involves a value and a pattern. There are three types of patterns – the type that you use defines how matching is done.

Exact Matching

The simplest pattern type is just a text value that you expect:

- `MATCH(value, "Apples")`

This match will happen if `value` is, in fact, the text that is used as the pattern.

Although it's called "exact matching", there are some additional rules that make the matching easier. The rules are:

- All leading and trailing whitespace characters are removed from the value.
- Text comparison is case-insensitive, which means APPLES will match `Apples`.
- The value (without leading and trailing spaces) must match the whole pattern.

Exact matching is used by default, if the pattern is not recognized as requiring Wildcard or Regular Expression matching.

Wildcard Matching

Wildcard patterns let you use symbol "`*`" to specify any number of any characters.

- `MATCH(value, "App*")`

You can use multiple asterisks to build your pattern.

The same rules as for the exact matching apply:

- All leading and trailing whitespace characters are removed from the value.
- Text comparison is case-insensitive, which means APPLES will match `App*`.

- The value (without leading and trailing spaces) must match the whole pattern.

Exact matching is used when the pattern is not recognized to be a Regular Expression Pattern but contains at least one asterisk.

Regular Expression Matching

This type of matching lets you use powerful regular expressions to specify exactly what you need to match with.

- `MATCH(value, "/^Ap+.*s$/")`

Structure uses regular expressions available with Java. For a full documentation about the regular expression language, see Java documentation for Pattern.

The regular expression matching is different from other types of matching. The following rules apply:

- Leading and trailing whitespace characters are **not** removed.
- Text comparison is case-insensitive, like with the other types of matching.
- The value does not have to fully match the pattern – it is sufficient that at least one occurrence of the pattern is found in the value. To make your pattern match the whole text, use "^" and "$" characters in the pattern.

Regular expression matching is turned on if the first and the last characters of the pattern are "`/`". (These characters are removed, they are not a part of the pattern.)

Expr Error Codes

Evaluating Expr expression may produce errors. Normally these errors are shown to the user with a human-readable message. However, you might need to check for a specific error using ISERR (see page 84) function.

Error Code	Name	Displayed As	Description
1	Parse Error	???	The expression is invalid. To fix, review and edit the expression.
2	Unknown Function	FUNC?	The expression contains a function that is not available or does not exist. To fix, review used functions in the expression, see if there are any typos.
3	Bad Number of Arguments	ARGS?	

Error Code	Name	Displayed As	Description
			A function is used with an incorrect number of arguments. To fix, review the expression and see if all functions are called with a correct number of arguments.
4	Arithmetic Error	DIV/0	An arithmetic error was encountered. Most often it is division by zero, but it may also be something else, like passing a non-integer value to a function that expects only an integer. To fix, first, find out what the problem is (you can try separately calculating parts of the formula). To avoid division by zero, use IF (see page 83) function.
5	Variable Error	VAR!	An attribute that a variable was bound to produced an error. To fix, review the attributes bound to the expression variables and see why they could have produced an error for the row that shows this error.
6	Function Execution Error	FUNC!	A function suffered an internal error. Refer to logs for details.
7	Value Conversion Error	VALUE?	A value could not be converted to the desired format. Check the formula, most likely a non-convertible text value is submitted to a function as a number.
8	Malformed Regex	REGEX?	A text with invalid regular expression was passed to a matching function. Check the regular expressions that you use. See Expr Pattern Matching (see page 117) for details.
9	Internal Error	ERROR!	There's an internal problem with JIRA or Structure. Refer to the logs for details.
10	Invalid Value	VALUE!	

Error Code	Name	Displayed As	Description
			An invalid value was passed as a function argument. For example, using an unknown unit of time would produce this error. Check arguments for the functions that expect specific values.

Expr Language - v 4.2

DRAFT **- to replace parent page when version with local variables is released**

Expr Language (pronounced like "expert" without "t") is a simple language that lets you specify an "expression", or a formula, which will be calculated for an issue or another item. When you use is in a Formula Column (see page 66), the expression is calculated for each visible row in the displayed structure or query result.

You can see examples of formulas by adding predefined columns in Structure (from the **Calculated** section), and then opening column options (see page 43) panel. The language itself and its grammar is quite obvious and is similar to arithmetic expressions with a number of functions.

> If you're familiar with writing formulas in **Microsoft Excel**, you'll recognize a few things in Expr. In particular, if you know some functions that Excel provides, they have a good chance to be supported by Structure as well.

Language Components

An expression may contains one or more of the following:

- Variables, which are bound to specific values when calculating expression for a specific item.
- Functions, which may take some arguments, and which produce the result at the moment of calculation.
- Numbers and text strings.
- Arithmetic, logical operations and parentheses.
- Local variables, which hold intermediate values.

Variables

Variables are user-defined names, containing letters (English only), numbers, dot (".") or underscore ("_") characters. The first character should be a letter or an underscore.

Examples:

- `Priority`
- `remaining_estimate`
- `abc11`
- `sprint.name`

> Variable names are case-insensitive. `Priority`, `priority` and `pRiOrItY` refer to the same variable.

When writing an expression, you'd usually associate a name with some value that an issue or another item has. After the expression is written, Formula Column (see page 66) lets you associate the variables used with specific attributes.

A few things to note about variables:

- You can use any names, from a simplistic "x" to a "VeryComplicatedCustomFieldName".
- But, if Formula Column recognizes the variable name to be similar to a field name, it may automatically assign an attribute to the variable. For example, "Priority" variable will be automatically mapped to the similarly named field.
- But, it is possible (although very unreasonable!) to edit the association and assign a variable with a well-known name to something else. ⚠ Please keep that in mind if you need to troubleshoot a formula and double-check the variables.

Functions

A function calculates some value based on its arguments and, sometimes, some external aspect. A function is written as the function name, followed by parentheses, which might contain arguments.

Examples:

- `SUM(-original_estimate; remaining_estimate; time_spent)`
- `CASE(priority, 'High*', 5, 1)`
- `TODAY()`

There are a number of standard functions available with Structure 4.0 – see Expr Function Reference (see page 82) for details.

A function may take zero, one or more arguments. Some functions take variable number of arguments. Each argument can be another Expr expression and include calls to other functions.

Function arguments may be separated by comma (,) or semicolon (;). But in every function call you need to use either all commas or all semicolons.

Function names are case-insensitive, like the variables. You can write `TODAY()` or `Today()`.

Numbers and Text Strings

Numbers

You can use numbers in your formula. The numbers are always written as a sequence of digits with optionally a dot (".") and a fractional part. Locale-specific, percents, currency or scientific formats are not supported.

Recognized as a number	Not recognized as a number
`0`	`0,0`
`1234567890123456`	`1 100 025`
`11.25`	`1.234e+04`
`.111`	`($100)`

You can write a number that is written with a locale-specific decimal and thousands separator as a text value and it will be automatically converted to a number if needed. For example:

- `"1 122,25" * 2  2244.5`

Text Strings

Text strings are a sequence of characters enclosed either in single (') or double quotes ("). Examples:

- `'a text in single quotes may contain " (a double quote)'`
- `"a text in double quotes may contain ' (a single quote)"`
- `""`

Everything within a text string is retained verbatim to participate in the expression evaluation, except for the following:

- A sequence of two backslashes (`\\`) is converted to a single backslash (`\`).
- A sequence of a backslash and a single quote (`\'`) is converted to a single quote character (`'`) in the text values enclosed in single quotes.
- A sequence of a backslash and a double quote (`\"`) is converted to a double quote character (") in the text values enclosed in double quotes.

Operations

Expr provides basic arithmetic operations, comparisons and logical operations.

The operations follow the general precedence rules for arithmetics, so `A + B * C` is calculated correctly. Comparison operations are done after the arithmetic operations and logical operations are done after comparisons. For detailed specification, see Expr Language Reference (see page 102).

Operations	Comments
`+ - * /`	Convert the value into a number.
`= !=`	Equality and non-equality: if either part of the comparison is a number, the other part is also converted into a number. If both values are strings, then string comparison is used. String comparison ignores leading and trailing whitespace and is case-insensitive (according to JIRA's system locale).
`< <= > >=`	Numerical comparisons – both values are converted to numbers.
`AND`, `OR`, `NOT`	Logical operations.
`( )`	Parentheses can be used to group the results of operations prior to passing to other operations.

Local Variables

Local variables allow you to simplify expressions and make them more readable by giving a name to some intermediate value used in calculation.

Local variables are defined right in the expression's text:

```
with PRI = CASE(priority; 'High*'; 5; 1):
with COST = (remaining_estimate + time_spent) * rate:
100 * PRI / COST
```

In this example, we defined two variables, `PRI` and `COST`, and used them to calculate the final value (`100 * PRI / COST`). Without local variables, we could have written the same expression as `100 * CASE(priority, 'High*', 5, 1) / ((remaining_estimate + time_spent) * rate)`, which is much harder to read.

Note the colon symbol ("`:`") that separates variable definition from the following expression.

Local variables are not associated with item attributes and Formula column does not offer to define the value for local variables. When defining a local variable, you can use other local variables defined previously in the same expression.

You can always copy and paste the whole expression, including local variables, into another expression. Just make sure to surround the whole thing in parentheses. For example, the sample above can be reused like that:

```
10 * (
   with PRI = CASE(priority; 'High*'; 5; 1):
   with COST = (remaining_estimate + time_spent) * rate:
   100 * PRI / COST
)
```

Or like that:

```
with PRI_BY_COST = (
   with PRI = CASE(priority; 'High*'; 5; 1):
   with COST = (remaining_estimate + time_spent) * rate:
   100 * PRI / COST
):
with URGENCY_FACTOR = IF(duedate; 90 - MIN(90; MAX(0; DAYS_BETWEEN
(NOW(); duedate))); 0) / 7:
PRI_BY_COST * URGENCY_FACTOR
```

See Also

- Expr Function Reference (see page 82)
- Expr Language Reference (see page 102)

Notes Column

Notes column allows you to add arbitrary text to items in a structure, without having to create custom fields in JIRA.

The typical use case for Notes column is storing some additional information about the item's status and use it in a report.

Portfolio Overview with Automation

Key	TP	Summary	Status	Notes
TP-124		Site preparations	OPEN	
SP-4		Flatten building site - Our theme park ne	OPEN	Bob's team
SP-2		Bulldoze French Alps - Someone p	OPEN	
SP-3		Remove waste rock and soil - After	IN PROGRESS	still need a confirmation
SP-6		Build access road - We need an eig	IN PROGRESS	the attachments need improvement
SP-11		Build access and protection	OPEN	
SP-9		Build a transparent dome over the th	IN PROGRESS	is it necessary?
SP-15		Install entrance checkpoints	OPEN	
SP-8		Dig escape tunnels for staff - We n	OPEN	should be closed. see the report
SP-6		Build access road - We need an eight-la	IN PROGRESS	the attachments need improvement
SP-10		Move stuff to another place	OPEN	consider teleportation
TP-31		Marketing + PR activities	OPEN	
MKT-4		'Theme Park is Safe' campaign - We nee	TO DO	check who has the same slogan
TP-8		Rides + attractions	OPEN	pitch new ideas to Mark please

The values in the Notes columns are **per-structure, per-item**. This means that:

- Text entered as Notes for some issue in one structure will not be seen for that issue in another structure. You may have different notes for the same issue in different structures.
- If an item occurs several times in a structure, they will have the same value in the Notes column, similar to the issue's fields.

Permissions

The user might have permissions to edit notes even if he or she does not have permissions to edit the issue.

The data stored in the Notes column is considered to be a property of the selected structure. That has the following effect on the permissions.

Who can view notes?

To be able to see the notes, the user needs to have:

- View access to the structure that stores the notes.
- View access to the item (issue, project, etc.)

Who can edit notes?

To be able to edit the notes, the user needs to have:

- Edit access to the structure that stores the notes.
- View access to the item (issue, project, etc.)

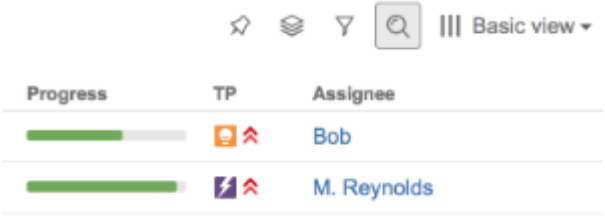
> Note that this allows you to create your own structure and leave notes for some issues that you can't edit.

1.4.6 Searching and Filtering

The Search feature provides several important functions:

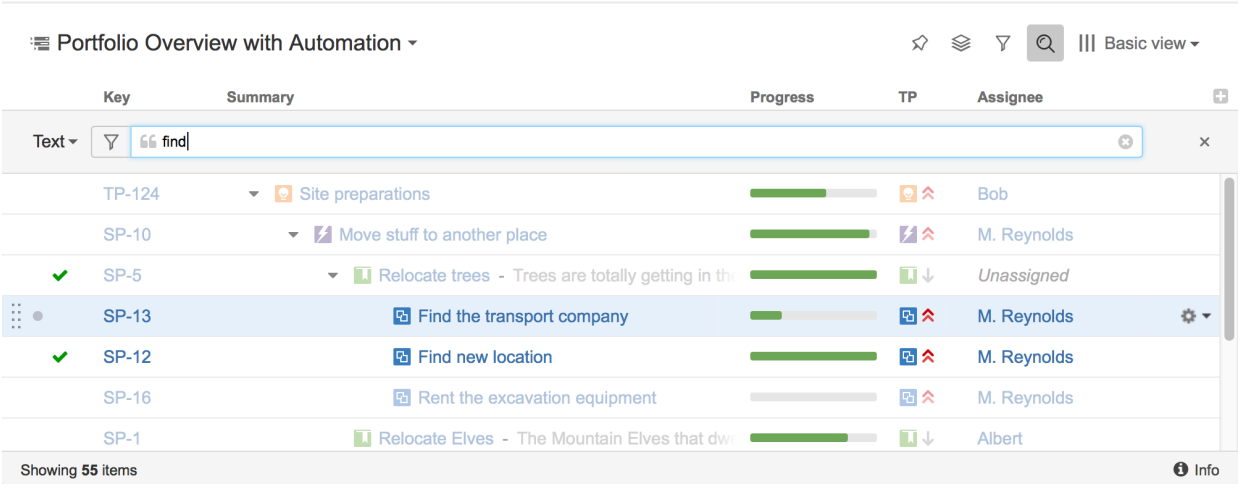
- Find and highlight issues in your structure.
- Filter (see page 129) your structure so that it only displays specific issues.

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



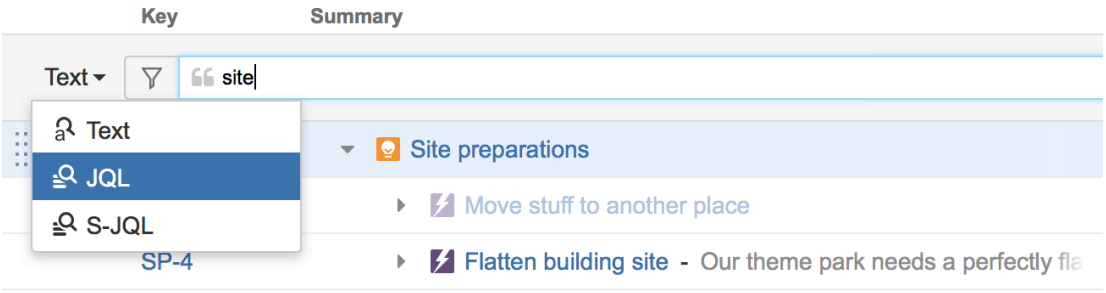
The Search panel will appear below the toolbar. By default, the Search looks for the entered text in issues' summary or items' names.

The search starts once you start entering the query, refining results as you keep typing. The non-matching items are greyed-out in order to highlight the matching items.



Once you've found the items you need, you can quickly move focus between the matching items by pressing **Ctrl+Alt+]** and **Ctrl+Alt+[**.

Search function allows you to search for issues inside the currently selected structure in Simple, JQL, and S-JQL (see page 127) modes and use text search for all other item types. To switch between the modes, click the name of the currently selected mode and select the one you need from the menu.

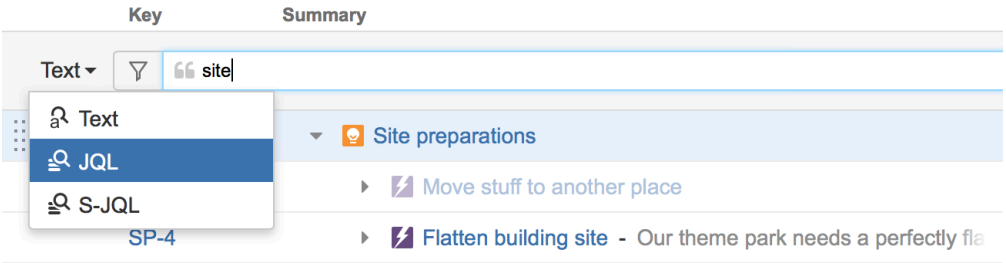


If data changes on the server, search results are automatically refreshed for the structure. So issues can be hidden and shown in the structure in realtime.

You can turn on searching on by pressing **Ctrl+Alt+/**. If you press it again, you will switch to the next search mode. To close the search panel and cancel the search, press **Escape** or click again the search button in the toolbar.

Simple, JQL, and S-JQL Search

In the Search Area, you can specify a simple text condition (see page 127), a JQL condition (see page 128), or a Structure query (see page 129). To switch between these search modes, click the name of the currently selected mode and select the one you need from the menu or press **Ctrl+Alt+/**.



Simple Search

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

Condition Type	Example	How it works
Simple text	*structural hierarchy*	Look for items that have **all** mentioned words in the **Summary** field. Each word in the search sentence must be present in the summary or name, 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 or name (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 items". Only issues from the projects enabled for Structure are found.

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, there is a green tick icon shown in the input box. When the JQL is incorrect or not complete, the red icon with the exclamation mark is shown.

More information on JQL is available in the JIRA documentation.

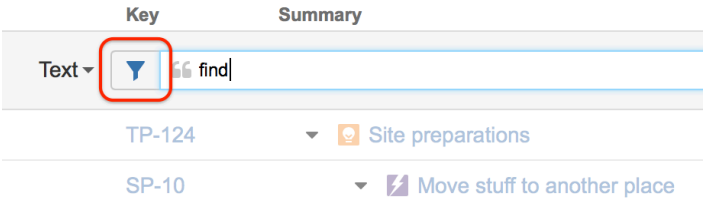
S-JQL Search

In the S-JQL mode, the search condition defines a Structure query (see page 258). 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 (see page 252) for more information.

As with the JQL mode, the there is an indicator showing whether the query is correct or not.

Filtering

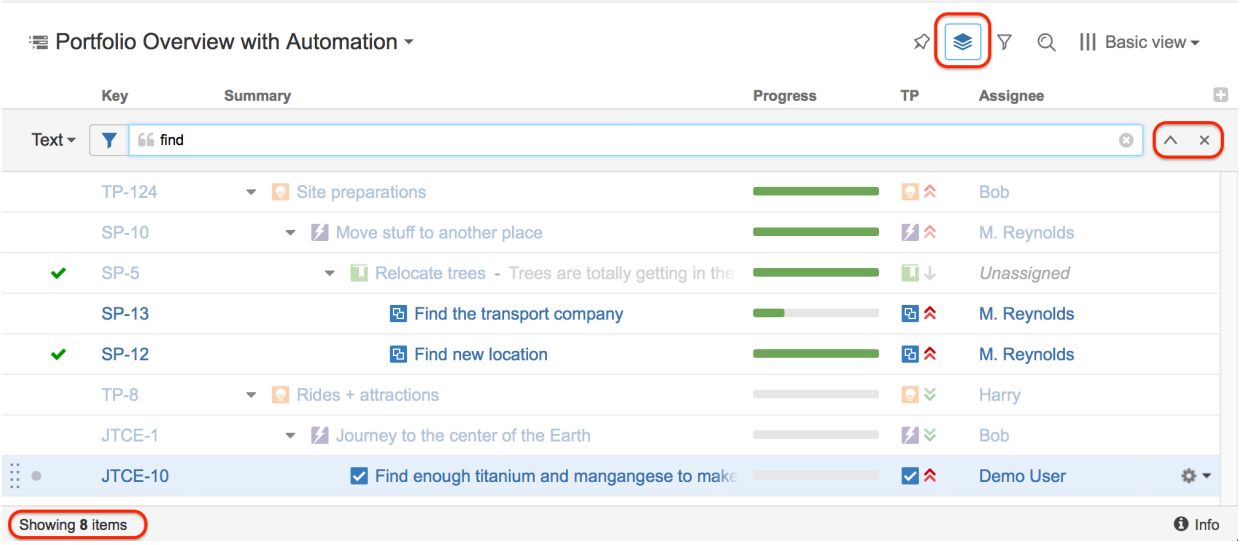
If you wish to see only the items that match the criteria specified in the search field, click the **Filter** button on the left from the input box.



You can use keyboard shortcut **Ctrl+Alt+;** or **;** to turn filtering on and off.

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

In the status bar at the bottom you can see the updated items count.



Filtering is just one of many ways to adjust the scope of the items you are seeing. Such adjustments in Structure are called Transformations (see page 133) and you can add them clicking the Transformations button.

As you add the filtering - the button is highlighted, showing you that a transformation has been applied.

You can either remove the filtering completely by clicking the close button on the right of the search field, or you can hide the search panel by clicking the arrow button next to it.

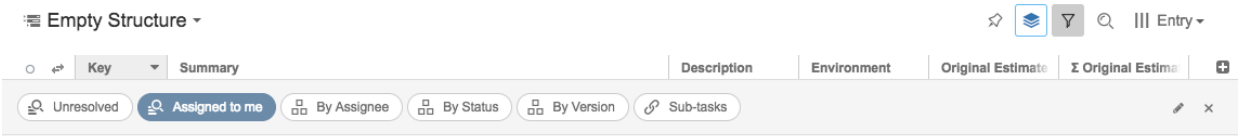
Filtering mode remains even if you navigate to another page.

Default Quick Transformations

There are six predefined saved filters:

- Unresolved (works as a shorthand for filtering using JQL: *Resolution is EMPTY)*
- Assigned to me (JQL: *Assignee = currentUser())*
- By Assignee (items are grouped by Assignee)
- By Status (items are grouped by Status)
- By Version (items are grouped by FixVersion)
- Sub-tasks (Sub-tasks are added underneath tasks)

To apply them, click the **Filter** button next to the **Transformations** button in the structure panel toolbar and select the filter you need.



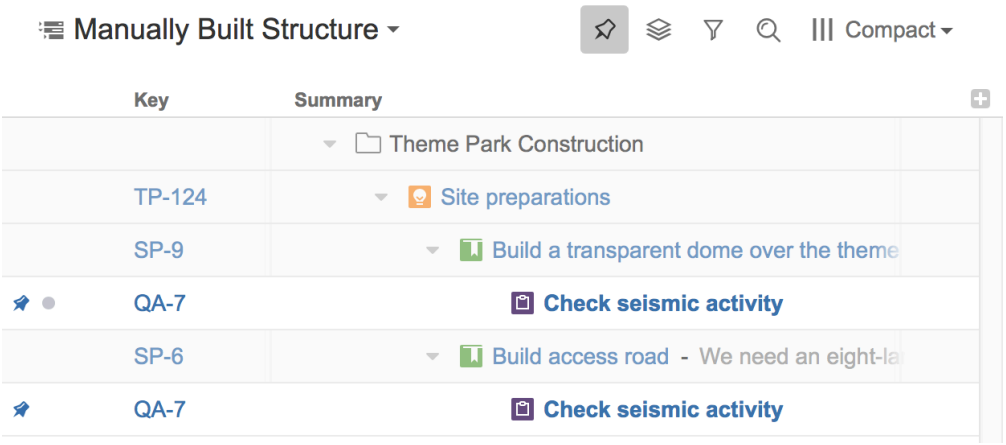
You can apply these filters and then use search and additional filtering at the same time.

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

Pinned Item Mode

You can view only a part of a structure that is related to a specific item, by pinning that item with a **Pin** icon on the structure panel toolbar. Structure Panel on the Issue Page (see page 17) automatically pins the issue being displayed, so you only see the relevant part of the structure.

In an item is present in several places in the structure, both instances of this item are shown.



What is Displayed in Pinned Item mode

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

- The pinned item itself
- All parent items of the pinned item, up to the top-level item
- All sub-items of the pinned item, down to the deepest level

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

The items that are not displayed when an item 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 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.

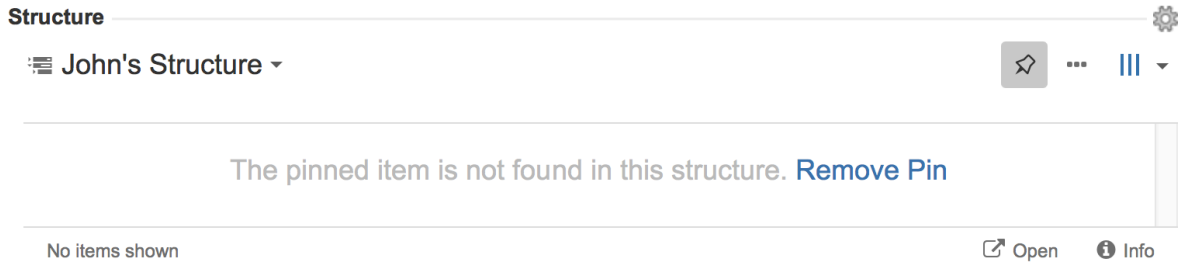
Limitations Imposed by the Pinned Item Mode

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 item, but you can't add items to the pinned item's parents or move pinned item somewhere else.

Even though you can't move parent items 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 item is missing from structure, the structure widget will not be able to display any data and will offer to remove Pin:



If you are seeing this on the Issue Page, one you remove the pin you'll be able to add the issue you are viewing to the structure you have opened in the widget (**Place** issue).

1.4.7 Transformations

As you work with a structure, sometimes you may want to reorganize the issues you are seeing, so that you can focus on the most important ones. The simplest example of this is Filtering (see page 129) and Sorting. For example, you may want to see only issues assigned to you sorted by progress.

These are the two examples of the **Transformations** functionality. With transformations you can locally adjust the structure without changing it for everyone else. All other users will see this structure without these transformations applied.

Available Transformations

Transformations use the same type of functions as generators (see page 178). The main difference is that the transformations can only be applied to the whole structure (while generators can be inserted under some folder or under a manually added issue) and you cannot use Insert generators as you already have a set of issues to work with. You can use the following transformations:

- Filter
- Sort
- Group
- Extend

For more details on how they work, please check the documentation on Generators (see page 178).

Working with Transformations

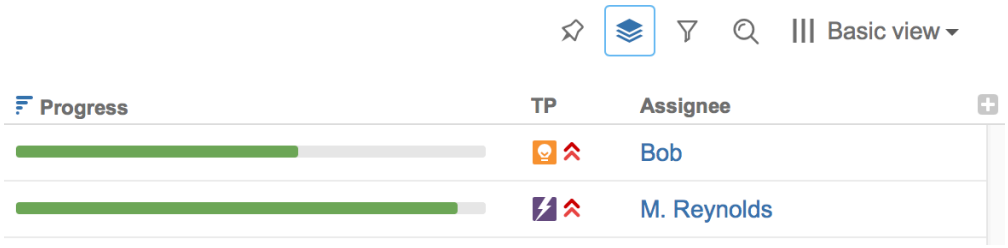
Using Transformations

Sorting and Filtering

Sorting and Filtering are two kind of transformations you can apply really quick.

To **sort** your items, simply click the header of the column to sort the structure by this column in ascending order. On every level, the items will be sorted accordingly. To sort in descending order - click the column header again.

As you add the sorting, the Transformations button is highlighted, showing there are transformations applied:

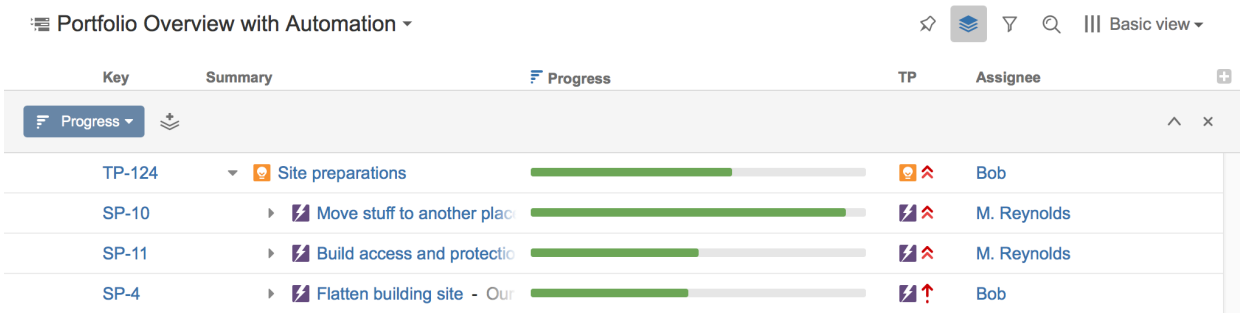


To remove the sorting - click the Summary column header.

To apply a **filter**, you can run a search (see page 126) and then filter (see page 129) out non-matching items, or you can access the Quick Filters panel through the **Filter** button on the panel toolbar and select any of them.

Transformations Panel

All transformations can be added and modified on the transformations panel. To access it, click the **Transformations** button in the panel toolbar. The panel with the currently applied transformations will appear.

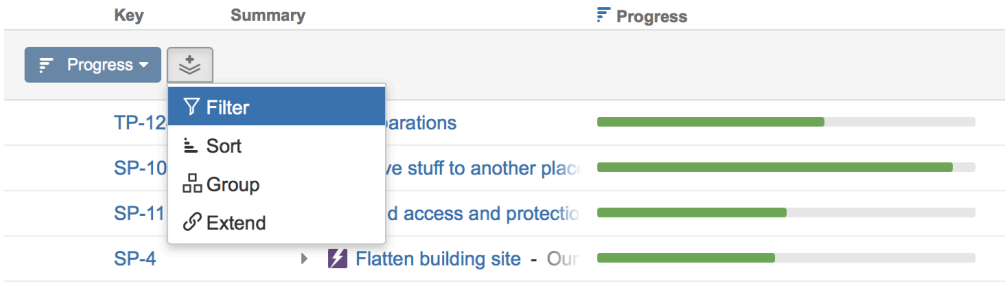


Once you add the transformations, they'll be preserved as you switch between structures if you are using the Structure selection in the widget (see page 33), not the Structure menu. This allows you to quickly check several structures focusing on the issues you need.

There are several things you can do with transformations.

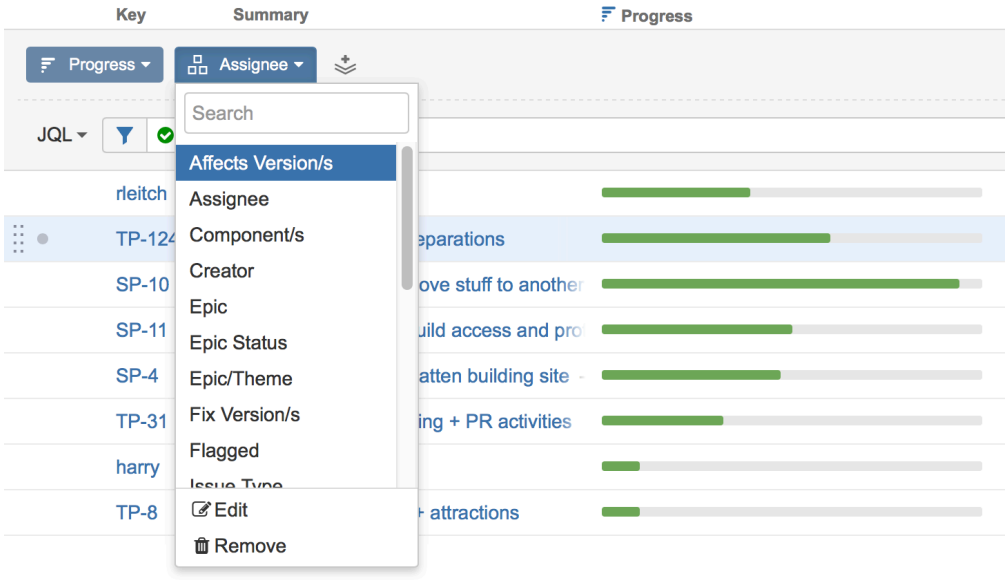
Add

To add a new transformations, click the Add button in the Transformations toolbar and select one of the available transformations.



Edit

To edit an existing transformation, click the transformation name - for Grouping and Sorting you can quickly select a column by which you want to sort or group, for Filtering and Extending, select the type of filter or extender. You can also click the Edit button, to open the transformation Edit Dialog with all the options. The options you see are the same as generators have (see page 188).



Remove a Transformation

To remove a transformation, click the transformation name and select Remove from the menu that appears.

Hide Transformations Panel

After you've configured your transformations, you can hide the panel, so it doesn't take up the screen space. Click the up arrow button on the right side of the transformations panel to hide it.

Remove All Transformations

To remove all transformations, click the cross button on the right side of the transformations panel.

Save Transformations

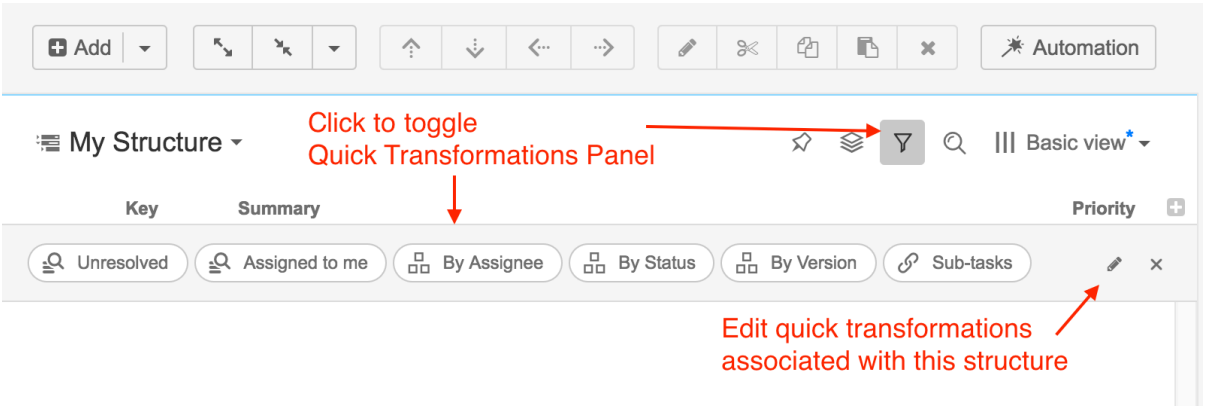
Users who have Control permission on a structure can save transformations as Quick Transformations (see page 135), associated with that structure.

Quick Transformations

Quick Transformations feature lets you set up some transformations to be readily available for a structure, so they can be turned on and off with a single mouse click. (They are sometimes called "Quick Filters", although there could be other transformations besides filters.)

Quick Transformations Panel

You can start using quick transformation by showing Quick Transformation panel.



Quick transformation panel shows transformations that are associated with the current structure. If quick transformations were not customized for the displayed structure, default quick transformations are shown. Default transformations are also shown when panel contains a query result or another non-structure content. See Default Transformations (see page 139) for description of each default transformation.

To activate a quick transformation

- Turn on Quick Transformation Panel
- Click on the desired transformations

Structure will remember the selected transformations, so next time you open that structure, the transformations will be already applied.

To deactivate a quick transformation

- Click on the transformation you'd like to stop applying, OR
- Toggle Quick Transformations Panel or click "x" button to close Quick Transformations Panel.

When you close Quick Transformations Panel, all quick transformations are deactivated.

You can use keyboard shortcuts to toggle quick transformations, based on their position in the transformation list. The shortcut is **Q** and then the number (**1—9**), typed in quick succession.

The transformations are applied in the order they were turned on. If you need to change the order in which quick transformations are applied, you can turn off the transformation that should come last, and then turn it back on.

Alternatively, you can use Transformations panel to reorder currently active transformations with drag and drop.

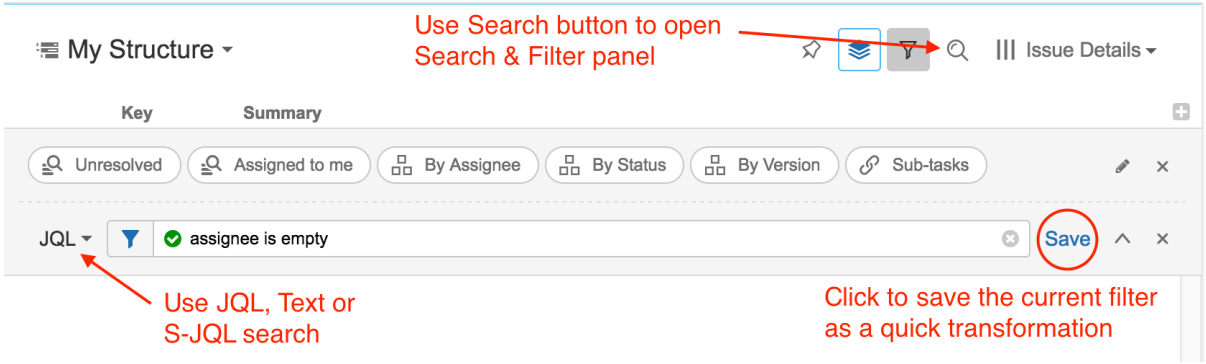
Defining Quick Transformations

Quick transformations can be customized for each structure by anyone who has **Control** access to the structure. You either use a filter (see page 126) or use a transformation (see page 133) and then save it as a quick transformation.

> If you don't see a way to add a quick transformation, probably you don't have the access and you need to ask the structure's owner to add the quick transformation that you need.

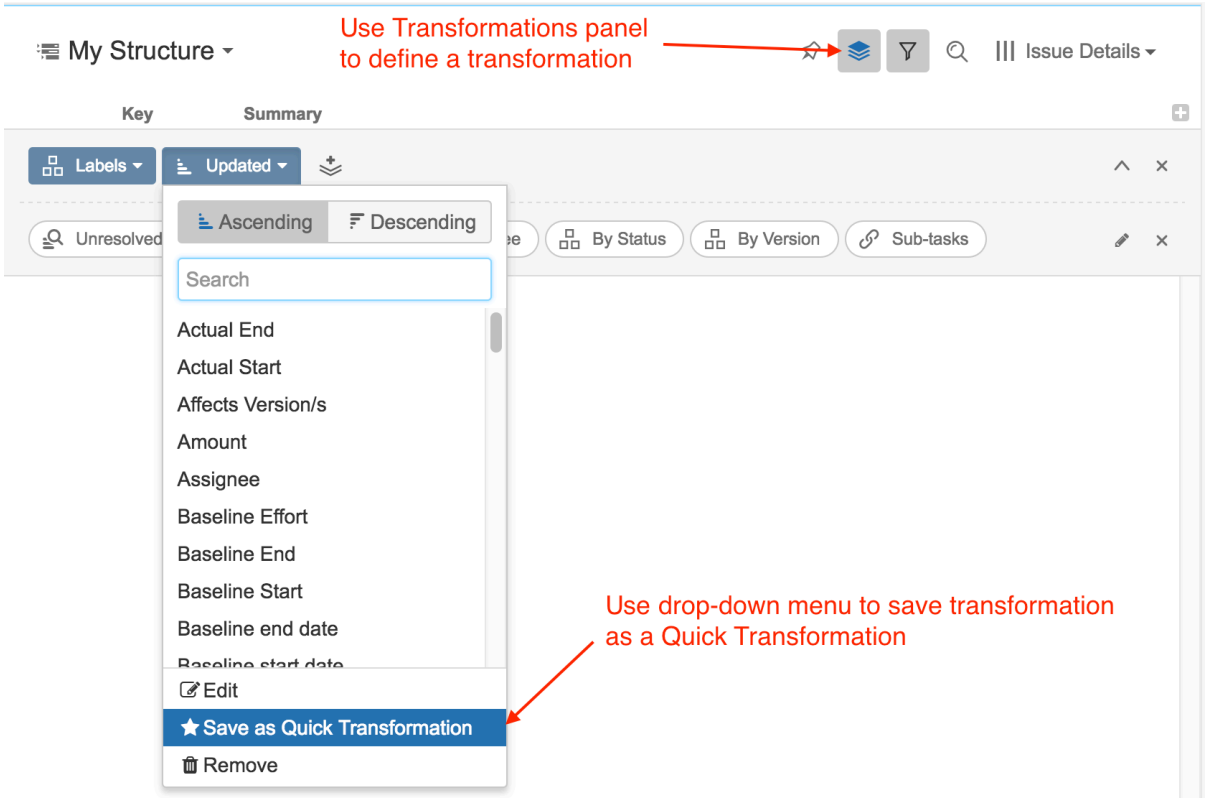
Adding a Quick Filter

To add a Filter transformation, click "Save" on the filtering panel.



Adding an Extender, Grouper or Sorter Transformation

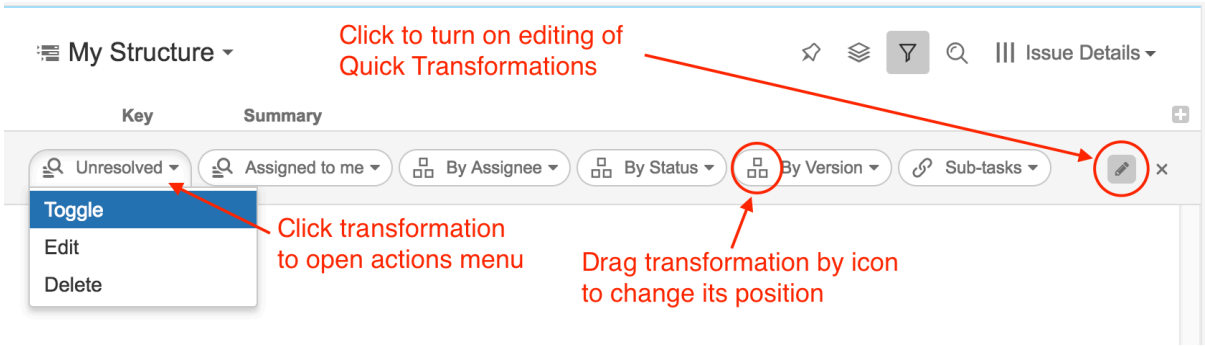
For all other transformations, use the Transformations panel.



Modifying Quick Transformations

If you have **Control** access to the structure, you can change the associated quick transformations, remove unused transformations, or change the order in which transformations appear.

To edit Quick Transformations, click the Pencil icon on the quick transformations panel.



You can remove the default quick transformations for your structure if they are not needed. It will not affect other structures.

Default Transformations

Default quick transformations are shown whenever a structure does not have customized quick transformation or when a panel shows query result, clipboard or other non-structure content.

The following transformations are available:

Transformation	Effect of applying this transformation
Unresolved	Only issues with empty Resolution field are shown.
Assigned to me	Only issues assigned to the current user are shown.
By Assignee	All top-level issues are grouped by Assignee
By Status	All top-level issues are grouped by Status
By Version	All top-level issues are grouped by Version
Sub-tasks	Sub-tasks are added to the structure under their parent tasks

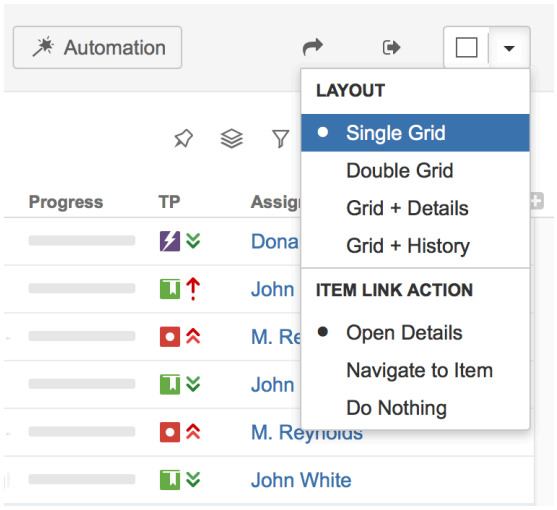
1.4.8 Two-Panel Mode

When working with Structure on the Structure Board page, you can switch to the two-panel mode and thus take the full advantage of the screen space.

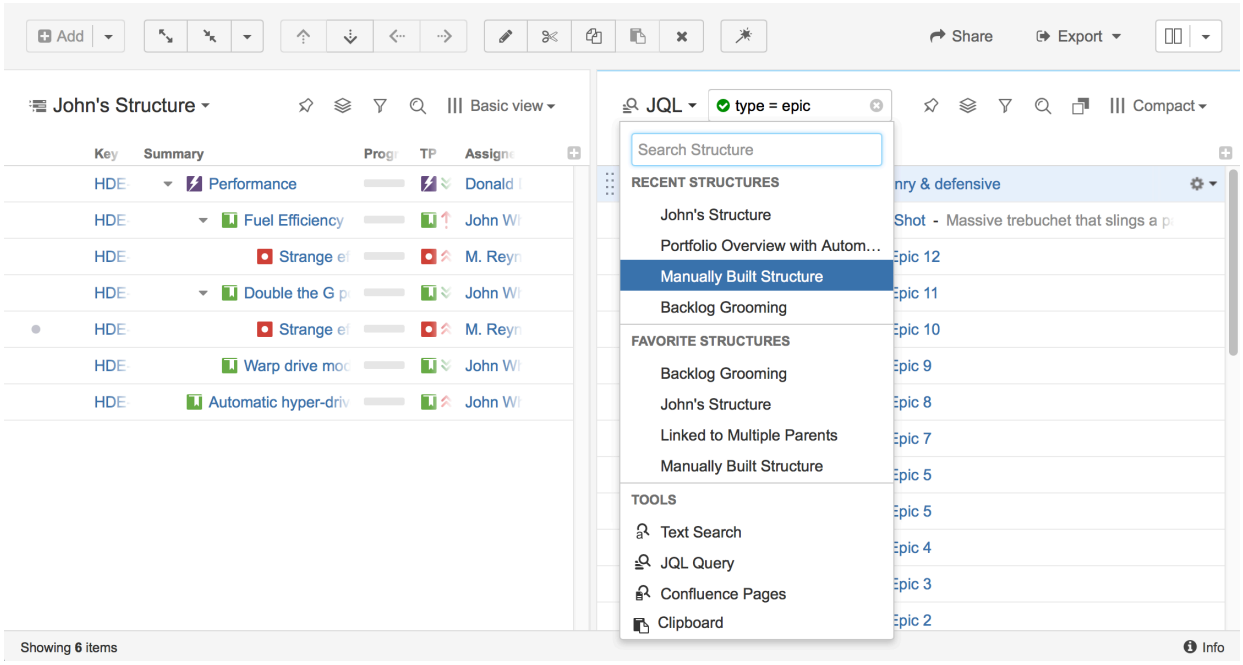
The left panel always displays the structure widget or search and on the right panel you can open one of the following:

- Another structure widget. In the widget you can open another structure and work with two structures side by side or you can use it to run text/JQL search or show clipboard.
- Issue details (see page 151). As you click an issue in the structure, you can see the issue details in the panel on the right.
- History (see page 166). You can see the list of changes done to the structure and navigate through them to see the previous versions of the structure.
- Other items details. If you have Structure.Pages installed, you can display the Confluence page contents there.

You can switch to the two-panel mode using the **Toggle Panels** button and menu in the Main Structure Toolbar (see page 38).



Clicking the button opens/closes the second panel with the structure widget. By default the widget opens with the JQL search and you can switch to text search, clipboard or another structure by clicking the JQL label.



Resizing Secondary Panel

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

Swapping panels

You can swap panels completely using corresponding quick action:

1. Open quick actions menu by pressing s+q shortcut;

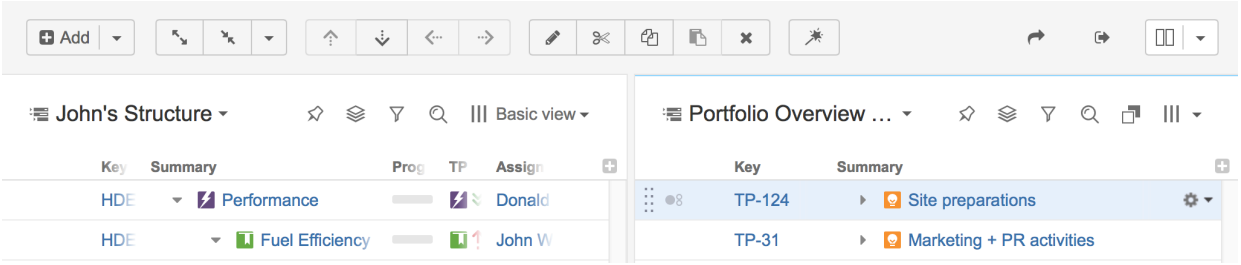
2. Type 'swap' and find the swap action;
3. Apply the action by pressing enter.

Structure Widget on Secondary Panel

The structure widget that you open in the secondary panel is fully functional and differs very little from the widget in the main panel on the left. In both of them you can open structures, run JQL and Text search and open the clipboard.

Just like the main panel, it has its **panel toolbar** and the **view menu**.

You can also use the Main Structure Toolbar actions to work with the secondary panel widget. The toolbar actions will be applied to the panel that is in focus. The focused panel is highlighted with a thin blue line at the top.



The toolbar of the secondary panel has one extra function - hide/show the items that exist in the panel in the right:

This is especially useful when you need to make sure that the structure in the main panel has all the issues you've found using search in the secondary panel.

Issue Clipboard

The structure widget on the Structure Board allows you to see not only the structures and search results, but also the clipboard contents. To see what you have in the clipboard, click the structure name or the search type label and select Clipboard from the menu.

You can open it both on main and secondary panel (if in two-panel mode).

For details about using Issue Clipboard, see Using Cut, Copy and Paste (see page 147).

1.4.9 Changing Structure

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

- Adding existing issues and other items (see page 142) to the structure
- Moving items (see page 143) inside the structure

- Creating new items inside the structure
- Removing items (see page 144) from the structure.

There are several ways to make these changes. Some of these operations can be applied to a group of items (see page 144) and some to individual items only. See the respective subsections for more details.

See also: Creating New Items (see page 154)

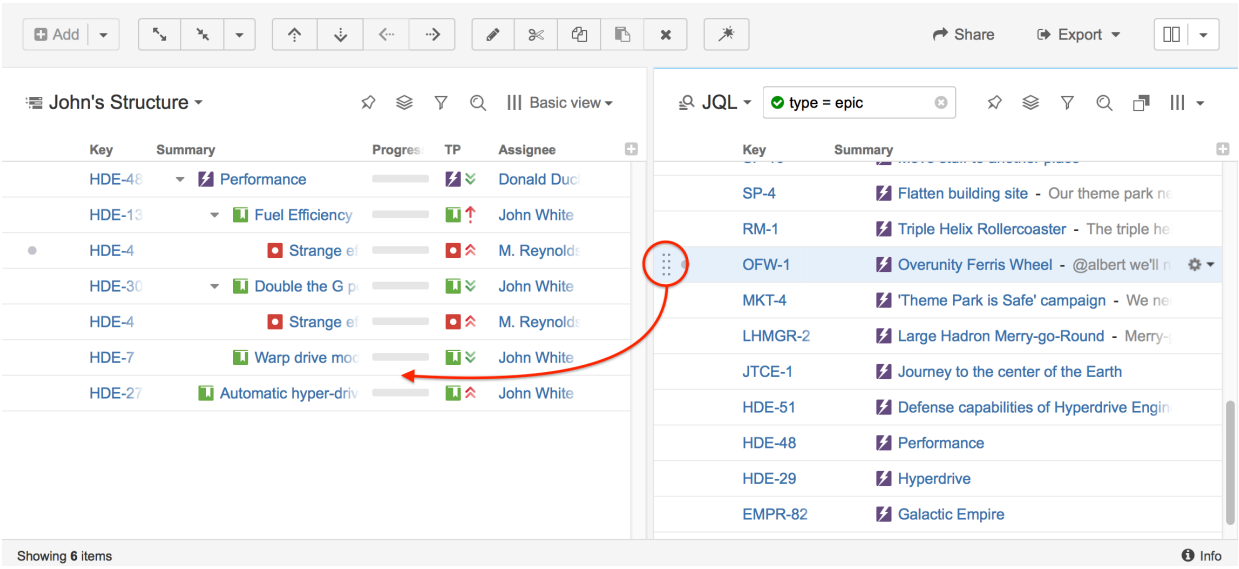
Adding Existing Issues to Structure

You can add an issue to a structure both from the Structure Board (see page 15) and from the Issue Page (see page 19).

On the Structure Board open the text or JQL search on a secondary panel (see page 139), find the issues you need and add them to a structure using drag-and-drop (see page 145), copy /paste (see page 147).

Instead of search you can also open a structure, which contains the items you are looking for and add them to your structure from there.

You can also add multiple items at once. To do that, select the items you need (see page 36) and add them using drag-and-drop (see page 145), copy/paste (see page 147).

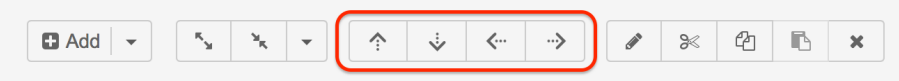


On an Issue Page, select the structure where you need to add an issues and use the Place (see page 19) button.

Moving Items 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 items's parent, moves the item up and places it before the previous child - if possible.
Move Down	Ctrl + Down	Without changing the items's parent, moves the item down and places it after the next child - if possible.
Level Up / Outdent	Ctrl + Left	Move the item one level up. This will place the item after its parent.
Level Down / Indent	Ctrl + Right	Move the item to be a sub-issue of its current preceding sibling.

When you move an item that has sub-items, 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 **synchronization** icon in the widget status bar that tells whether there are pending uploads or downloads.

Moving an Item to an Arbitrary Position

The basic moves can only adjust item position one place at a time, so if you need to place an item at a specific position not close to its current position, you can do that with Drag-and-Drop (see page 145) or Cut & Paste (see page 147). Cut & Paste also allows to copy hierarchy from one structure to another.

Moving item with Drag-and-Drop or Cut & Paste can be undone (see page 150).

Multiple Selection

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

See also: Selecting Multiple Items (see page 36)

Removing Items from Structure

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

You can select multiple items (see page 36) and remove them all in one action.

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

Removing items can be undone (see page 150).

Changing Multiple Items

You can apply most of the changes to multiple items in one action. Select multiple items (see page 36) and use toolbar, keyboard shortcut or drag and drop.

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

The following actions work with the multi-selection:

- Drag and drop (see page 145) lets you move a selection of items within a structure or add them to a structure from the secondary panels (see page 139), such as Clipboard or Search Results.
- Cut and paste (see page 147) allow you to move items both within a structure and between different structures.
- Remove button (see page 144) or *Delete* key lets you remove multiple items from the structure.
- Toolbar buttons *Move Up*, *Move Down*, *Indent*, *Outdent* are allowed for multiple items only if all items in the selection are at the same level in hierarchy and have the same parent item.
- Bulk Change (see page 164) button lets you use JIRA bulk change wizard with selection of issues from the structure.

See Selecting Multiple Items (see page 36) for details about working with multi-selection.

Using Drag and Drop

Drag-and-drop feature allows you to quickly move or copy items or selections of items within the structure or add them from the secondary panels (see page 139) to the structure.

Basic Drag-And-Drop

To grab an item, move your mouse pointer over the "handle" of the item (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 drag items by grabbing them anywhere in the row. To do that, press **Shift** on the keyboard and press mouse button anywhere on the row except the link. It is especially useful if you need to outdent an item, since the drag handle is usually close to the edge of the screen.

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

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

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

Moving vs Copying

As you move an item you will also see a hint saying if the item will be **moved** or **copied**. As you move items inside the structure, the default behaviour is moving. If instead of moving an item, you want to create a copy of it in a new place, hold the **Alt** (for Mac) or **Ctrl** (for PC) key as you move it.

If you have two structures open side by side and you drag an item from one to another, the default behaviour is copying (so the item ends up in both structures). If you want to move it from one structure to another (so it's removed from the original structure) hold the **Alt** (for Mac) or **Ctrl** (for PC) key as you drag the item.

Dragging Multiple Items

To move more than one issue, first select multiple issues (see page 36) (start with hitting **Space** button) and then move them using the "handle" of one of the selected items. Holding **Shift** and dragging by any other place on the item row also works.

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

Cancelling Drag

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

Drag-and-drop can also be undone (see page 150).

Impossible Moves

If it's not possible to move the dragged items onto the current position (for example, due to structure permission settings), you will see the **Drop is not possible** message.

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 items. Just move the items over the top or bottom edge of the structure widget and the structure will be scrolled up or down. The further you move the dragged items, the faster the scrolling is.

Using Cut & Paste (see page 147) may be more effective than Drag-and-Drop if you need to move several items to distant positions.

Using Cut, Copy and Paste

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

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

While drag and drop (see page 145) is a nice and visual way to rearrange items, it might get tedious if you have to drag items across long distances. Copy and paste solves that problem perfectly – copy items to clipboard and scroll through the structure looking for a place to paste them to.

See Selecting Multiple Items (see page 36) for details about multi-selection.

Copy / Paste Scenarios

There are two main scenarios for using the Issue Clipboard:

> Please note that if you 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.

Moving Items Between Structures

The contents of the clipboard is preserved in the **current browser window**, which allows you to copy/cut items in one structure and paste them into another.To copy items (with their sub-items) from one structure to another do the following:

Cut/Copy

First add the desired items to the clipboard:

1. Open the structure to cut/copy from.
2. Select the items you want to cut or copy. Either select a single item, or use multiple select (see page 36).
3. Click the **Cut/Copy** button on the structure toolbar (or press **Ctrl+x/Ctrl+c** or **Command+x/Command+c**).
4. Selected items will be added to the clipboard and marked with a small scissors icon for cut and the clipboard icon for the copied .

> Note, that the cut items are not removed from the structure until you paste them into another structure.

> If the copied item contains sub-items, these sub-items are not automatically copied with their parent. You need to select them explicitly. To copy an item and all of the children, select a parent item, press Shift+Arrow Right (this selects an item and all of the sub-issues) and then the Copy button.

Paste

After you have cut/copied the items, 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 (see page 15)).
2. In the structure grid select the item after which the items from the clipboard should be placed.
3. Either click **Paste** button on the toolbar (or press **Ctrl+v** or **Command+v**) to place the items **after** the selected issue at the same indentation level, or press **Ctrl+Shift+v** (or **Command+Shift+v** on Mac) to place the items **under** the selected item (as the children).

When you paste items from a different structure, it's possible that the target structure already contains some of them. In this case the existing items will not be affected and new copies will be created as you paste.

If you need to copy the same set of issues to several different structures, you can open the clipboard in the secondary panel (see page 139) and use drag-and-drop operation to move the issues instead of using Paste. In this case the issues will not be removed from the clipboard.

The Paste operation can be undone (see page 150).

Moving Items Within A Structure

Instead of using drag-and-drop (see page 145) function to move items 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 items from the top of the structure to the bottom or the other way around.

Cut

First add the desired items to the clipboard:

1. Select the items you want to cut. Either select a single item, or use multiple selection (see page 36).
2. Click the **Cut** button on the structure toolbar (or press **Ctrl+x** or **Command+x**).
3. Selected items will be added into the clipboard and marked with a small scissors icon .

The cut issues are not removed from the structure until you paste them to a new location.

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 **Clipboard** panel, you can open it in the secondary panel (see page 139). However, this is not necessary to use the Cut/Paste function.
2. In the structure select the item after which the items from the clipboard should be placed.
3. Either click **Paste** button on the toolbar (or press **Ctrl+v** or **Command+v**) to place the items **after** the selected item at the same indentation level, or press **Ctrl+Shift+v** (or **Command+Shift+v** on Mac) to place the items **under** the selected item (as the children).

If the cut item contains sub-items, these sub-item are pasted with their parent.

After Paste the clipboard is cleared.

The Paste operation can be undone (see page 150).

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 (see page 142) items from search results;
- Removing (see page 144) items from a structure;
- Drag-and-Drop (see page 145);
- The Paste operation of a Cut & Paste (see page 147) 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 items, 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.

When you remove items (see page 144), a notification pop-up with an "undo" link also appears at the top of the page.

Currently only the last operation can be undone, but we are working on the new functionality for Undo and it will be added in the future versions.

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 (see page 166).

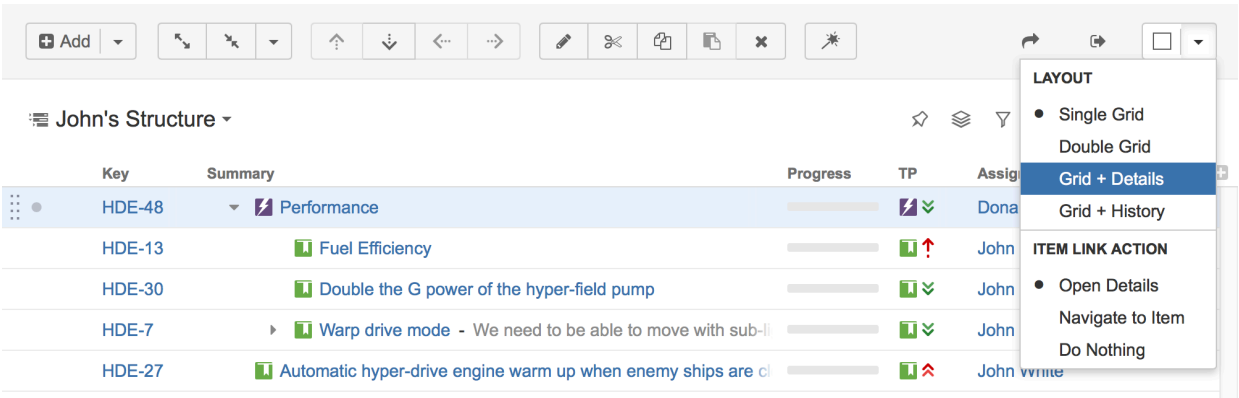
1.4.10 Working with Issues

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

Viewing Issue Details

As you work with a structure or search results on the Structure Board, you can open the full issue information in the issue details panel on the right.

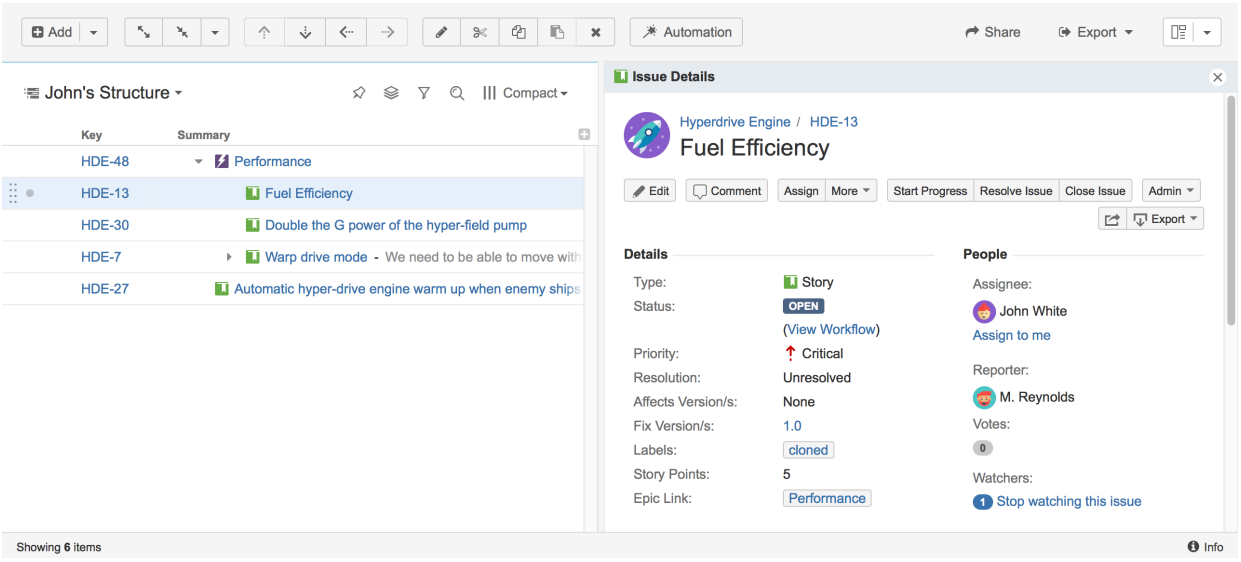
By default the issue details panel opens as you click the issue link (key or summary). You can also access it via the **Toggle Panels** menu:



You can define what should happen when you click the issue link in the structure. By default the Issue Details panel is opened. If you don't like this behaviour, you can open the standard JIRA issue page instead, or do nothing. You can switch between these two options in the **Toggle Panels** menu in the **Item Link Action** section.

Working with issue

In the details panel, you can work with the issue in the same way as in the Issue Navigator: edit , view and add comments, share, view history, view development information, and do many other things. We refer you to the JIRA documentation for more information on that.



To see details for another issue in the structure, simply select another issue (see page 35) by clicking it or pressing arrow keys.

To close the issue details panel, click the close button in its top right corner.

You can use Structure's inline editing (see page 158) when the details panel is shown. Details panel will be disabled while you are in the edit mode.

You can also open issue page in a separate browser tab or window by clicking issue key or summary in the table while pressing `Ctrl` (Mac: `Cmd`)or `Shift` on the keyboard.

Separate view for issue details

As the details panel is typically wide, the space for the main Structure panel is constrained, so you might want to use a different view with less columns when the details panel is open. By default, a compact view with only *Key* and *Summary* columns is used.

Structure remembers views used with and without issue details, and switches between them automatically when you open or close issue details.

> Default views and the list of views in the View Menu for both modes (with and without issue details) are configured in View Settings (see page 196).

Resizing details panel

You can divide the horizontal space between the details panel and the main panel by dragging the separating border. Structure remembers the ratio of the details panel width to the window width, and it will maintain that ratio when you open Structure Board the next time or resize your browser's window.

Details panel width is remembered for the selected view. Thus, if you select another view and adjust the details panel width, the original width will be restored when you select the original view.

Details and secondary panels

Details panel displays information only for the issues selected in the main panel. If you have the secondary panel open when you click an issue in the main panel, secondary panel (see page 139) will be hidden while the details panel is open and restored back when the details panel is closed.

Using keyboard

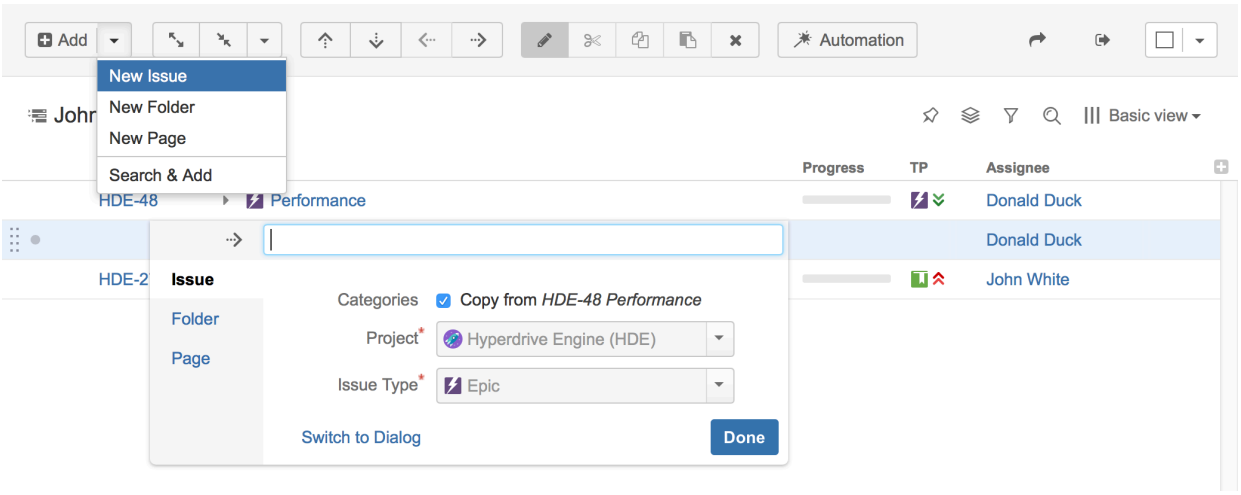
Use **o** or **Shift+o** to show/hide the details panel.

As with the secondary issue panels (see page 139), you can switch keyboard focus between panels using the **** (backslash) shortcut. When the focus is in the issue details panel, keys like or (also `PgUp, PgDn, Home, End, or , , ,`) scroll the details panel, while all other Structure shortcuts (see page 285) work as usual (including `j` and k which select next/previous issue in the structure). All shortcuts available to you on the Issue Page should also work as usual: e.g., **,** (comma) should open field selector, **e** should open Edit Issue dialog, etc.

When you open the details panel with **o**, the details panel is automatically focused. **Shift+o** doesn't switch focus.

Creating New Issues

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



Quickly Add a New Issue

1. Use the **Add** button in the toolbar and select **New Issue** or press **Enter**. Use **Shift+Enter** to create an issue as a sub-issue of the currently selected issue.
2. Specify the issue summary.
3. Make sure you have the **Categories** checkbox cleared, select the project and the issue type.
4. Press **Enter** or click **Done** to finish editing and create a new issue on the server.

> Hit **Escape** to cancel creating a new issue.

Create a New Issue Based on Existing Issue

As you create a new issue, you can copy Project, Type and most other attributes from the previously selected issue.

1. Select an issue in the structure.
2. Use the **Add** button in the toolbar and select **New Issue** or press **Enter**. Use **Shift+Enter** to create an issue as a sub-issue of the currently selected issue.
3. Specify the issue Summary.
4. Make sure **Categories: Copy from...** checkbox is selected.

5. Press **Enter** or click **Done** to finish editing and create a new issue on the server.

When you create an issue this way, the following fields are copied 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.

Editing Other Fields during Creation

The New Issue panel only allows you to select the project and the issue type or copy some fields from another issue, but sometimes you may want to edit more fields (for example, if you have some required fields).

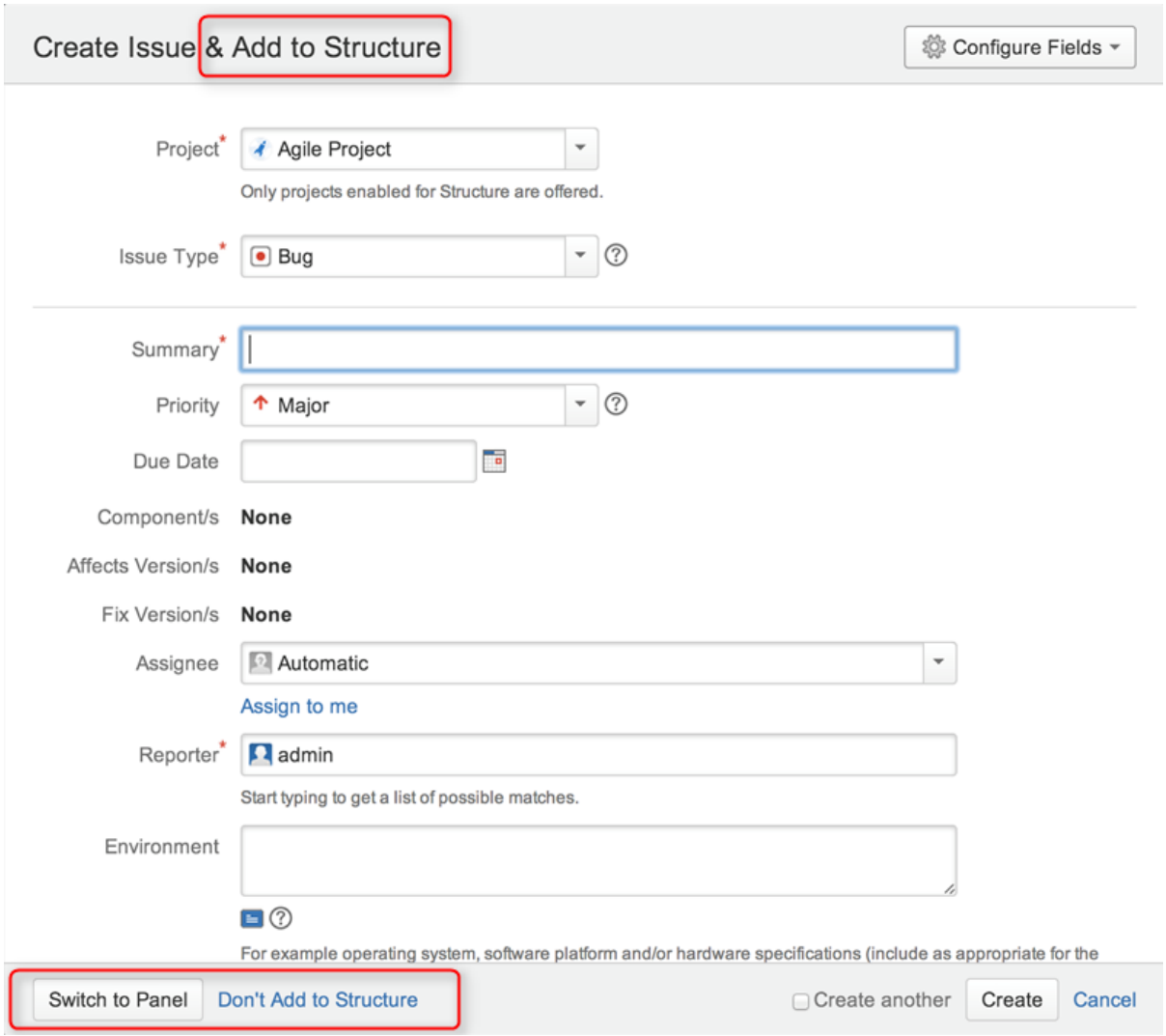
To be able to edit them, you need to add the corresponding columns to your current view before you start creating a new issue. Then, when editing a field, you can navigate to other fields by using **Tab**, **Shift+Tab**, or **Ctrl+Alt+arrows ()** keyboard shortcuts, or simply by clicking a cell you wish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

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 provide all required values.

You cannot create a JIRA sub-task from scratch in the structure widget, but if you already have one in the structure, you can create a sub-task with the same parent using the copying option (this refers to JIRA sub-task issue type, not Structure sub-issues).

Using "Create Issue" Dialog

You can click **Switch To Dialog** in any editor panel to display JIRA's **Create Issue** dialog, which allows 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. You can also switch back to dialog mode at any time. The system will remember the last used mode (dialog or panel) and use it next time you start creating a new issue.

You can click **Don't Add to Structure** if you want to create an issue, but don't want to add issue to the structure.

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 press **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 (see page 158) – you can also enter values for other fields besides Summary by clicking on the field to be modified, or using keyboard shortcuts (see page 160). 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 (see page 163).

You need Edit Issue permission on the issue to edit its fields. If you don't have the permission, a read-only flag (see page 65) is shown.

Entering Edit Mode

You use the Edit Mode when you are editing an existing issue or creating a new issue (see page 154). To edit a value displayed in the Structure widget, do one of the following:

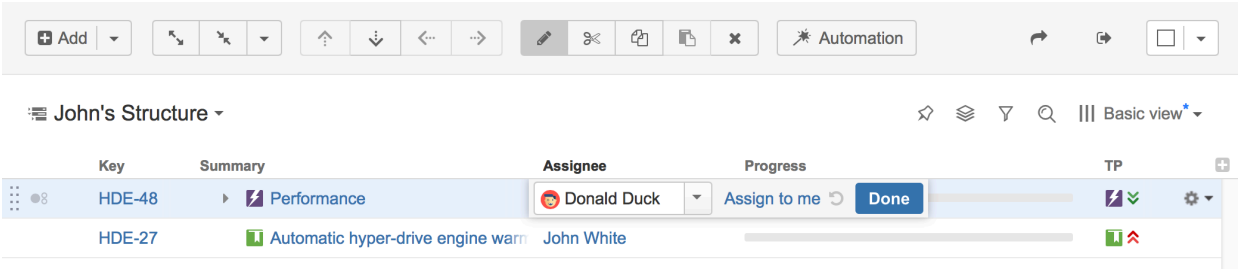
- **double-click** that value;
- select the issue and click **Edit** button on the toolbar;
- select the issue and use a keyboard shortcut – **Tab** or **F2.**

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 (see page 160)).

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 (see page 163)
.

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 used on the Edit Issue page, but is designed to be a bit more compact.

- All help texts, descriptions and field labels are not shown. Hover mouse pointer over the input field to see help and field description.
- 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.)

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* **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.
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	

Keyboard Shortcut	Action
	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.

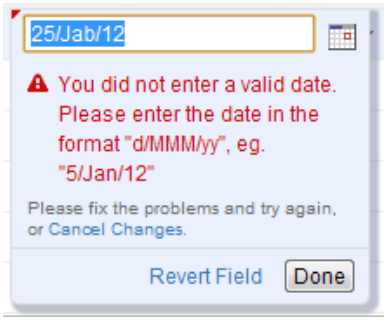
See Also: Keyboard Shortcuts (see page 285)

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 the warning message or 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 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 allows editing issues too, but due to some incompatibilities between field editors and gadget framework, not all fields can be edited.

The following JIRA fields are editable from Structure Gadget:

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

To edit other fields, open Structure Board or issue page or any other page with the structure.

To be able to edit the fields in the gadget, the user should have permissions to edit them and the fields should be present on the Edit Screen.

Structure Notes (see page 124) can also be edited in Structure Gadget.

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:

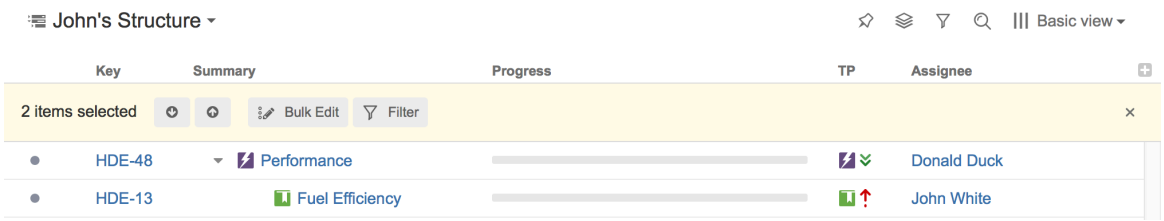
- click on another field that you need to edit;
- use Tab, Shift-Tab, Ctrl+Alt+arrows keyboard shortcuts (see page 285) to move to the next/previous fields.

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 (see page 36) 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 (see page 285) for selecting issues.
2. Click **Bulk Edit** action on the panel.



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 (see page 201).

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 (see page 36).
2. Use **Copy** action on the toolbar (or hit Ctrl+C / Command+C) to copy the issues to the Issue Clipboard (see page 148).
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 (see page 201) . 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

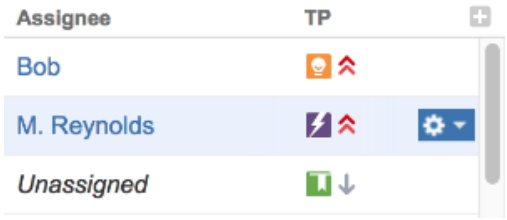
Structure widget lets you use JIRA actions available for the issues from the JIRA's Action & Operations Drop-Down and JIRA keyboard shortcuts for the most frequent actions.

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 (see page 17), keyboard shortcuts are always applied to the viewed issue - regardless of the selection in the structure!

No Page Reload

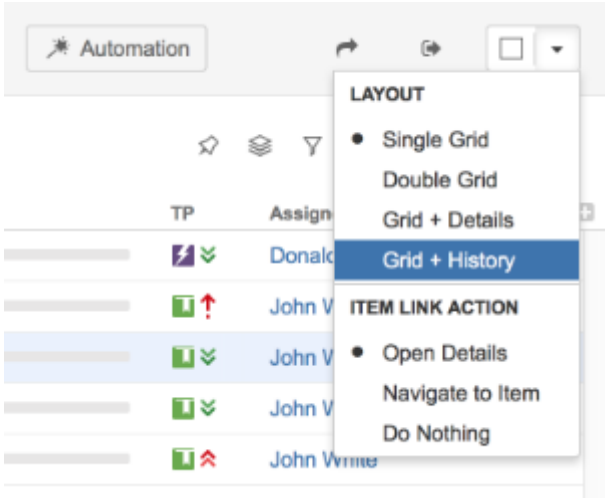
In many cases Structure is able to proceed without page reload after you have applied a JIRA action to an issue. The applied changes are immediately visible in the Structure widget, and that gives you a very smooth experience of working with a collection of issues.

> 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 (see page 15), most actions do not require page reload.

1.4.11 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 **Toggle Panels** menu button and select **Grid + History**. The list of recorded changes will appear in the **History** panel on the right.



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

History does not work for dynamic parts of the structure. Changes done to issues added to the structure by generators (see page 178) will not be stored. However, the addition, moving and removal of the generators themselves is 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.

New changes are loaded and added to the top of the list as they happen.

For each change, the following information is shown:

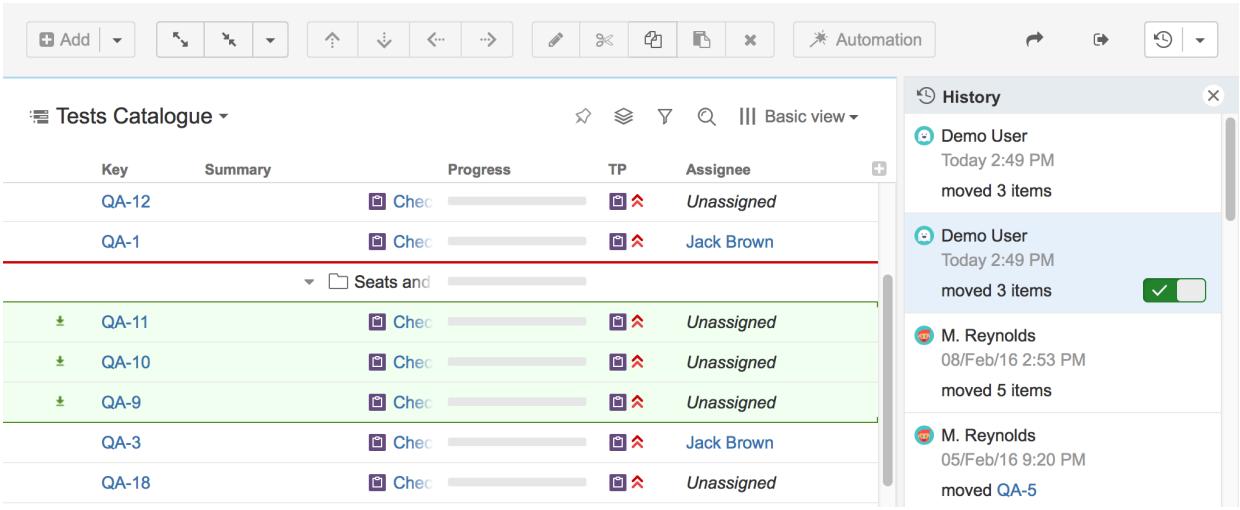
- The avatar and the name of the user who has made the change. In JIRA 4.4 and up you can hover your mouse over the user avatar to see the user details.
    - If the change has been made by a synchronizer, the synchronizer's name is shown. User avatar displays the user account that the synchronizer was 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 only tracks the structure changes, not the changes of JIRA fields. All columns with issue fields show current values – **not** the values that the issue had when the structure change was made.
- You cannot edit issues, create new issues or change structure when viewing history.
- The history cannot be modified. (The administrator can clear the entire Structure history.)

Printing a Previous Structure Version

You can Open Printable Page (see page 171) 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.

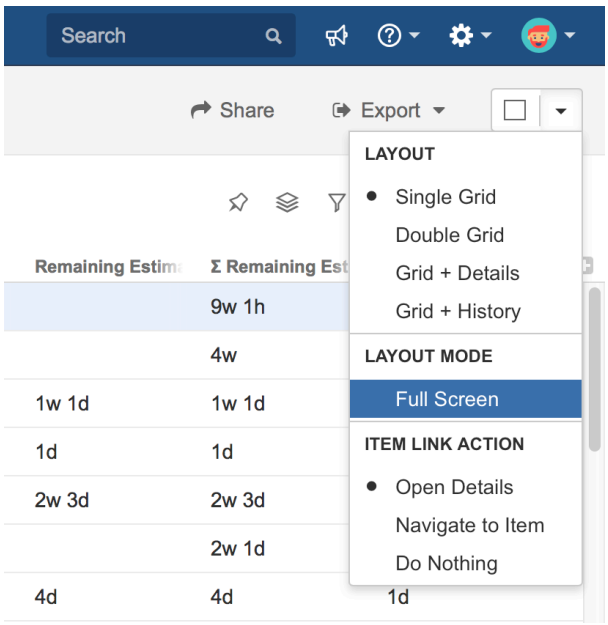
Note that all limitations apply: the current values of the fields will be displayed and Progress and other aggregate columns will not be displayed.

Exporting a Previous Structure Version to XLS Format

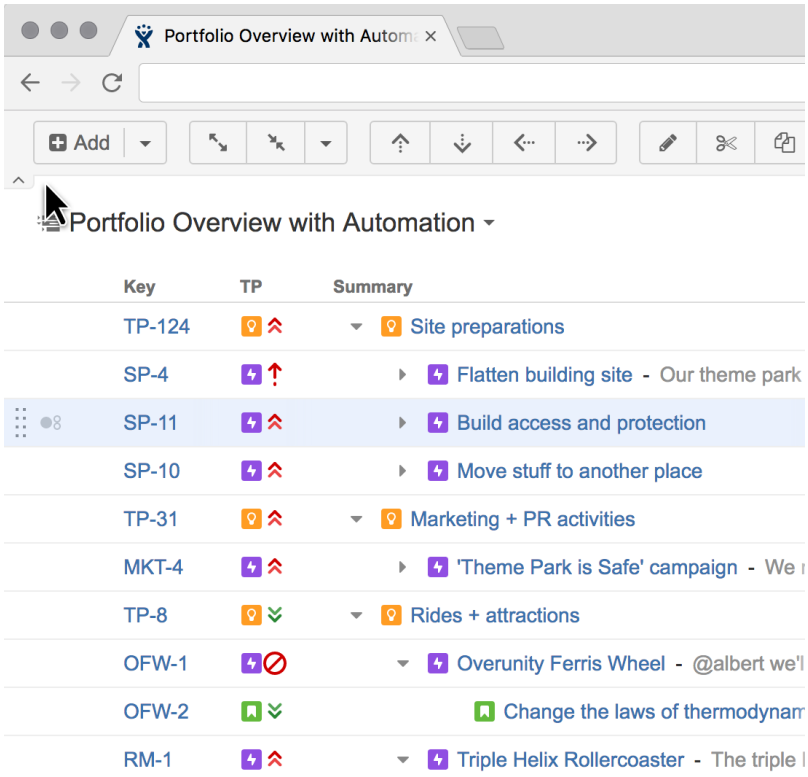
Like with the printable page, you can export structure to XLS (Excel) (see page 172). The XLS file will contain structure as it was after the selected change has been applied.

1.4.12 Full Screen Mode

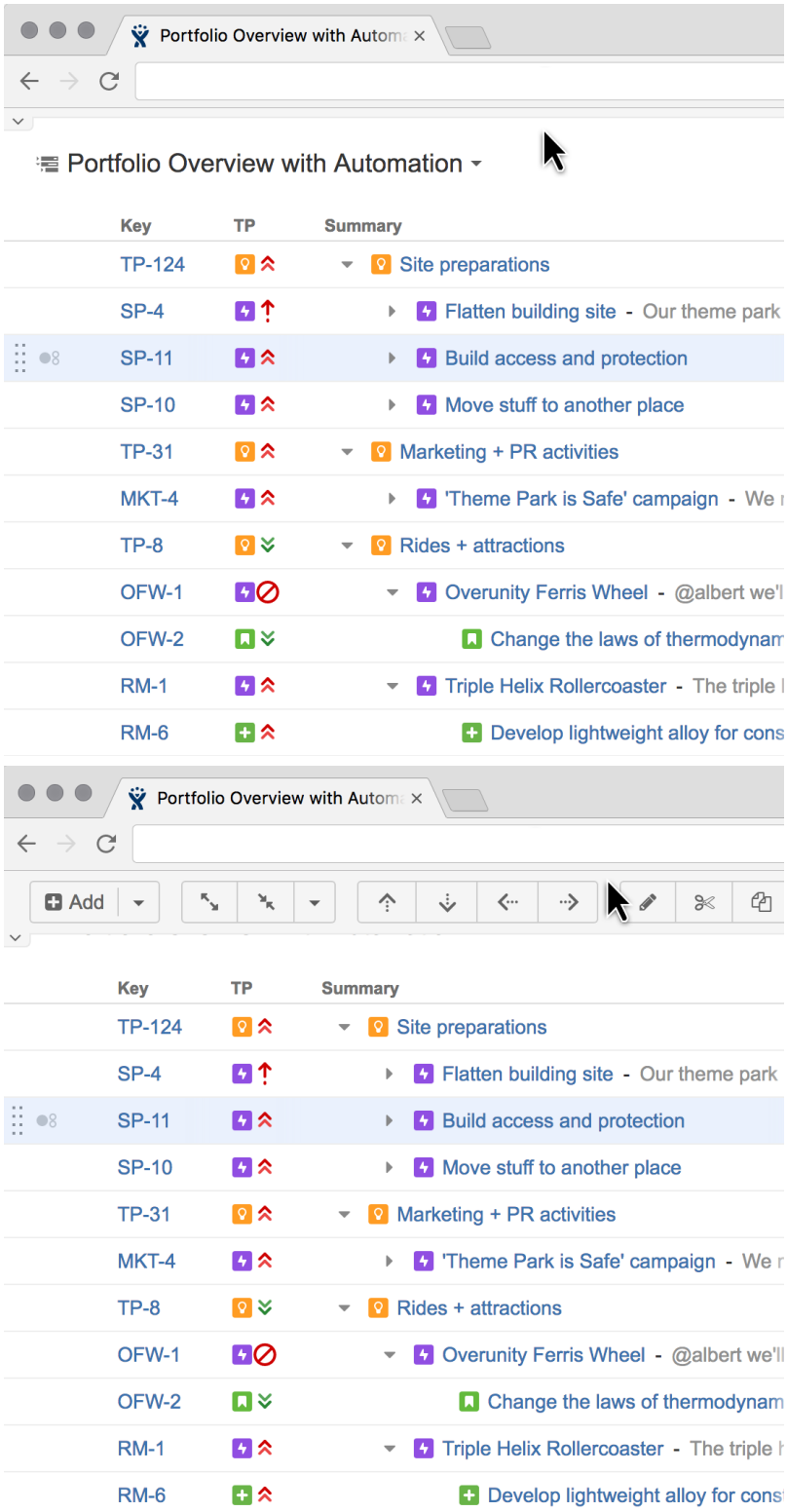
When working with the Structure Board you can turn on Full Screen mode to give more screen space to your data. Full Screen mode can be toggled using the **Toggle Panels** menu or by pressing Z on your keyboard.



In Full Screen mode the JIRA application header is hidden and the main Structure toolbar becomes more compact. You can collapse the main toolbar by clicking the Collapse button to save even more screen space.

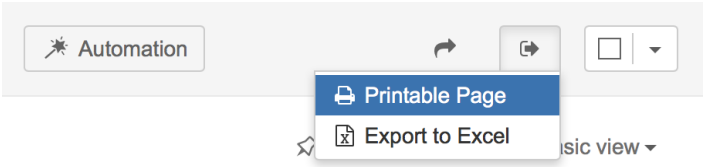


Bring the mouse cursor over the collapsed toolbar and the toolbar will show up temporarily, allowing you to perform a quick action.



1.4.13 Printing Structure

Printable page lets you print the current structure from the browser.Click the **Export** button on the structure toolbar and select **Printable Page**.



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 open the page 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.

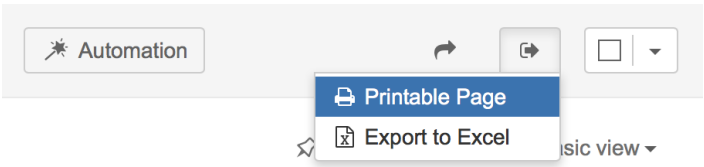
Depending on the number of columns, and the amount of text, it may be necessary to adjust font size before printing.

> 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.

1.4.14 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 **Export** button in the toolbar and select **Export to Excel**.



The browser will download a new XLS file, which you can save or open. The XLS file will contain all the issues that are present in the structure and the hierarchy will be preserved.

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.

Note that XLS format allows up to 65536 rows in the spreadsheet, so a larger structure wouldn't fit - use filtering to hide some of the issues, if you have more than that.

Likewise, if you have issues that are more than 15 levels deep in the hierarchy (have more than 14 "parents" and "grand-parents"), in Excel they will all be shown on level 15, 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.

Column Type	Notes
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 (see page 171).

1.4.15 Real-Time Collaboration

Structure widget is a real-time collaboration tool.The 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. Items 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 Main Structure Toolbar (see page 38) and turn on Continuous Polling with the button

1.5 Automation

Automation is a powerful feature that lets you create **dynamic structures**. Unlike a manually created structure, a dynamic structure can update itself when there are changes in JIRA. You can make parts of your manually created structure dynamic – for example, automatically place all issues that match a query under a manually added folder.

Automation works through **generators** – special rules which define, how the issues should be organised. Generators are added right inside the structure just like other items and their scope is defined by their position in the structure.

With generators you create a structure "skeleton" - the rules based on which your structure should be built, and only the skeleton is saved, not the issues. This means the structure is generated when you open the structure - hence the **Generators**.

Once you open some structure, the automation will start checking for relevant changes in your JIRA and update your structure as necessary, ensuring that the structure you are seeing is up-to-date. So there is no need to reload the page to see the latest updates.

1.5.1 Types of Generators

- Insert. (see page 178) Insert generators allow you to automatically add issues to the structure. It can be the results of a JQL or text query, Epics or Stories from an Agile board, or an entire structure. If some issue is updated and no longer satisfies the query, it's not shown anymore.
- Filter. (see page 179) If you want to remove some of the issues after you've built the structure (for example, remove all closed issues), you can use the **Filter Generator** and specify the query that describes the issues that you want to keep - the rest will be removed from the structure.
- Sort. (see page 185) To arrange your issues in a certain order you can use the Sort generator.
- Group. (see page 186) With this function you can break a list of issues into groups based on your JIRA fields.
- Extend. (see page 186) Using the extend functions you can pull in issues related to those that are already in the structure. You can extend with sub-issues, JIRA links and Epic links (if you are using JIRA Agile).

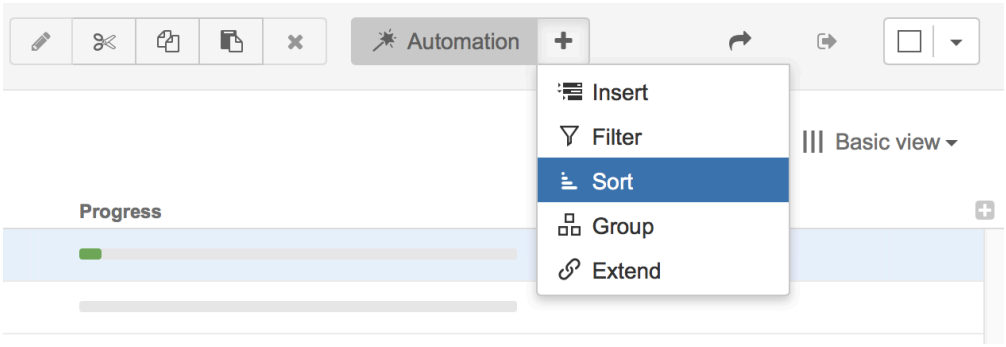
1.5.2 How to Add a Generator

1. Normally, generators are not shown – you need to switch on the **Automation** button to be able to see and configure generators.

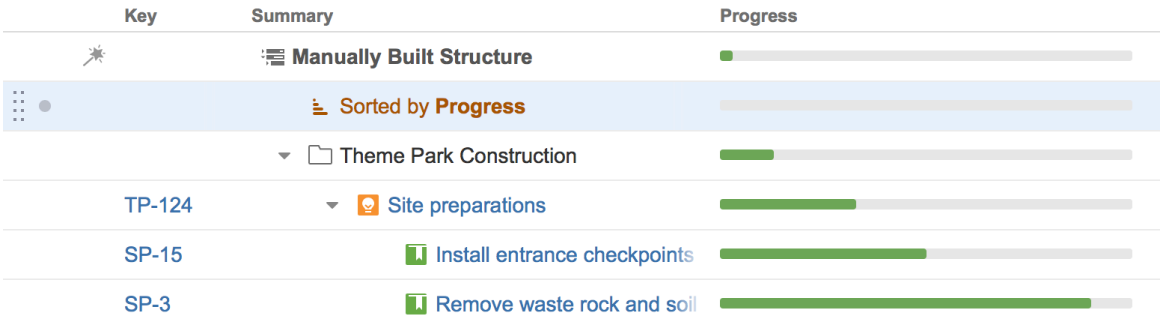
2. As you click the button, you will see the top level item with the structure name appear at the **Root** of the structure - this item works a parent for the whole structure.
3. Select where you want to add your generator - you can select the **Root** item if you want the generator to be applied to the whole structure, or select some static item if you want the generator to work under this item only.

> You can only add generators under static parts of the the structure. You can not add them under items, which are added by other generators.

4. Now click the **+** icon next to the **Automation** button and select the type of generator you need.



5. For some generators you will need to select some options. Once you did that, click **Apply** . You will see the generator added to the structure.



6. Click the **Automation** button to hide the generators from the structure.

1.5.3 How to Edit a Generator

1. If you want to edit some existing generator, first switch on the Automation Editing mode by pressing the Automation button.
2. Find the generator you want to edit.

3. To change the scope of the generator, simply move it to the new place in the structure, just as you would move any other item. You can use drag-and-drop, or copy/paste.

> You can not move the generator under an item, that was added by a generator. Only the static items added manually.

4. To change the generator settings, double-click the generator or use the **Edit** button in the structure toolbar.
5. Make the required changes and click **Apply** to save to them.
6. Click the **Automation** button to hide the generators.

1.5.4 How to Remove a Generator

1. If you want to remove some existing generator, first switch on the Automation Editing mode by pressing the Automation button.
2. Select the generator you want to delete and press **Delete** or use **Delete** button in the toolbar.
3. Click the **Automation** button to hide the generators.

1.5.5 Generators

There are several types of generators, that are available by default:

Insert

Insert generators allow you to automatically add issues to the structure. The issues can be the result of a JQL or text query, Epics or Stories from an Agile board, or an entire structure.

As you add and Insert generator you can choose from the following options:

- Agile Board. Will add epics or stories from the Agile board that you select.
- JQL Query. Will adds results of the JQL query.
- Text Query. Will add results of a text search.

> Please note, that both for text and JQL, options only issues from the projects for which Structure is enabled (see page 305) will be added.

- Structure. Will add the selected structure into your structure. This option is especially useful when several teams work with their own structures and you want to create a structure with an overview of them all.

  When you insert a structure into another structure, it's always a good idea to create a special folder for this structure and insert it there. Thus you'll be able to see where the inserted structure begins and ends.

  If you have allowed editing via structure (see page 188) for this generator, you can update the inserted structure right from your master structure. For example, if you have inserted structure A into structure B, as the users work with structure A, you will see all the changes reflected in structure B. And the other way around - if you open structure B and start modifying structure A in it, the actual structure A will be updated too.

Since the generators run every time you open the structure, the list of issues added by the inserter is always up-to-date. If some issues no longer pass the query criteria, they won't be added. Also, if you already have the structure open, the generator will be tracking the changes in your JIRA and add/remove issues as necessary.

Filter

Filter generator allows you to hide from your structure issues that don't pass certain criteria. You can use JQL, Text and S-JQL (see page 252) queries.

Once you add this generator, you will only see the items from the structure that pass the query and their parents. The parents are necessary to show the issues without breaking the hierarchy.

Additional Options

- All filters have an option to show not only matching issues, but also all their descendant. Select the **Show all sub-items of matching items** option in the generator settings to enable this.
- JQL Filter has an option that allows you to keep non-issues as you apply the filter. Since the folders and other types of items are not returned by the JQL, this is the only way to tell the filter not to remove them.

Inserter/Extender Duplicates Filter

This filter should be used when you want to add a number of issues into your structure using an Inserter and then arrange them into a hierarchy based on the links which exist between them.

Here is how you can do it:

1. The Inserter adds a number of issues on the top level.
2. The Link Extender adds the issues which are linked to them as children.

If some of the children have already been added to the top level by the Inserter, you will get duplicates - an issue at the top level also added as a child.

The Inserter/Extender Duplicate Filter will remove such issues and will only keep the children. Please see examples below for a more detailed explanation.

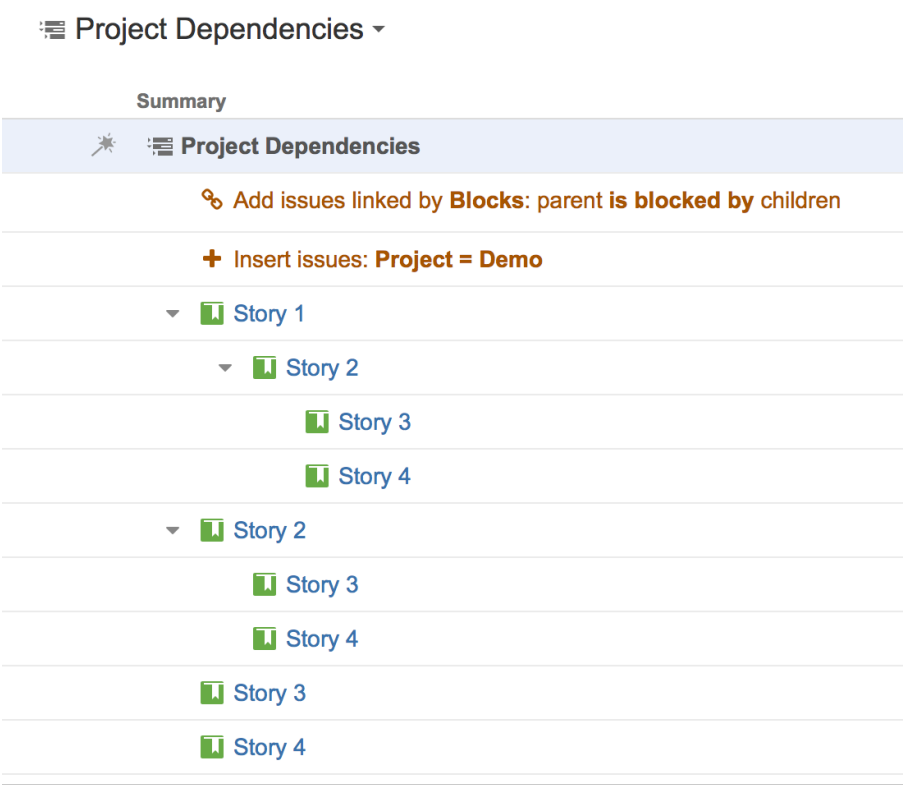
Basic Example

Imagine we have a project with issues Story 1, Story 2, Story 3, and Story 4, and some of the issues are blocking other issues:

- Story 1 is blocked by Story 2
- Story 2 is blocked by Story 3 and Story 4

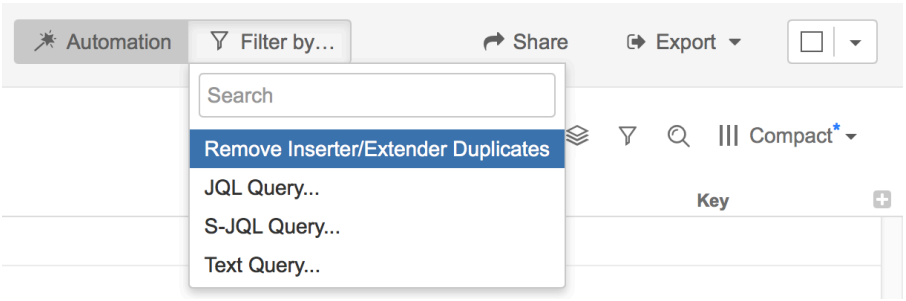
In our structure we want to see all issues from our project arranged based on the existing "Blocks" links.

After you add all four issues by a JQL Inserter and add a Links Extender, you will get the following hierarchy:

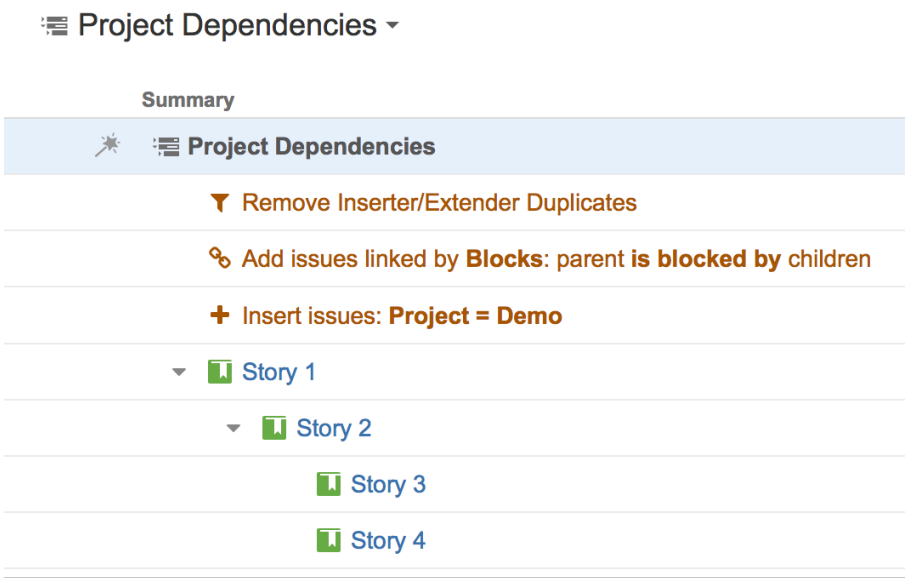


You can see that the issues are now duplicating as the Extender just adds the children under parents, not moves them.

Now let's add the Duplicates Filter:



As a result we get a structure with the hierarchy and no duplicates:

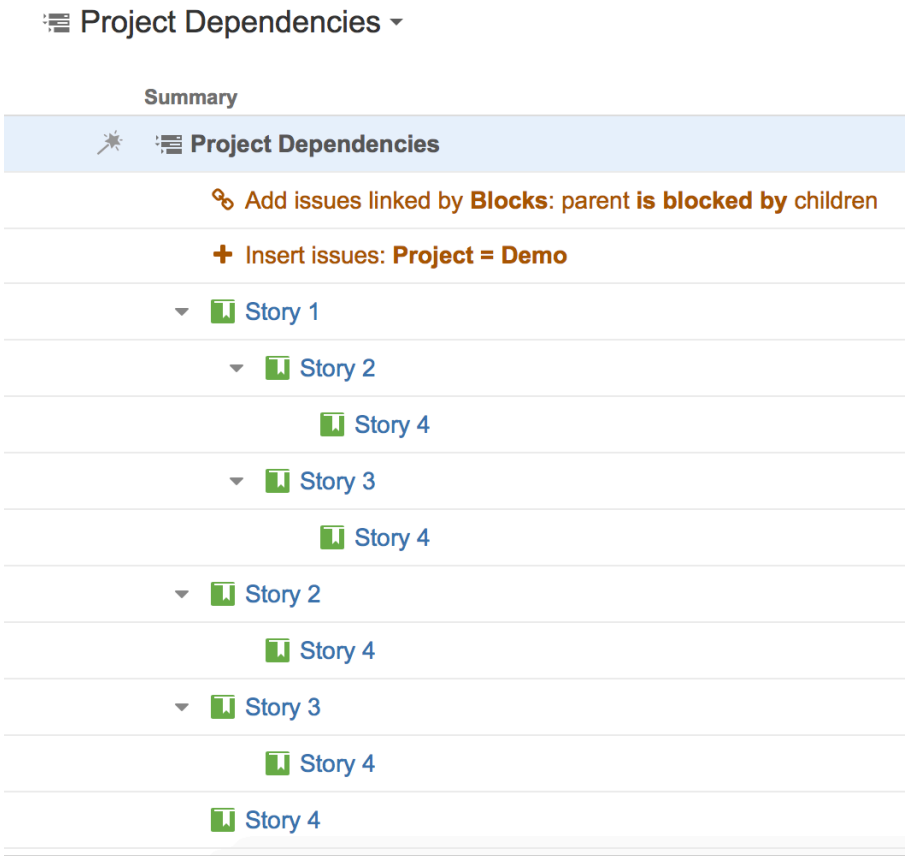


Multiple Parents Example

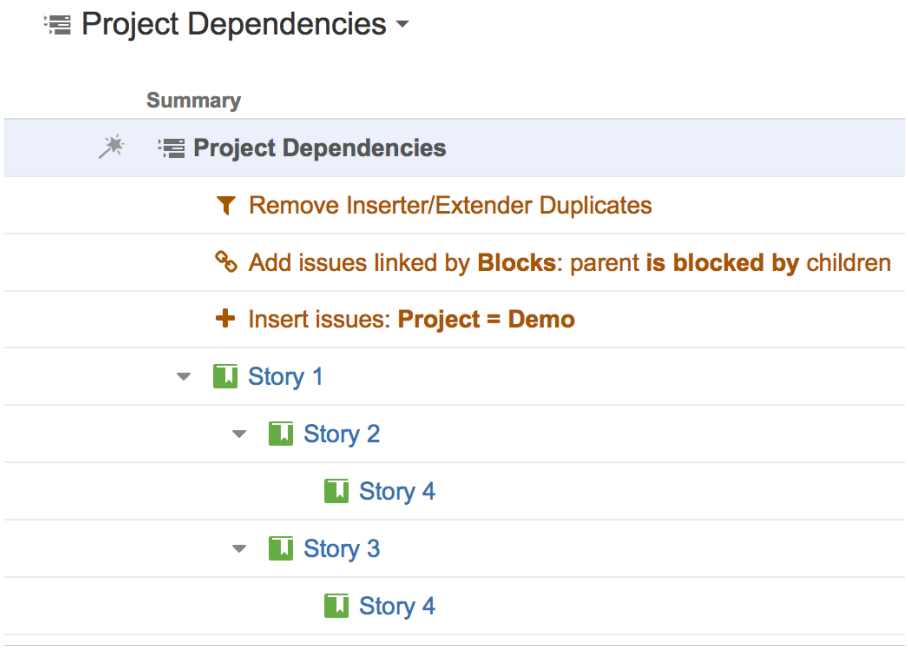
We have the same situation as in the example above, but we have one story that blocks two other stories, so it should be shown under both of them:

- Story 1 is blocked by Story 2 and Story 3
- Story 2 and Story 3 are blocked by Story 4

So without Duplicate Filter it looks like this:



And with the filter applied all the duplicates are removed. Please note that Story 4 is present in two place to reflect the hierarchy and in this case these two instances are not duplicating each other:

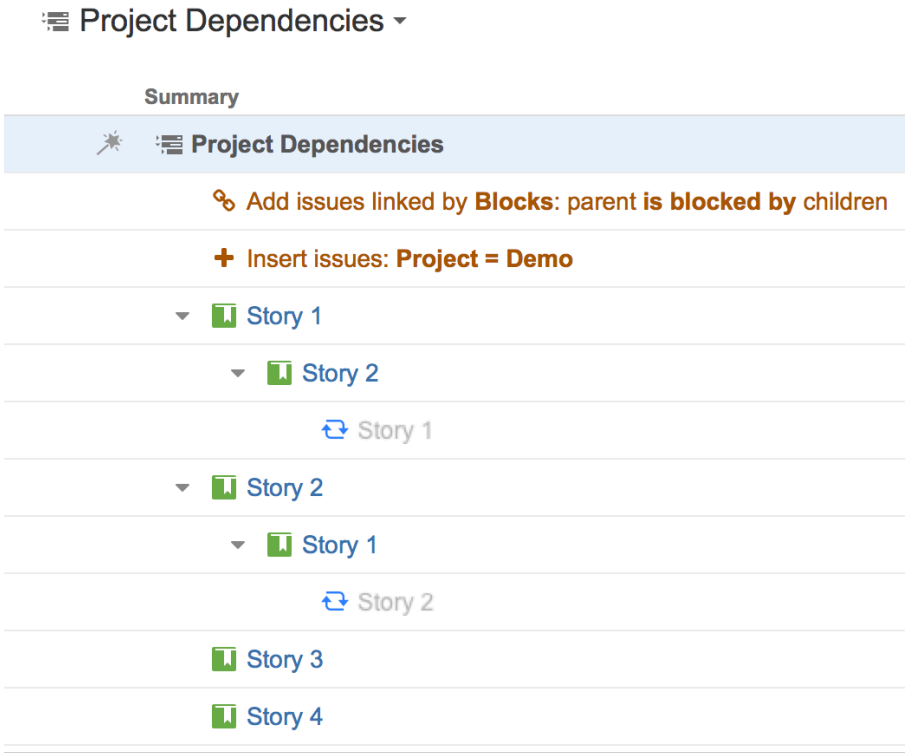


Example With Link Cycles

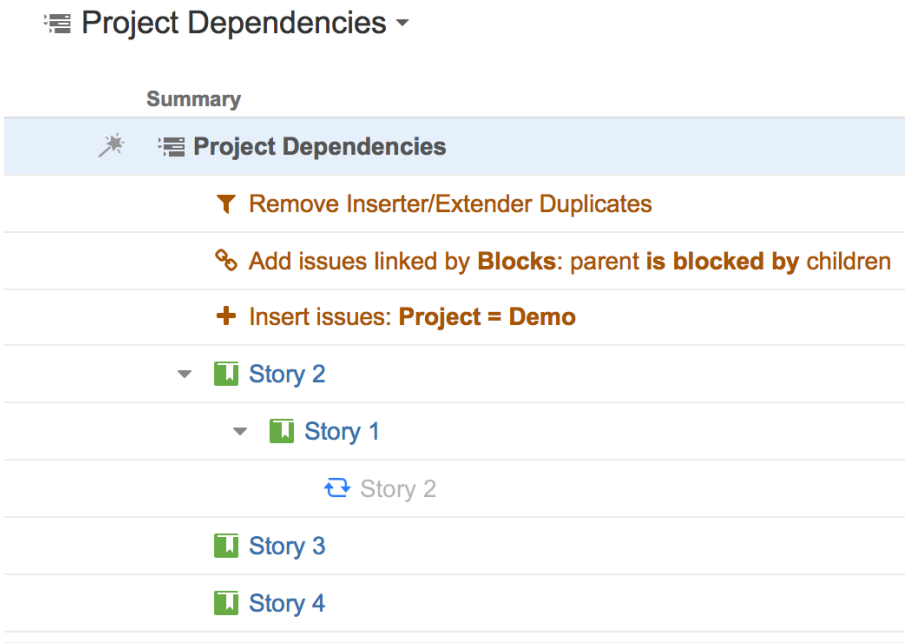
If there are link cycles between the issues, the Filter will remove one of the branches and will keep the other to make sure all the issues added by the inserter and extender are in the structure.

In this example Story 1 blocks Story 2 and Story 2 blocks Story 1.

Without the filter we get the following structure:



And as we add the Filter, one of the branches with the cycle gets removed:



Sort

Rank. If you use JIRA Agile and add sorting by Rank, Structure can update the Rank as a user moves issues up and down inside the Structure.

Manual. If you are working with a structure where issues were added by the **Insert Generator** based on some JQL query, by default you will not be able to re-order issues in such a structure. To be able to do that, you need to add the **Manual Sorting Generator**.

Group

Structure lets you group by most standard JIRA fields, custom fields provided by JIRA and other issue aspects:
Standard fields: Affects Version, Assignee, Component, Epic, Epic Status, Epic/Theme, Fix Version, Flagged, Issue Type, Labels, Priority, Project, Reporter, Resolution, Status, Sprint.
JIRA custom fields: fields that give you a list of values to choose from, such as radio button, list single choice, checkboxes, user picker, labels, select list.
Text attributes: built-in and custom text fields.
Portfolio parent link: you can group issues by their parent issue as defined in Portfolio for JIRA.

Tempo Account: you can group issues by Account custom field defined in Tempo for JIRA.
Issue links: you can group issues by their linked issues. You can select link type and direction, so that, for example, you can group issues under their respective blockers (issues that block them).

The fields that cannot be grouped by are dates and numbers.

Groups can be nested. For example you can take a list of issues and group them by fixVersion and then by Assignee. Thus you will see the existing fixVersions on the top level, Assignees on the second and then the issues themselves on the third, grouped accordingly. If you enable editing via Structure in the generator settings, Structure can update the fields by which the issues are grouped when you drag-and-drop issues from one group to another.

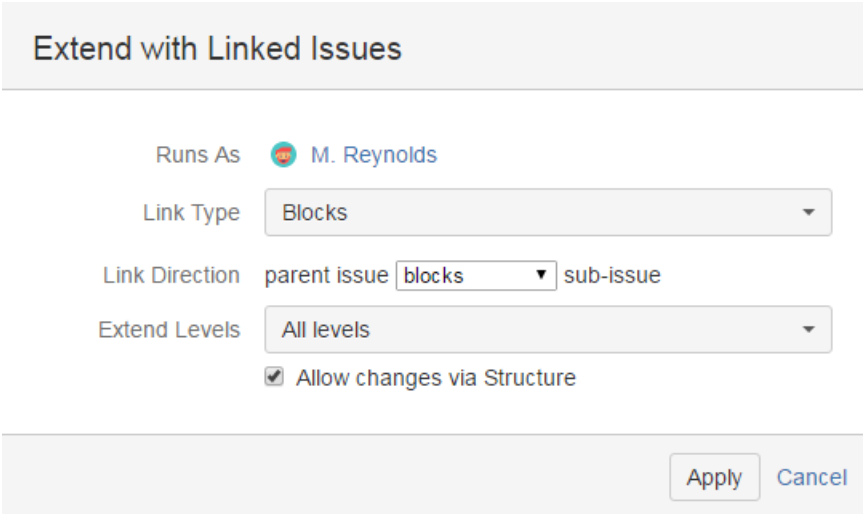
Extend

The Extend generator allows you to pull issues into the structure hierarchy based on Issue Links, Epic Links and Sub-task relationships.

Linked issues

The linked issues extender allows you to pull in issues that are linked to issues already in the structure.

Select the link type and direction from the drop-down lists.



You can also choose to limit the levels on which the linked issues extender acts.

- The default option is **All levels up to 10**, meaning the extender will extend issues at the first 10 levels of the hierarchy, starting from the level where the extender itself is located.
- The **All levels** option means that the extender will extend issues at any level in the hierarchy.
- The **Current level only** option restricts the extender to operating on issues at the same level in the hierarchy as the extender itself.
- The **Manual levels range** option allows you to specify a range of levels at which the extender should operate.

As with other generators, you can enable *editing via Structure* and the links will be updated as you move issues in your structure.

Stories under Epics

The Epic links extender pulls in stories belonging to epics already in your structure.

As with the links extender, you can choose the hierarchy levels at which the Epic links extender operates.

As with other generators, you can enable *editing via Structure* and the Epic links will be updated as you move stories in your structure.

Sub-tasks

The Sub-tasks extender pulls in sub-tasks belonging to issues already in your structure. You can choose the types of sub-task to be pulled in.

As with the links extender, you can choose the hierarchy levels at which the Sub-tasks extender operates.

As with other generators, you can enable *editing via Structure* and sub-tasks will be assigned to new parents as you move them in your structure.

Child issues from Portfolio for JIRA

The Portfolio extender pulls in child issues using the Parent Link field from the Portfolio for JIRA add-on.

As with the the other extenders, you can choose the hierarchy levels at which the Portfolio for JIRA extender operates.

As with other generators, you can enable *editing via Structure* and the Portfolio parent links will be updated as you move issues in your structure.

1.5.6 Generators Options

Even though the generators do different things, there are a couple of settings most of them share:

- Permissions to change JIRA data via changes in structure
- The scope of the generator (the levels where it works)

Editing Issues via Generators

When you use the **Group**, **Extend** or **Sort by Rank** generators, by default the structure will be able to update some JIRA data as you move issues in the structure:

- For the grouper it will update the field by which you group as you move an issue between the groups. For example, if you have issues grouped by assignee, as you move an issue from one group to another, it will be re-assigend.
- For the extender it will update the JIRA, Epic links or parent of a sub-task as you move a child from one parent to another. For example, if you move a story from one epic, under the other, the epic field of the story will be updated.
- For the sorter by rank it will update the Agile Rank as you move an issue up or down.

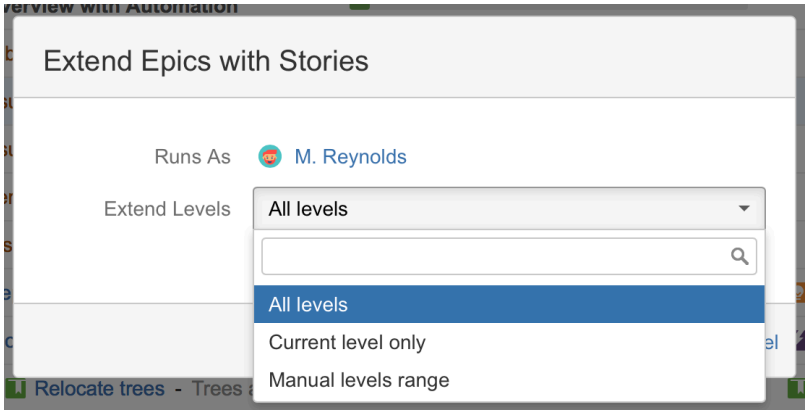
If you want to disable this, deselect the **Allow changes via Structure** checkbox in the generator settings.

Defining Generator Scope

The scope of the generators is defined by its position in the structure and the **Levels** option.

If you place the generator under the top-level root item, the generator will be applied to the whole structure. If you place it under some static item in the structure, the generator will only affect the descendants of this item.

To limit the scope further you can set on which levels the generator should run.



- **All levels** means it will be applied to all descendants of the parent item.
- **Current level only** means it will be applied on the level, where the generator is added.
- **Manual levels range** allows you to define the specific levels, where the generator should work.

The levels created by Group generators are not taken into account when applying the specified manual levels limitation.

The From and To fields define the range of levels to which the generator will be applied. For example, if you want to pull in issue linked to the issues on level 2, you need to set the **From** field to 2 and **To** to 2, not 3. Setting the value to 3 will make extender add issues linked to the ones on level 2 and add the issues linked then to the ones that were just added.

This option is especially useful when you are defining generators for the hierarchy, where on different levels you have different types of relations. For example, the top level and the 2nd are connected with issue links, the 2nd to the 3rd with epic links, and on the 4th level you have sub-tasks.

In this case you will have three generators added under the root of the structure with the following **Levels** settings:

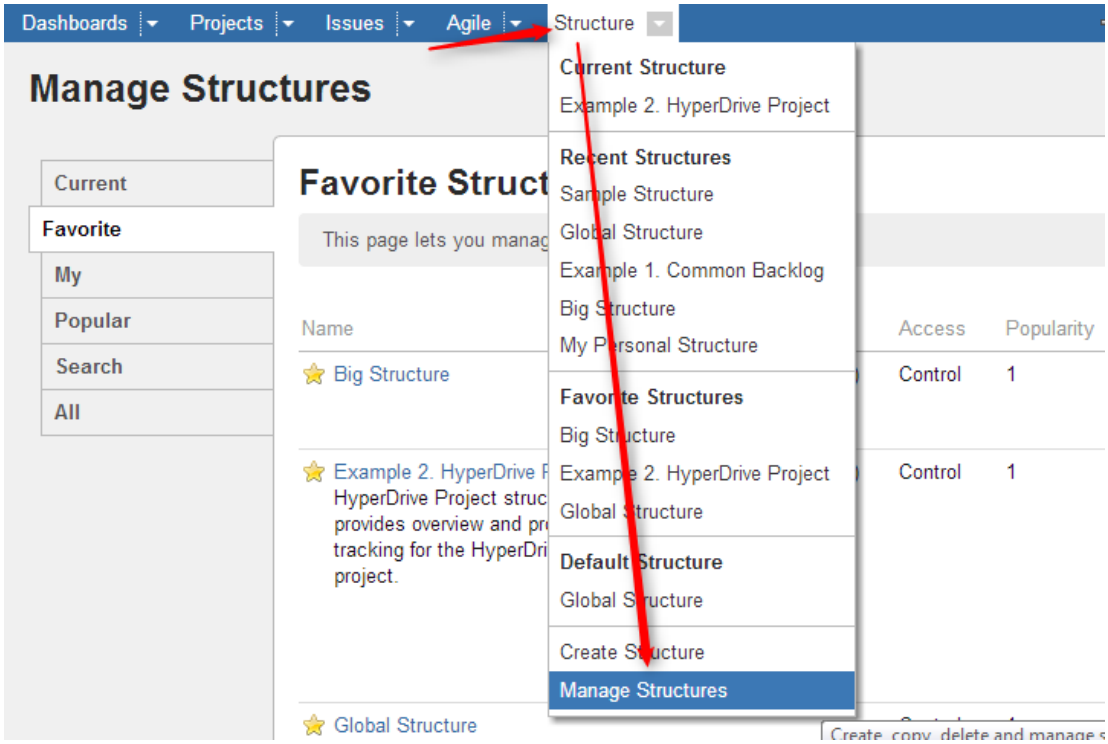
1. Links extender (**Linked Issues**) working only on the top level - **Current level only**
2. Agile extender (**Stories under Epics**) working on the second level - **Manual levels range**: from 2 to 2

3. Sub-tasks extender working on the third level - **Manual levels range**: from 3 to 3

1.6 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:

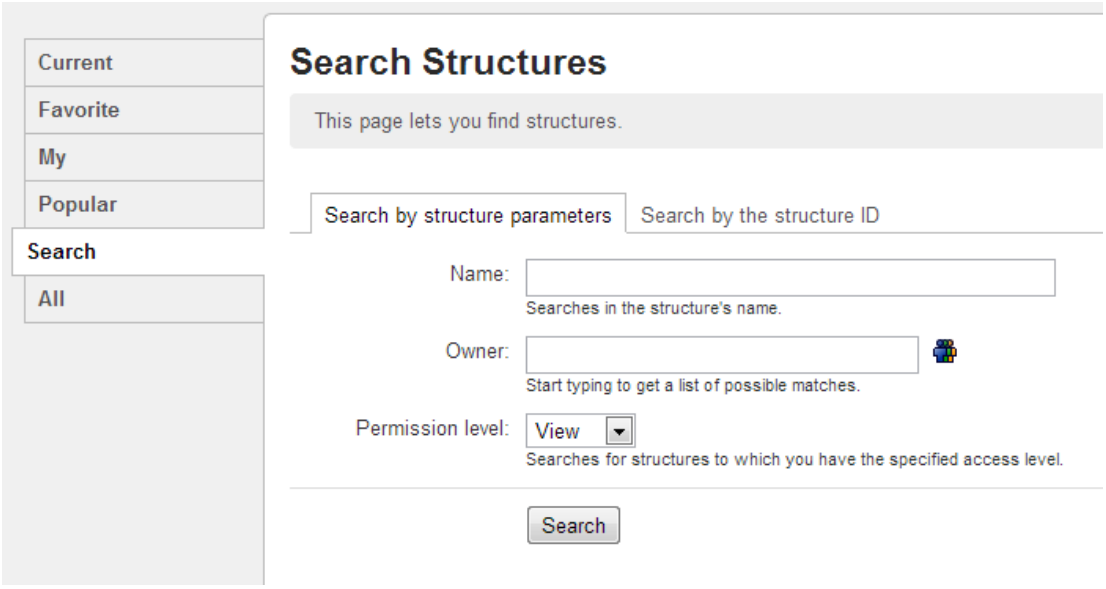
- **Current** – shows only the structure you're currently working with.
- **Favorite** – lists structures that you have marked as your favorite (see page 12).
- **My** – lists structures created by you.
- **Popular** – lists structures that are marked as favorite by at least 2 users, ordered by their popularity (see page 12).
- **Search** – allows you to find structure by name, owner or ID (see page 191).
- **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:

1.6.1 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

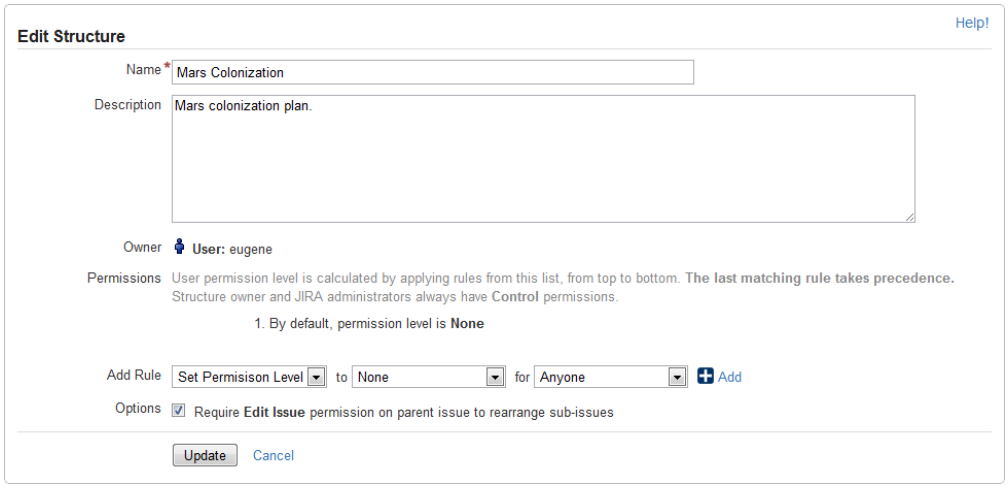
To perform a search by structure's numeric ID:

- 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.

1.6.2 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 owner of the structure. Only JIRA administrators can change the owner.
Permissions	Define who can view, edit or configure the structure. See Structure Permissions (see page 194) 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 the user is allowed to make. See Structure Permissions (see page 194) for details.



You can specify structure details when Creating New Structures (see page 193) and when Editing Structure Details (see page 193).

Editing Structure Details

To edit details (see page 192) 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.

1.6.3 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 (see page 192) 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 (see page 194).

Only logged in users who have access to the Structure Plugin are allowed to create new structures. See Who Has Access to the Structure (see page 306).

1.6.4 Structure Permissions

Every structure has a list of permission rules, which define 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. The user cannot, however, create or modify Generators (see page 178).
Edit Generators	The user has full edit access to the structure, including modifying generators.
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 (see page 193).

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 (see page 192) 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 (see page 194)). 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.

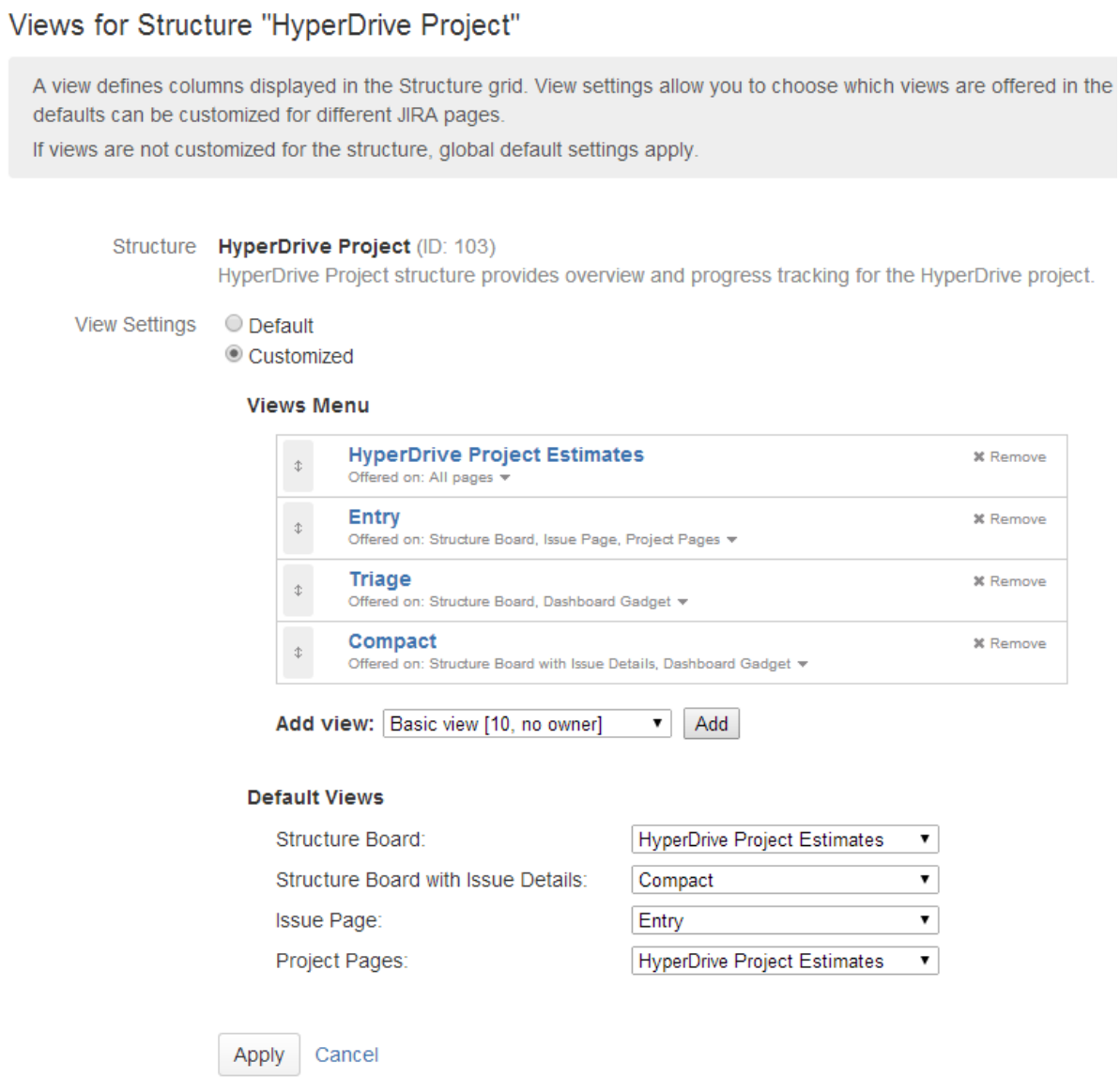
1.6.5 Customizing View Settings

A structure's view settings determine which views are offered to the users in the Views Menu (see page 42) 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.



View Settings page

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 (see page 42) 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, Structure Board with Issue Details, 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.

1.6.6 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 a part of a structure, create a new empty structure and use Issue Clipboard (see page 148) to copy a part of the structure.

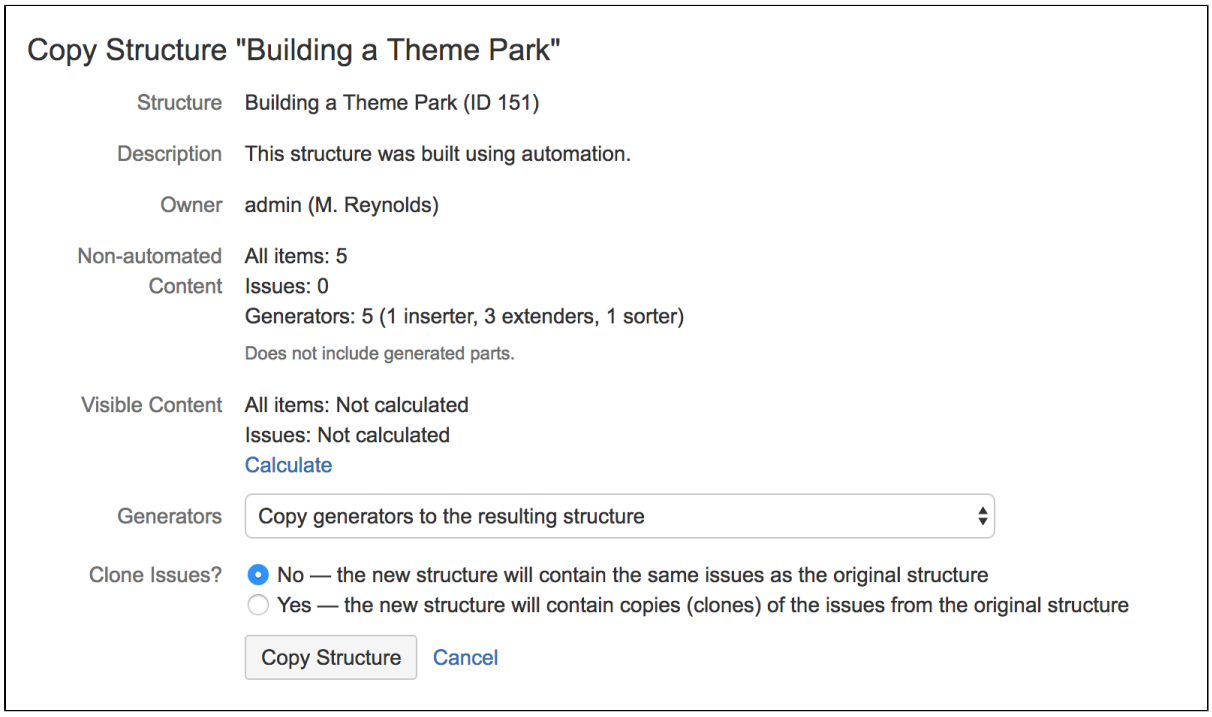
To create structure copy:

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.

The **Copy Structure** page will show you the information about the structure, including its size and the number of issues, generators, and synchronizers it contains. If the structure contains automation, you can click the **Calculate** link in the **Visible Content** section to execute the generators and see the generated content statistics.

3. Choose how to handle generators and generated content, if the structure has any. The following options are available:
   - Copy generators to the resulting structure – the new structure will contain copies of the generators from the original structure, which will generate the same content. This is the default.
   - Replace all generators with the generated content – the new structure will contain a snapshot of the original structure at the time of copying, with no automation installed.
   - Do not copy generators – the new structure will contain only the non-automated content from the original structure. The generators and generated parts will not be copied.
4. Choose if you'd like to copy synchronizers, if the structure has any. If you don't see **Copy Synchronizers** option, then you probably don't have a permission to create synchronizers. See Copying Synchronizers (see page 206) for more details.
5. Choose if you'd like to clone issues.
6. When cloning issues, enter additional parameters for the cloning process (see page 201).
7. 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 (see page 201).

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 (see page 194).

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

Copying Structure and Cloning Issues

When copying a structure (see page 198), 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.

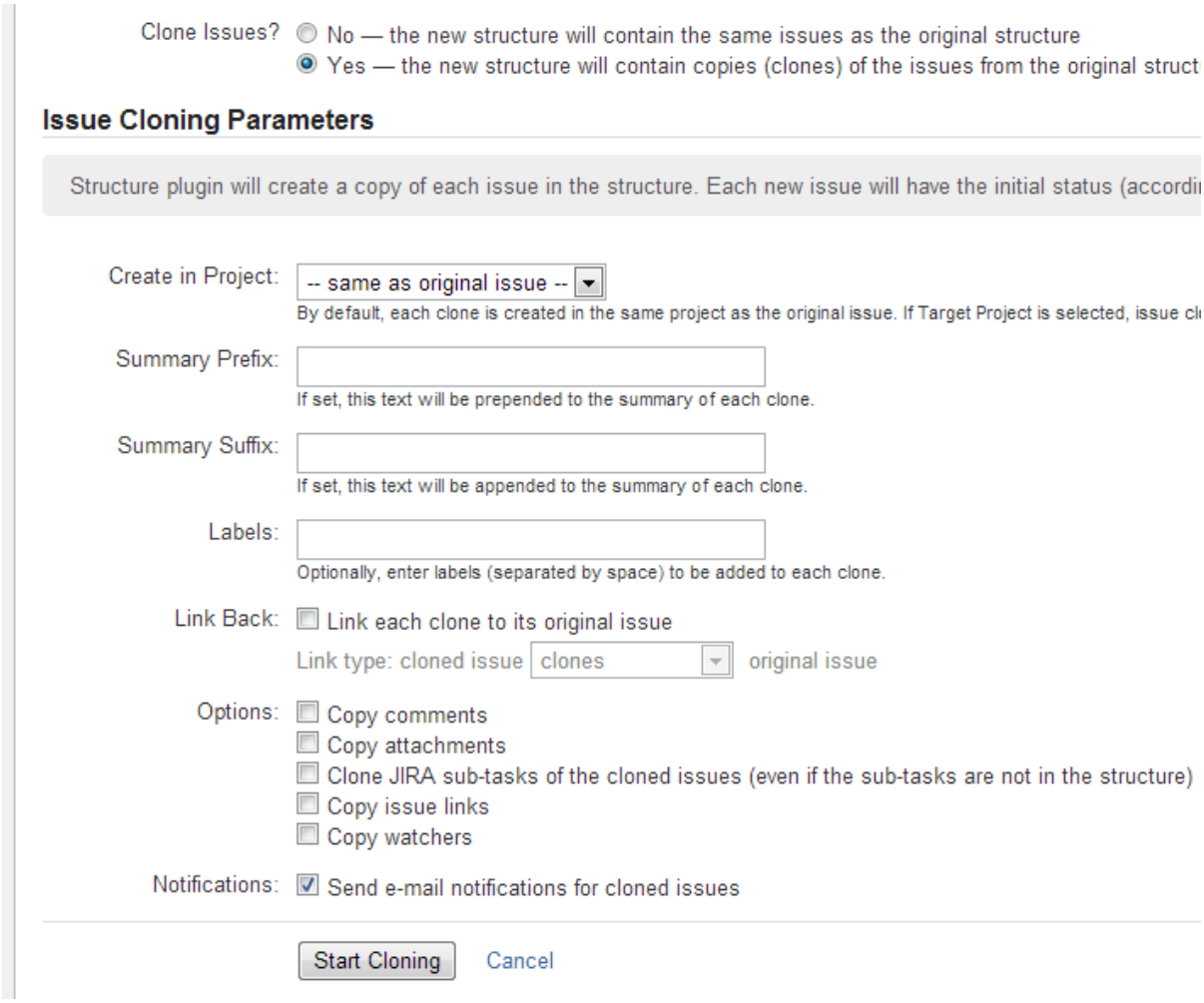
Cloning issues to a different project may even be impossible, for example, when a certain field is required in the target project, but absent (or not required) in the source project. If this is the case, you will need to either change the target project restrictions or make sure that every issue in the 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 (see page 307).)
- 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.

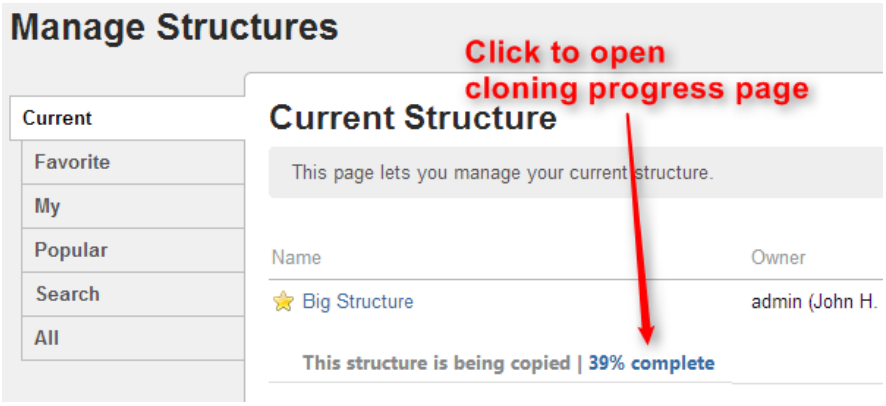
At step 2 the cloner process might discover some problems. If critical problems are discovered, an error message is shown and process is aborted. If non-critical problems are discovered, then warnings are shown and user input is required. The warnings may suggest that cloning may continue, but the resulting 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 (see page 164) to quickly delete the unwanted issues.

Cloning Queue

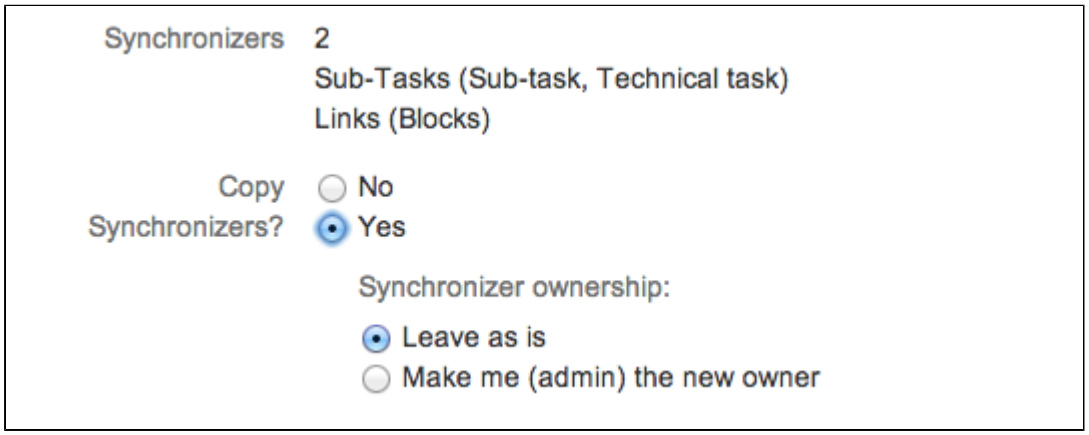
Cloning issues can place considerable load on JIRA server. To avoid overloading server with cloning jobs, there is a limit to the number of cloning processes that can happen simultaneously. If this limit is exceeded, your cloning process will initially be in "waiting" state, pending for other cloning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

Copying Synchronizers

When copying a structure that has synchronizers, you can use the **Copy Synchronizers** option to make Structure plugin create a copy of every synchronizer installed in the original structure.

> ⚠ If you don't see the **Copy Synchronizers** option, then you probably have no permission to create synchronizers.

Synchronizers Copying Parameters



You can decide to leave the original ownership of a synchronizer ("**Run As**" parameter) or make yourself a new owner for each of the copied synchronizers.

> ⚠ Only JIRA administrators can change synchronizer ownership.

> ⚠ Making yourself the new owner means that all synchronizers in the copied structure will run under your account.

Required Permissions

To be able to copy synchronizers you need a permission to create and configure synchronizers (see page 308).

Copied Synchronizers

When structure copying is complete, all of the copied synchronizers become disabled until you run **Resync & Enable** manually. To use them, you need to review their configuration, adjust if necessary, and run **Resync & Enable**.

1.6.7 Archiving a Structure

With the **Archive** action you can make a structure read-only and hide it from search results and menus (including structure selector on the Issue Page (see page 17)).

Read-only means that users cannot add, remove or move issues in the archived structure.

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

To archive a structure:

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

   You need **Control** access level to be able to archive a structure.

3. Review the structure you are about to archive and confirm the operation. You can **Unarchive** the structure in future.

If there are any synchronizers installed for the structure you archive, they will be disabled.

Unarchiving Structure

You can **Unarchive** the archived structure to make it editable and visible in all menus.

To unarchive a structure:

1. Open **Manage Structures** page using the top navigation **Structure** menu.

2. Select **Archived** tab on the left side or search for structures on the **Search** tab with an option **Show Archived** checked.
3. Find the structure you'd like to unarchive and click **Unarchive** link in the Operations column.

> You need **Control** access level to be able to unarchive a structure.

> If there are any synchronizers installed for the structure you unarchive, you probably need to review the synchronizers configuration and maybe resync and enable them.

Searching for Archived structures

You can find an archived structure on some tabs of **Manage Structures** page:

- On the **Archived** tab.
- On the **Favorite** tab if your favorites list contains any of archived structures.
- On the **Search** tab when searching for structures by structure parameters with the option **Show Archived** checked.
- On the **Search** tab when searching for structures by the structure ID.

Synchronizers

If there are any synchronizers installed for the structure you archive, they will be disabled. After unarchiving you will probably need to review the synchronizers configuration and maybe resync and enable them.

Until the structure is unarchived you cannot resync and enable synchronizers.

Nevertheless, you can Export (see page 221) an archived structure if you have a special permission to control synchronizers.

1.6.8 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 (see page 235)), 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.

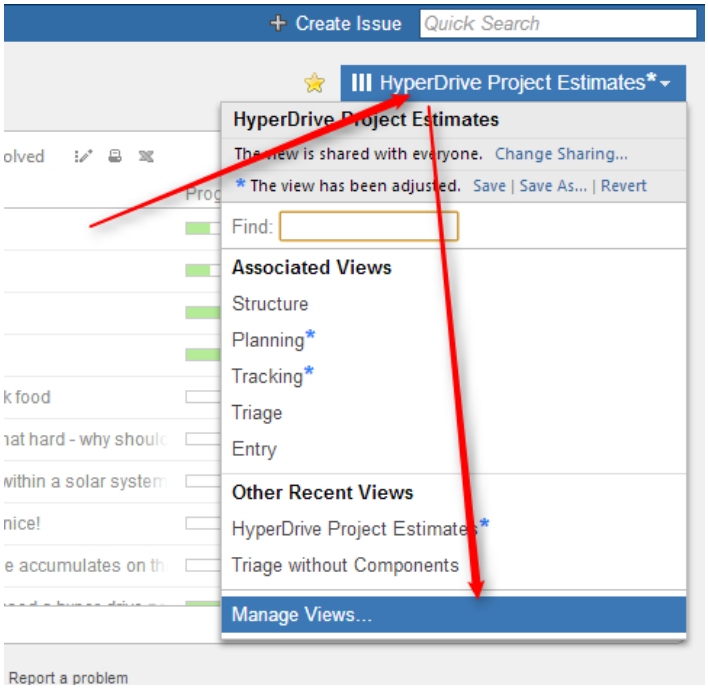
   > ⓘ You need **Control** access level to be able to delete a structure. See Structure Permissions (see page 194).

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

1.7 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 (see page 42). You can also save changes, create a new view and share a view (see page 46) in the same Views drop-down. For other operations with the views, you need to open **Manage Views** dialog:

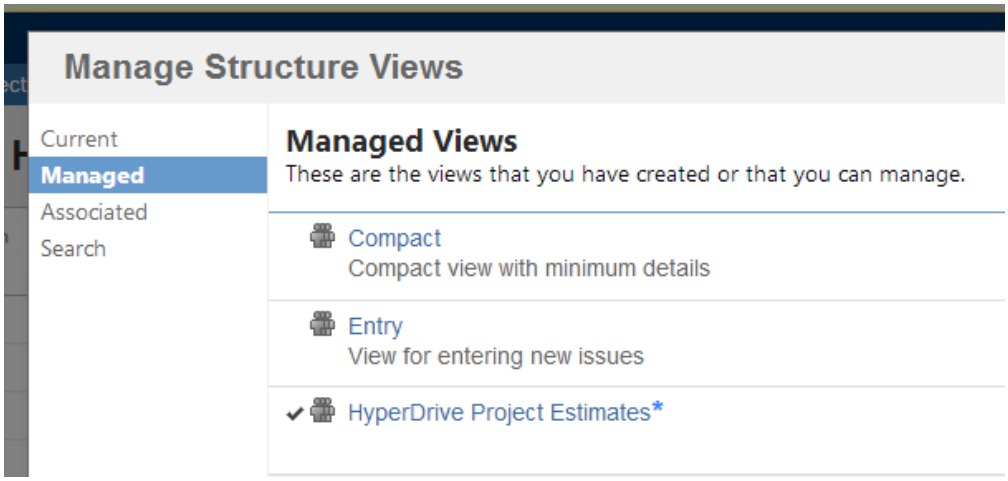


See the following sections for details on view management:

1.7.1 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 (see page 42), and find a view by name. You can also open **Manage Views** dialog and find the view on one of its tabs:



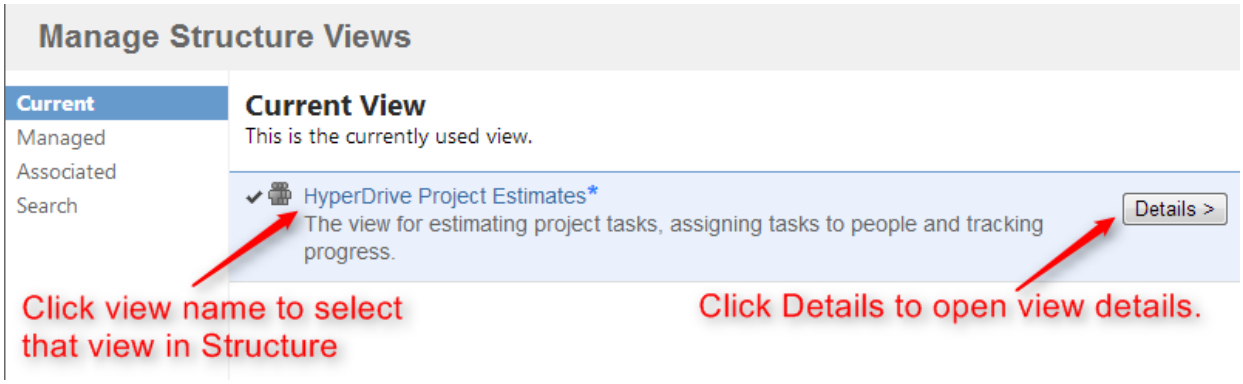
The following tabs are present:

- **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.

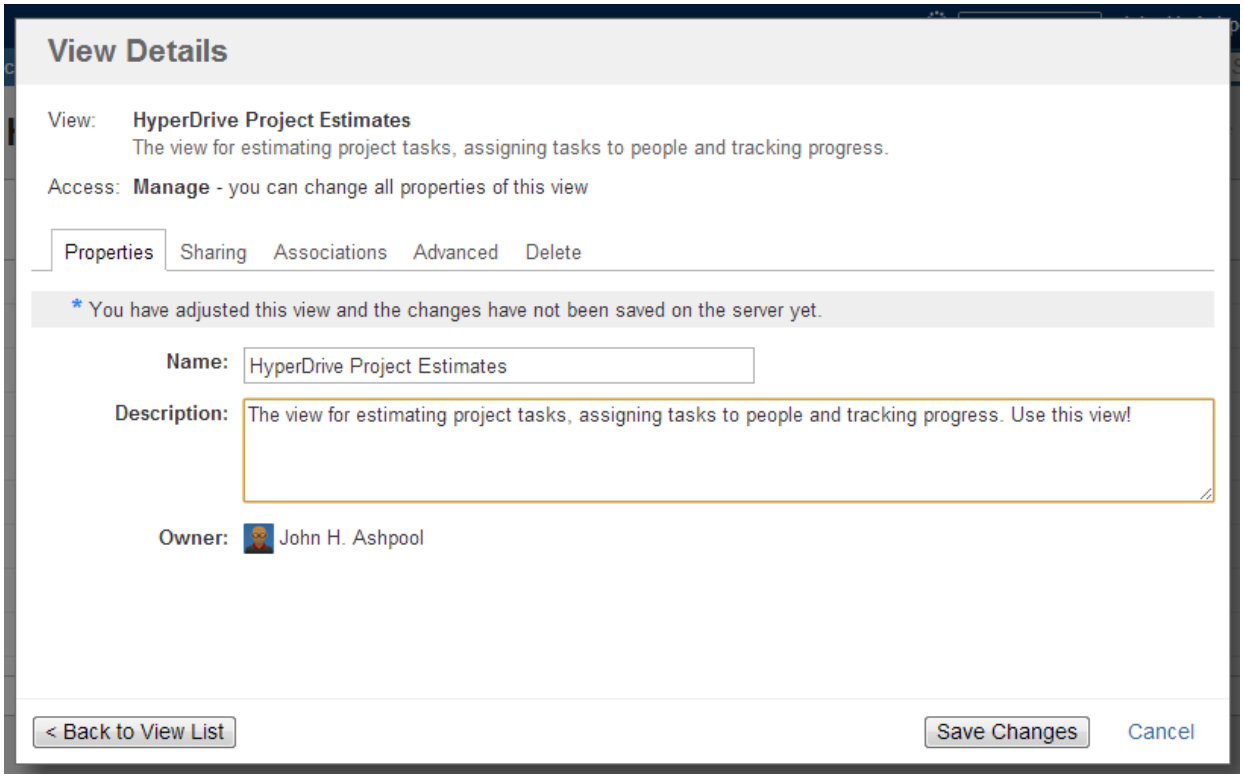
When you have located the view, you can click its name to switch to that view in the Structure widget. The view will also appear in **Also Recently Used** section of the Views Menu (see page 42).

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



1.7.2 Changing View Settings

When you have located a view (see page 210) 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 (see page 213).
- **Associations** tab shows structures, which are associated with the view (have this view in their Views drop-down). See Associating Views with Structures (see page 215).
- **Advanced** tab shows some technical information about the view.
- **Delete** tab lets you delete this view (see page 215).

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 (see page 190) page.

1.7.3 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:

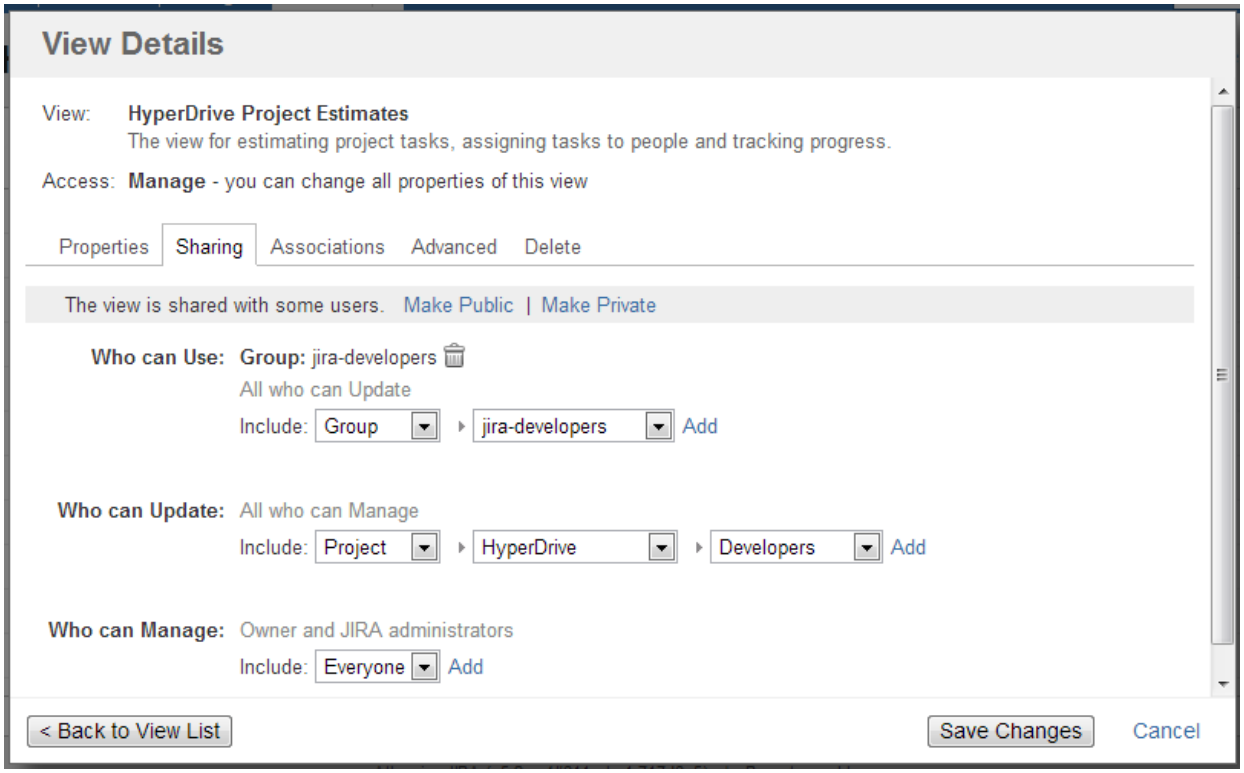
None	The view is not visible nor usable by the user.
Use	Read-only access: the user can use the view, but cannot modify it.
Update	The user can use the view, and also save view adjustments as the new version of the view. The user cannot modify view name or sharing permissions.
Manage	The user can change 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 (see page 43), 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 (see page 42) – 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.

1.7.4 Associating Views with Structures

Views, which are associated with a structure, can be easily accessed from the Views Menu (see page 42) 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 (see page 196) for details. Also, JIRA administrator may specify global default view settings (see page 312), 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.

1.7.5 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 (see page 42) or Manage Views dialog (see page 210) 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 (see page 46) link to create a new view.
4. Use **Manage Views** dialog to review the new view's description and sharing permissions.

1.7.6 Deleting a View

To delete a view:

1. Open **Manage Views** dialog.
2. Locate a view (see page 210) 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.

1.8 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.

1.8.1 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 (see page 252).

1.8.2 Creating Issues and a Structure from Template

Once you have a template structure, you can use Copy (see page 198) action from the **Manage Structures** page and turn on Clone Issues (see page 201) option. For details about configuring and running cloning operation, refer to Copying Structure and Cloning Issues (see page 201) 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 (see page 164) on all issues - for example, to remove a template marker.

1.8.3 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:

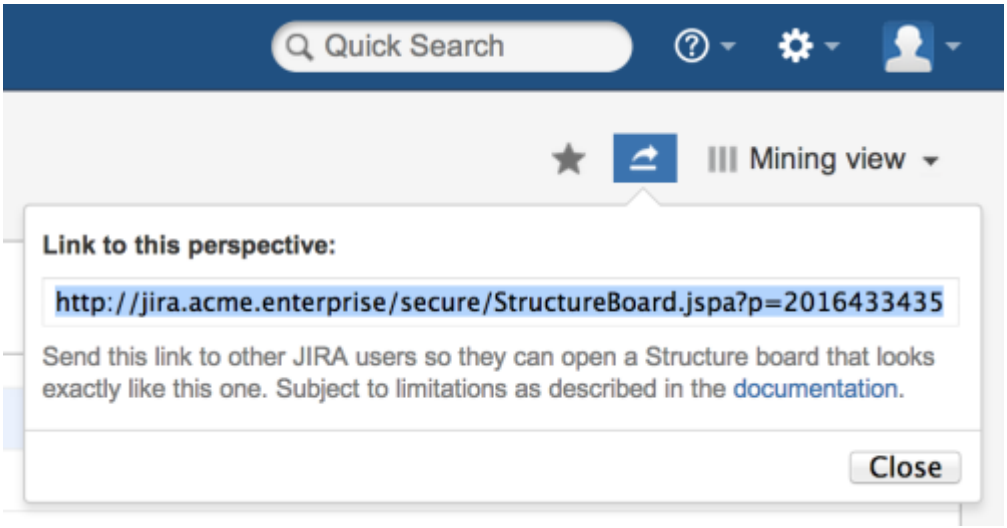
1. Manually create an empty new project.
2. Create new structure and issues from template structure, as advised above. When configuring cloning parameters, specify the new project in the **Create in Project** parameter.

1.9 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 (see page 15)
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 (see page 209) 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 (see page 318).
5. Once created, a perspective becomes accessible to any person who has access to the structure for which that perspective was created.

1.10 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 (see page 230) places JIRA sub-tasks under their parent issues in the structure
- Links Synchronizer (see page 235) 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 (see page 231) 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 (see page 239) 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 (see page 244) 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 (see page 221) or Import (see page 220), or when you run a Resync (see page 226). 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.

1.10.1 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 (see page 231) to add All Open Issues to a structure (or the results of whatever Saved Filter you have), or JIRA Agile (GreenHopper) Synchronizer (see page 239) 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 (see page 229). 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 possibly 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 (see page 220) and Export (see page 221) are actually a one-time Resync (see page 226). 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!

1.10.2 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 (see page 235) 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 (see page 229). 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 (see page 220) and Export (see page 221) are actually a one-time Resync (see page 226). 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!

1.10.3 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 (see page 235), 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 (see page 231) 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 (see page 224) 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 (see page 229).

   >

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 (see page 220) and Export (see page 221) are actually a one-time Resync (see page 226). 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!

1.10.4 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 (see page 226) 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.

1.10.5 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.

1.10.6 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 (see page 226) 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 (see page 226) and enable immediately after resync finishes.

1.10.7 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 (see page 221).
- *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 (see page 220).

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.

1.10.8 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 (see page 194) 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.

1.10.9 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 (see page 230) can be configured to put a sub-task under an issue, while a links synchronizer (see page 235) 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 (see page 327).

1.10.10 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.

> 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 (see page 226)*).

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 update 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 (see page 226) 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 issues are placed	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.

This synchronizer supports only Import / Resync into Structure (*more about resync (see page 226)*).

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 (see page 305).

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.

Automatic Branches Removal

The “Double-check sub-issues” option is useful when you want to build and, more importantly, maintain a Structure, where you have a certain set of issues on the top level and then all the issues that are linked to them added under them.

Specifically the Double-Check option is necessary for removing from the structure the top level issues and all the issues linked to them, when the top level issue no longer passes the filter of the synchronizer.

Here is an example.

You are trying to build a structure, where you have all Open Stories on the top level and then the issues which block them added below.

To build this structure you will need to configure the Filter synchroniser, which will add Open Stories to the top level and the Links synchroniser, which will add linked issues.

You can find more information on Link Synchroniser in this article.

To get the list of the top level issues you can use the JQL query, which looks like this:

```
issuetype = Story and status = open
```

However, if you use this as the filter query and select the remove option, the Filter Synchroniser will remove all children, which will be added by the Link Synchronizer, because the children do not pass the JQL query.

To solve this problem you can extend the query with S-JQL expression, which returns both parents and their children, which are already in the structure - this will prevent the Filter Synchronizer from removing children from the Structure:

```
(issuetype = Story and status = Open) or issue in structure(“Open
Stories Structure“, "issueOrAncestor in [type = Story and status
= Open]”)
```

For more information on S-JQL please refer to the documentation.

Now the last step is the removal of the Story and issues linked to it when the status of the story changes. If the Double-Check option is not selected, once the Story status changes the synchronizer will see, that the Story should be removed, but will think that the children still pass the filter (because there were no explicit changes done to them). As a result it will keep both the Story and the children in the Structure. Selecting the Double-Check option will force it to check if the children still pass the filter and it will remove the whole branch.

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 (see page 226)*). 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 (see page 237)).

No matter how synchronizers are configured, they will only affect issues from the projects that are enabled for synchronization (see page 305).

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 (see page 238), 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

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 mode (GreenHopper/JIRA Agile 6.1+ only) parameters:
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	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 (see page 230).

This synchronizer supports both Import and Export / Resync into/from Structure (*more about resync (see page 226)*). 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

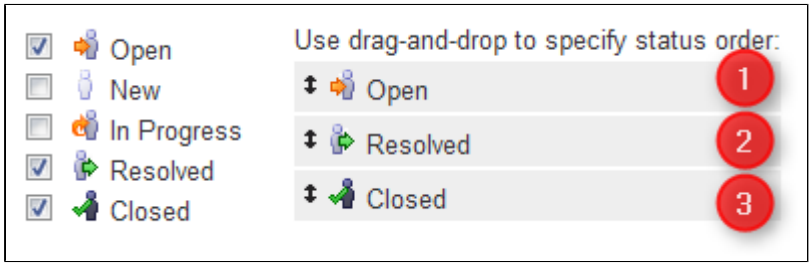
Enabled Projects	Only issues belonging to the selected projects are **changed**. It does not matter what project sub-issues belong to, as long as their parent belongs to the enabled project — every sub-issue counts with its status.
Enabled Issue Types	Same with types — you can select issues of which 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

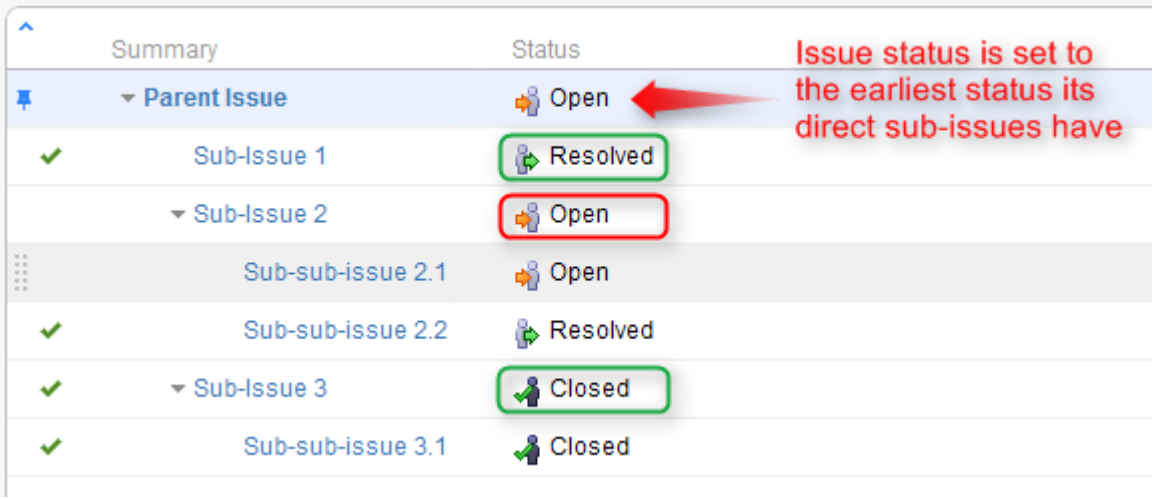
The synchronizer tracks updates to issues and to structure, and tries to make sure that the status of the parent issue corresponds to the aggregate status of its direct children.

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.



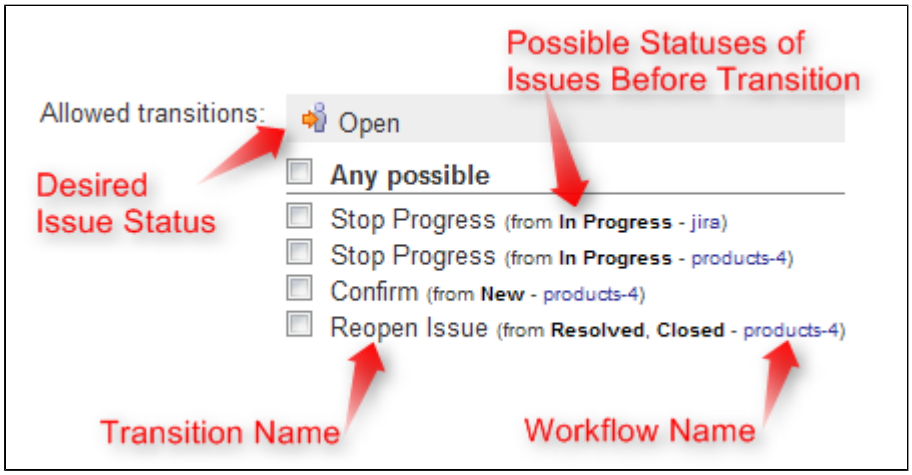
On the screenshot above:

- 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.

1.11 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.

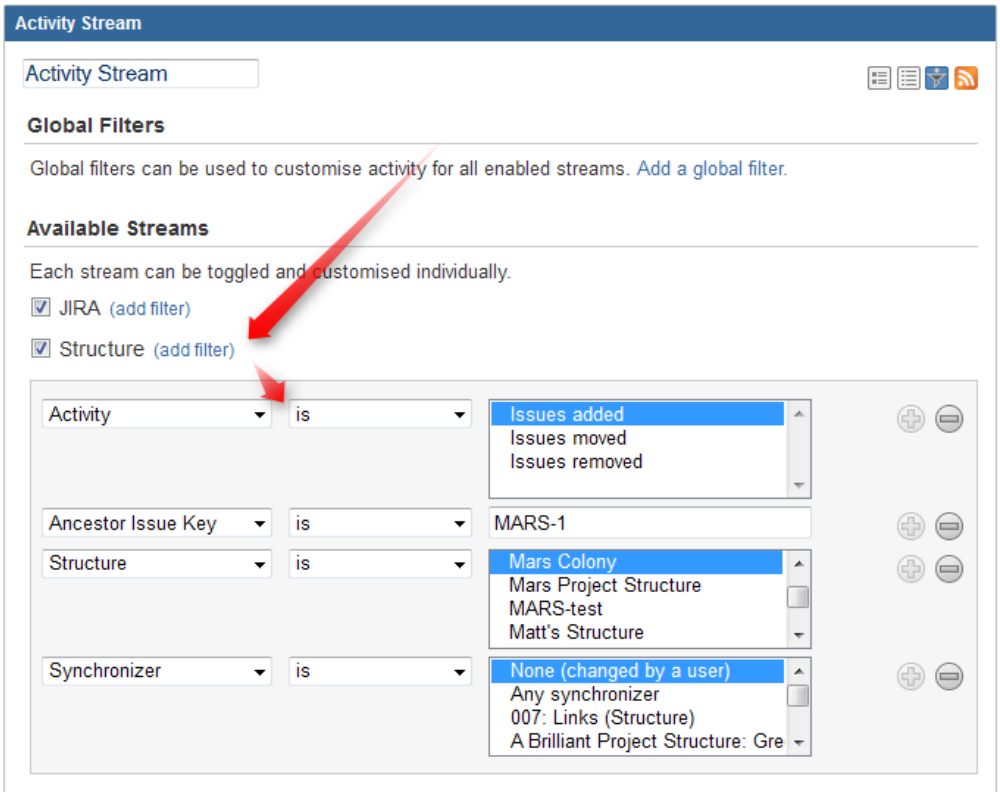
To activate the Structure stream, select the Structure option in the Available Streams section of the Activity Stream gadget configuration.

1.11.1 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**.



1.11.2 Reading Activity Stream

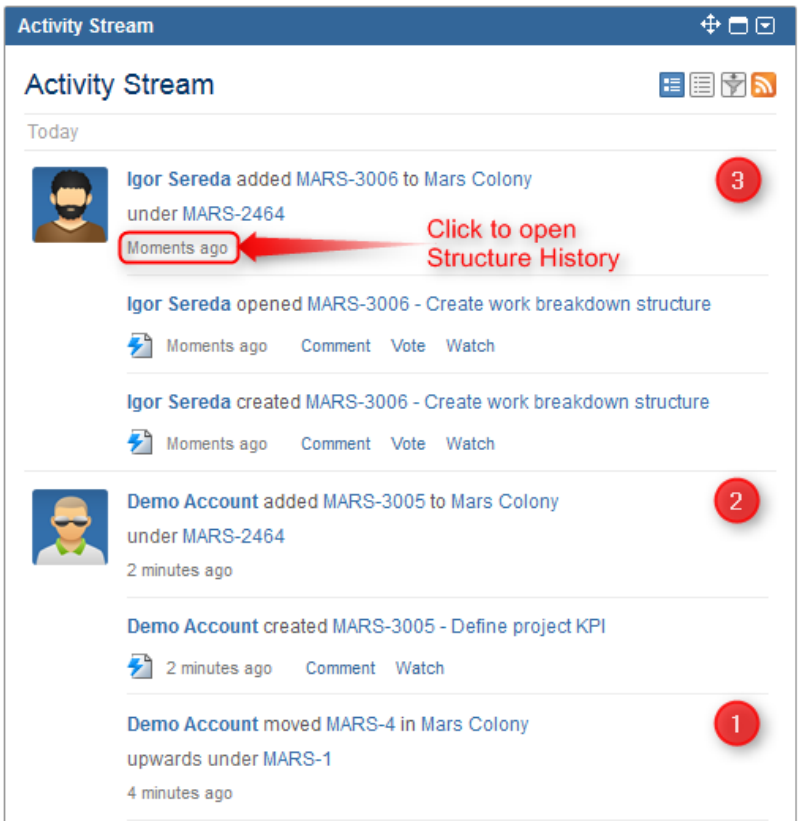
Changes in the Structure activity stream are ordered chronologically, newest first. For each change a short summary is displayed, containing:

- 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 (see page 166).

1.11.3 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.

1.12 Structured JQL

Structure not only displays hierarchy of items, it allows to search items based on their relative positions in the hierarchy. The language used to express such queries is called **Structured JQL** or **S-JQL** (where JQL stands for JIRA Query Language).Structured JQL queries are available in these places:

- inside `structure()` JQL function — that allows to search for issues in structure on the Issue Navigator, see structure() JQL function (see page 282);
- in the Search Area on the Structure Widget — see Simple, JQL, and S-JQL Search (see page 127);
- in the S-JQL Filter generator (see page 179);
- in the workflow validator or condition — see Workflow Integration (see page 321).

To quickly find a solution to a common structure querying problem, consult S-JQL Cookbook (see page 252), 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 (see page 258) for a comprehensive description of the language and `structure()` JQL function.

1.12.1 S-JQL Cookbook

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

1. Find issues added to a structure

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

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

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

If you want to find issues added to the Default Structure (see page 12), you can omit the structure name:

```
issue in structure()
```

^ up to the list of examples (see page 252)

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

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

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

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

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

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

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

^ up to the list of examples (see page 252)

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

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

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

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

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

If you want to see these issues in the structure, go to Structure Board (see page 15) and type this query in the Search Area (see page 126) in the JQL mode.

Also, you can type only the last part of the query if you use S-JQL search mode (see page 129) :

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

^ up to the list of examples (see page 252)

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

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

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

How to achieve: You can use Issue Navigator's Advanced Searching capability or open the structure on the Structure Board (see page 15) and use its Search Area (see page 126) 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 (see page 129) on the Structure Board:

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

^ up to the list of examples (see page 252)

5. Find all issues that are blocking critical issues

Setup: Suppose that you have a structure named "Dependency structure" where parent-child relationship corresponds to dependency: each child blocks its parent. (You might have configured a Links Synchronizer (see page 235) 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 (see page 15) and use its Search Area (see page 126) 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 (see page 129) on the Structure Board:

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

^ up to the list of examples (see page 252)

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

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

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

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

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

^ up to the list of examples (see page 252)

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

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

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

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

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

^ up to the list of examples (see page 252)

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

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

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

How to achieve: In the Search Area (see page 126) on the Structure Board (see page 15), run the following JQL search (see page 129):

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

^ up to the list of examples (see page 252)

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

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

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

How to achieve: In the Search Area (see page 126) on the Structure Board (see page 15), run the following S-JQL search (see page 129):

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

^ up to the list of examples (see page 252)

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

Setup: Suppose that "Planning" is a work breakdown structure. Your team follows the convention that once all children of an issue are resolved, the issue can be resolved as well. The best solution for this would be to use a Status Rollup Synchronizer (see page 244), 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 (see page 15). When you paste the query given below into the Search Area (see page 126) (ensure that the JQL mode (see page 128) 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 (see page 252)

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

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

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

How to achieve: use the Search Area (see page 126) on the Structure Board (see page 15) to run the specific queries (ensure that the S-JQL mode (see page 129) is selected.) Toggle the Filter (see page) 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 (see page 252)

12. Get the contents of a folder

Setup: There is a structure with a folder named "Next Release". Issues are placed there manually and then queried via S-JQL for planning purposes (as an Agile board filter, for example).

Goal: The users want to see all issues that are located under the specified folder.

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

```
issue in structure("My Structure", "descendant of folder('next
release')")
```

Note that the folder name is case-insensitive.

^ up to the list of examples (see page 252)

1.12.2 S-JQL Reference

Structure query is a hierarchical condition on the items added to the structure. Structure query is expressed in the Structured JQL language (S-JQL), described in this article.

> Parts of this article assume that you are familiar with Advanced Searching capability of JIRA.

List of Structured JQL topics:

Multiple instances of items

If there are multiple instances of an item in the structure, some of these instances might match the query, and some might not.

Consider the following structure:

```
TS-239
  TS-42
TS-123
  TS-239
```

Here, issue TS-239 is present two times — one at the root position, and another under another issue. Query `root` will match the first instance but not the second one.

This difference is visible when you are filtering in the Structure Widget (see Filtering (see page 129)). However, structure() JQL function (see page 282) matches an issue if *at least one* of its instances in the structure matches the S-JQL query. In this example, `issue in structure(root)` will return `TS-239, TS-123.`

Constraints

Structure query consists of *constraints*. A constraint matches items 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 (see page 258)

Basic constraint

A basic constraint matches items that satisfy a condition — regardless of their relative positions to other items.

JQL constraint

JQL constraint matches all issues in the structure that satisfy a JQL query. To specify it, specify the JQL query enclosed in square brackets:

```
[status = Open]
```

leaf and root

This basic constraint matches items that are located at special positions within the structure.

```
leaf
```

```
root
```

The first constraint matches items at the bottom level of the hierarchy, i.e., items that do not have children (sub-items).
The second constraint matches items at the top level of the hierarchy, i.e., items that do not have a parent.

Specific issue

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.

You can specify a comma-separated list of issue keys:

```
TS-129, TS-239
```

One issue key:

```
TS-129
```

Issue ID (or a list of them):

```
19320
```

Function constraint (folder, item)

Functions in S-JQL play the same role as in JQL: it is an extension point, so any vendor can develop their own functions to match items in a custom way.

Structure comes bundled with a few functions: *folder* (matching all folders or folders by name) and *item* (matching all items of the specified type or items by name).

Syntax

A function constraint has a *name* and zero or more *arguments*, depending on the function you are using:

```
folder(Urgent)
```

In the example above, function name is *folder* and its argument is *Urgent.*

You can insert any amount of spaces around the name and arguments:

```
folder ( Urgent )
```

Multiple function arguments should be separated by commas:

```
item(Status, In Progress)
```

If an argument contains commas or parentheses, you need to enclose it in "double quotes" or 'single quotes':

```
item(Status, "Done, Sealed, and Delivered")
folder("NU (non-urgent) issues")
```

The former example matches Status items in structure that are named *Done, Sealed, and Delivered.* If this name wasn't enclosed in quotes, the query would mean that function *item* is given four arguments: *Status*, *Done*, *Sealed* and *and Delivered*.
The latter example matches folders named *NU (non-urgent) issues.* If quotes were not used, the query would be incorrect because the first closing parenthesis would be understood as the end of *folder*'s arguments.

If your argument contains quotes, you need to use another type of quotes to enclose it. Suppose that you need to match a version named *3.0, 3.0.1 "Armageddon"*:

```
item(version, '3.0, 3.0.1 "Armageddon"')
```

You can also escape the quotes using backslash (\). Suppose that the version is named *3.0 Beta 1 "Armageddon's Near":*

```
item(version, '3.0 Beta 1 "Armageddon\'s Near"')
```

If you need to use backslash character on its own, you can escape it with another backslash (\\). Suppose that you need to match a folder named *\ (backslash) and related characters*:

```
folder ('\\ (backslash) and related characters')
```

Note that if you don't need to enclose your argument in quotes, then you don't need to escape quotes or backslashes contained within it:

```
folder (Joe's)
folder ( \ )
```

Finally, if there's only one argument and the argument doesn't contain spaces (or is enclosed in quotes), you can omit the parentheses:

```
folder Urgent
folder "Not urgent"
```

folder()

This function matches folder items in the structure, optionally filtering them by name.

Without arguments, this function matches all folders:

```
folder()
```

With one argument, this function matches folders by name (that you see in the *Summary* column). A folder is matched if its name *starts with* the text specified in the first argument. Difference between capital and small letters is ignored.

For example, the following queries match folders named `My issues`, `Issues for Carol`, and `Non-issues`; and do not match folders named `Is suing` or `Issuance:`

```
```

```
folder issue
folder Issue
```

If you specify several words separated by spaces, `folder` will match only folders containing all of these words.

If you're familiar with how Simple Search in structure (see page) works, then it's useful to think of this argument in the same way as of the simple query. The only difference is that `folder` doesn't recognize issue keys.

There's an advanced matching option for those who like to use *regular expressions.*

To tell `folder` that you are specifying a regular expression, enclose it in slashes (`/`):

```
folder /i.*ue/
```

If the argument starts with a slash but doesn't end with a slash, regular expression matching doesn't occur, and it's matched as a simple text. If you need to write a simple text search where a text starts and ends with a slash, escape the leading slash with a backslash (`\`):

```
folder \/???/
```

The query in the example above matches folder `/???/.`

Another advanced topic is how to query for the exact word (e.g., match `issue` but not `issues).`

This is called *strict searching*. Strict searching is turned on when the *search text* starts and ends with a double quote ("). Note, however, that quotes are stripped off from function arguments, since quoting is also used to allow specifying spaces or parentheses in the search text. Thus you'll need to enclose the search text in single quotes (').

```
folder '"issue"'
```

item()

`This function matches items of the specified type in the structure, optionally filtering them by name. It is a generalization of folder()` function to other item types.

The function takes two arguments: *item type* and *name* (optional). The second argument works in the same way as the argument for `folder()` function.

You can reference either standard item types (provided by Structure plugin) or item types provided by third-party plugins.

If you need to match items of all types, use asterisk (*). The following query finds all items that have the word “Infrastructure” in their Summary, regardless of their type:

```
item(*, Infrastructure)
```

Structure provides the following item types:

```
issue
project
version
project-component
issuetype
status
resolution
priority
label
user
group
date
cf-option
folder
generator
loop-marker
sprint
missing
tempo-account (when Tempo Timesheets plugin is available)
```

Structure.Pages plugin provides the following item types:

```
page
```

Item types provided by third-party plugins are specified similarly. Here's how `item()` function looks up item types:

1. It tries to interpret *type name* argument as referring to an item type provided by Structure and looks it up in the list above.

2. If not found, it looks at all item types provided by all plugins (including Structure itself) and checks if the type name *ends with* the specified text *as a word*. “As a word” means that `page` will match Confluence page item type, but `age` won't. More specifically, the considered word boundaries are hyphen (-), underscore (_) and colon (:).
3. It is an error to specify item type ambiguously, i.e. if there are two item types matching the description. The following forms of *item type* argument allow to specify item type more precisely.

- Fully qualified item type name, e.g. `com.almworks.jira.structure:type-issue` or `com.almworks.structure.pages:type-confluence-page`. More generally, the form is `<plugin key>:<type name>`.
- Shortened form of the fully qualified item type name, e.g., `structure:issue` or `pages:page`.
  More generally, the form is `<plugin key part>:<type name part>`.
  When `item()` function looks up item type for the argument, and the argument contains colon (:), the function first tries to interpret is as a fully quailified name. Only if nothing is found, it tries to interpret it as a shortened form.

Don't confuse “*matching items of some type”* and “*matching issues that have field value equal to that item”*. For example, `item(status, Open)` matches *status Open*, not *issues with status Open*. If you need the latter, use JQL constraint: `[status = Open]`.

Empty constraint

An empty constraint matching no items:

```
empty
```

This constraint plays the same role as JQL's `EMPTY` keyword. It is intended to be used as a sub-constraint (see page 270) in relational constraints, which are discussed further.

^ up to the list of S-JQL topics (see page 258)

Negation

Any constraint, basic or relational, can be negated using keyword `NOT`. This produces a constraint that matches all items that the original constraint doesn't:

```
not root
```

matches all items that are not top-level items in the structure.

You can always enclose a constraint in parentheses to ease understanding. So, all items in the structure except issues `TS-129` and `TS-239` are matched by this structure query:

```
not (TS-129, TS-239)
```

^ up to the list of S-JQL topics (see page 258)

Relational constraint

A basic constraint matches items that satisfy a condition. A relational constraint matches items *related to* items that satisfy a condition. *Related* corresponds to a relationship between positions of items 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 items that have `TS-129` as a child (sub-item).

Relational constraint has the form `relation operator subConstraint`. Here, `subConstraint` is a constraint on the relatives of items 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 items 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 (see page 258)

Relations

S-JQL has the following relations:

- `child`: item is a child (sub-item) of another item in the structure.
- `parent`: item is a parent of another item in the structure.
- `descendant`: item is a descendant (sub- or sub-sub-...-item) of another item in the structure.
- `ancestor`: item is an ancestor (parent, parent-of-parent, or parent-of-parent-...-of-parent) of another item in the structure.
- `sibling`: item is a sibling of another item in the structure. Two items are considered siblings if they are under the same parent item.
- `prevSibling`: item is a previous (preceding) sibling of another item in the structure. item *A* is a preceding sibling of item *B* if it is a sibling of *B* and *A* is higher than *B* (*A* comes before *B*.)
- `nextSibling`: item is a next (following) sibling of another item in the structure. item *A* is a following sibling of item *B* if it is a sibling of *B* and *A* is lower than *B* (*A* comes after *B*.)
- `self` and `issue` are relations of an item (or an issue) to itself. Their role is explained later, in the `self` and `issue` relation (see page 271) section, because at first one has to learn how operators and sub-constraints work.
  There are also combinations of `issue` and `self` with all other relations, listed for completeness below:

`childOrSelf`	`childOrIssue`
`parentOrSelf`	`parentOrIssue`
`descendantOrSelf`	`descendantOrIssue`
`ancestorOrSelf`	`ancestorOrIssue`
`siblingOrSelf`	`siblingOrIssue`
`prevSiblingOrSelf`	`prevSiblingOrIssue`

`nextSiblingOrSelf`	`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 (see page 258)

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 item.

   For example, consider

   ```
   child in (TS-129, TS-239)
   ```

   Here, `relation` is `child`, so an item's relative in question is its child in the structure. Thus, an item 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 item is matched if it *is not related to* any item matching `subConstraint`.

   As an important consequence, item that has no relatives is matched.

   For example, consider

```
child not in (TS-129, TS-239)
```

An item matches if *no child is TS-129 nor TS-239*. So, this constraint matches all items that either have no children or do not have any of these two items 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 item with no relatives is not matched by the transformed query.

For example,

```
child in (not (TS-129, TS-239))
```

matches all items that have at least one child that is neither `TS-129` nor `TS-239`. That is, the only items that are not matched are leaves and those that have only `TS-129` or `TS-239` as children.

3. `OF` matches the relatives of items that satisfy `subConstraint`.

   For example, consider

```
child of (TS-129, TS-239)
```

An item 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 items in the structure and put each of them, one by one, in column **item**. For each item, we take all of its relatives and put each of them, one by one, in column **relative**. Thus we get pairs of items. 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	item	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 (see page 258)

Sub-constraints

Any constraint can be used as a sub-constraint, whether basic, relational, or a combination of those (see page 272).
For example,

```
child of root
```

selects items on the second level of the hierarchy. To select items 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 items. It is used as a sub-constraint to match items 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 items that have no children (equivalent of `leaf`)
`child is not empty`	matches all items that have at least one child (equivalent of `not leaf`)
`child of empty`	matches all items that are not children of other items (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 item if it is the only child of its parent.

^ up to the list of S-JQL topics (see page 258)

self and issues relations: adding sub-constraint matches to the result set

A relational constraint with relation `self` behaves exactly as its sub-constraint, possibly negated if operator `NOT IN` (`IS NOT`, `!=`) is used.
Thus,

```
self in [status = Open]
```

is equivalent to

```
[status = Open]
```

Similarly,

```
self not in [status = Open]
```

is equivalent to

```
not [status = Open]
```

When combined with another relation, `self` allows to add the items 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 `descendantOrSelf`:

```
descendantOrSelf of TS-129
```

issue relation

`issue` is a special case of `self` relation that only matches issues. For instance, if on the top level of the structure you have folders and issues, and you want to hide all folders, you can write this:

```
descendantOrIssue of root
```

This query matches all top-level issues and all their sub-items.

^ up to the list of S-JQL topics (see page 258)

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]
```

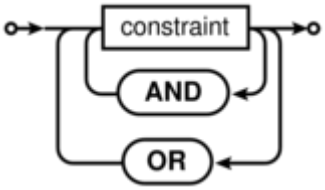
^ up to the list of S-JQL topics (see page 258)

Railroad diagrams

As a final piece of reference, here's the S-JQL syntax in the form of railroad diagrams.

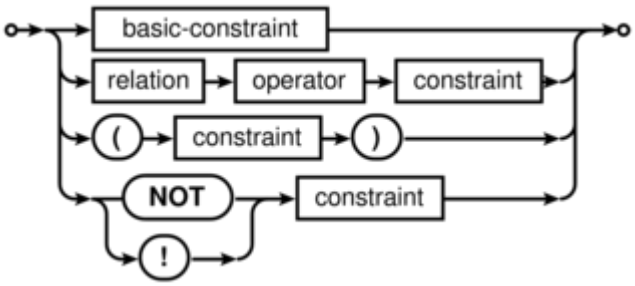
S-JQL keywords are not case-sensitive, and all underscores in keywords are optional.

structure-query

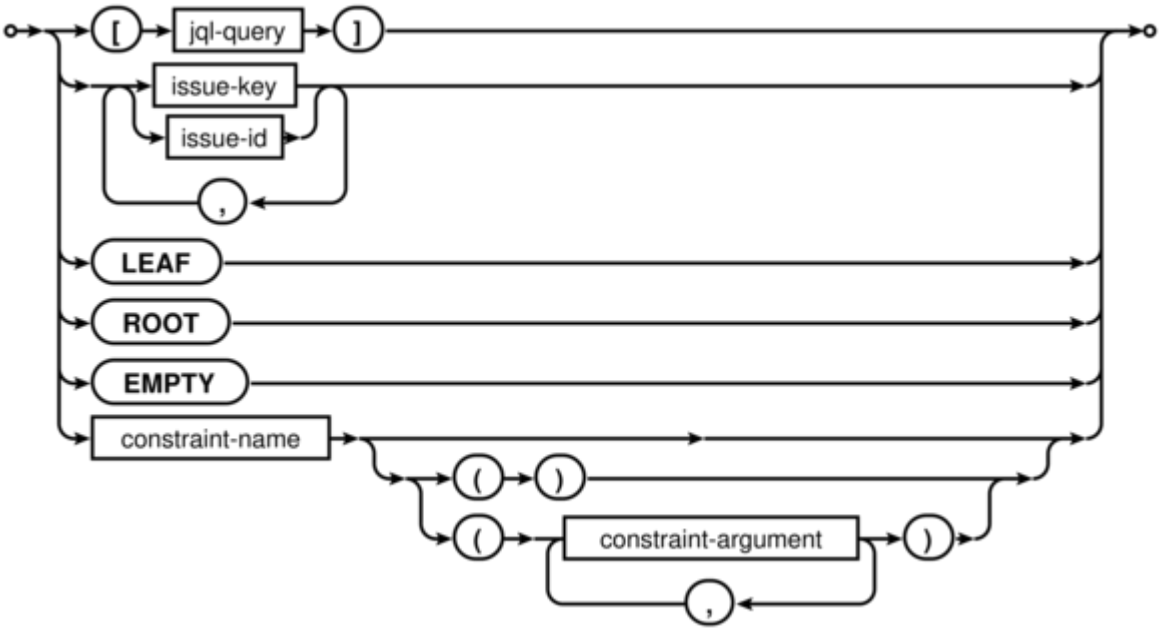


S-JQL admits using `&&` and `&` in place of `AND`, as well as `||` and `|` in place of `OR`.

constraint



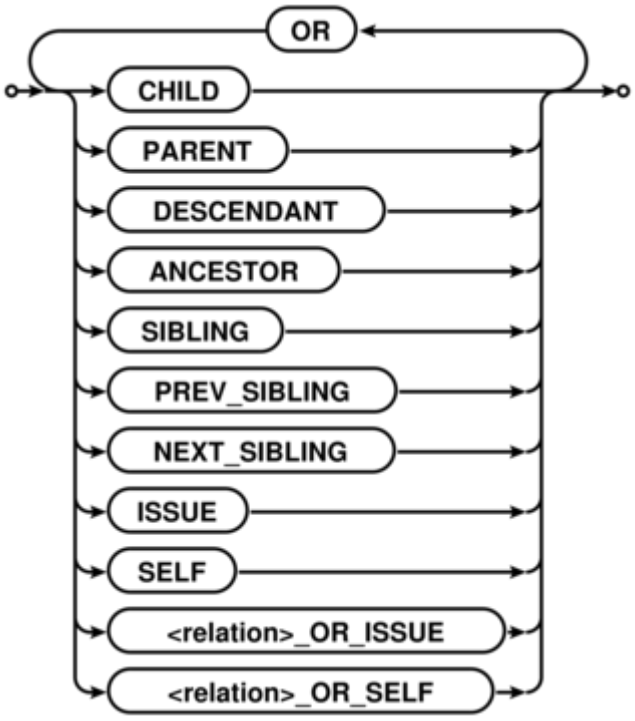
basic-constraint



- `jql-query` is any valid JQL query.
- `issue-key` is any valid JIRA issue key.
- `issue-id` is any valid JIRA issue ID.
- `constraint-name` is the name of the function constraint: either bundled with Structure (`folder`, `item`, or `row_id`) or provided by a Structure extension (plugin).
- `constraint-argument` is one of the following:
  - either a sequence of non-whitespace characters
  - or quoted text (inside "double quotes" or 'single quotes'), where quotes can be escaped via backslash: `\"`, `\'`; backslash itself can be escaped: `\\`.

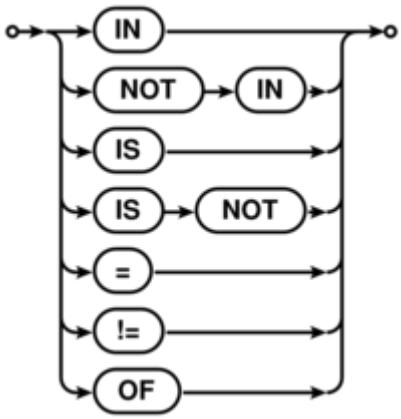
  See also Function constraint - Syntax (see page).

relation



S-JQL admits using | | and | in place of OR.

operator



^ up to the list of S-JQL topics (see page 258)

List of S-JQL keywords

In this article, all S-JQL keywords are listed in all of their spelling variants. This is intended for developers creating their own S-JQL function, because function name must not coincide with the existing keyword.

```
!
!=
&
&&
(
)
,
=
[
]
ancestor
ancestor_or_issue
ancestor_or_issues
ancestor_or_self
ancestorOrIssue
ancestorOrIssues
ancestorOrSelf
ancestors
ancestors_or_issue
ancestors_or_issues
ancestors_or_self
ancestorsOrIssue
ancestorsOrIssues
ancestorsOrSelf
and
child
child_or_issue
child_or_issues
child_or_self
childOrIssue
childOrIssues
childOrSelf
children
children_or_issue
children_or_issues
children_or_self
childrenOrIssue
childrenOrIssues
childrenOrSelf
descendant
descendant_or_issue
descendant_or_issues
descendant_or_self
descendantOrIssue
descendantOrIssues
descendantOrSelf
descendants
descendants_or_issue
descendants_or_issues
descendants_or_self
```

```
descendantsOrIssue
descendantsOrIssues
descendantsOrSelf
empty
following_sibling
following_sibling_or_issue
following_sibling_or_issues
following_sibling_or_self
following_siblings
following_siblings_or_issue
following_siblings_or_issues
following_siblings_or_self
followingSibling
followingSiblingOrIssue
followingSiblingOrIssues
followingSiblingOrSelf
followingSiblings
followingSiblingsOrIssue
followingSiblingsOrIssues
followingSiblingsOrSelf
in
is
issue
issue_or_ancestor
issue_or_ancestors
issue_or_child
issue_or_children
issue_or_descendant
issue_or_descendants
issue_or_following_sibling
issue_or_following_siblings
issue_or_next_sibling
issue_or_next_siblings
issue_or_parent
issue_or_parents
issue_or_preceding_sibling
issue_or_preceding_siblings
issue_or_prev_sibling
issue_or_prev_siblings
issue_or_previous_sibling
issue_or_previous_siblings
issue_or_sibling
issue_or_siblings
issue_or_sub_issue
issue_or_sub_issues
issueOrAncestor
issueOrAncestors
issueOrChild
issueOrChildren
issueOrDescendant
issueOrDescendants
```

```
issueOrFollowingSibling
issueOrFollowingSiblings
issueOrNextSibling
issueOrNextSiblings
issueOrParent
issueOrParents
issueOrPrecedingSibling
issueOrPrecedingSiblings
issueOrPreviousSibling
issueOrPreviousSiblings
issueOrPrevSibling
issueOrPrevSiblings
issueOrSibling
issueOrSiblings
issueOrSubIssue
issueOrSubIssues
issues
issues_or_ancestor
issues_or_ancestors
issues_or_child
issues_or_children
issues_or_descendant
issues_or_descendants
issues_or_following_sibling
issues_or_following_siblings
issues_or_next_sibling
issues_or_next_siblings
issues_or_parent
issues_or_parents
issues_or_preceding_sibling
issues_or_preceding_siblings
issues_or_prev_sibling
issues_or_prev_siblings
issues_or_previous_sibling
issues_or_previous_siblings
issues_or_sibling
issues_or_siblings
issues_or_sub_issue
issues_or_sub_issues
issuesOrAncestor
issuesOrAncestors
issuesOrChild
issuesOrChildren
issuesOrDescendant
issuesOrDescendants
issuesOrFollowingSibling
issuesOrFollowingSiblings
issuesOrNextSibling
issuesOrNextSiblings
issuesOrParent
issuesOrParents
```

```
issuesOrPrecedingSibling
issuesOrPrecedingSiblings
issuesOrPreviousSibling
issuesOrPreviousSiblings
issuesOrPrevSibling
issuesOrPrevSiblings
issuesOrSibling
issuesOrSiblings
issuesOrSubIssue
issuesOrSubIssues
leaf
leaves
next_sibling
next_sibling_or_issue
next_sibling_or_issues
next_sibling_or_self
next_siblings
next_siblings_or_issue
next_siblings_or_issues
next_siblings_or_self
nextSibling
nextSiblingOrIssue
nextSiblingOrIssues
nextSiblingOrSelf
nextSiblings
nextSiblingsOrIssue
nextSiblingsOrIssues
nextSiblingsOrSelf
not
null
of
or
parent
parent_or_issue
parent_or_issues
parent_or_self
parentOrIssue
parentOrIssues
parentOrSelf
parents
parents_or_issue
parents_or_issues
parents_or_self
parentsOrIssue
parentsOrIssues
parentsOrSelf
preceding_sibling
preceding_sibling_or_issue
preceding_sibling_or_issues
preceding_sibling_or_self
preceding_siblings
```

```
preceding_siblings_or_issue
preceding_siblings_or_issues
preceding_siblings_or_self
precedingSibling
precedingSiblingOrIssue
precedingSiblingOrIssues
precedingSiblingOrSelf
precedingSiblings
precedingSiblingsOrIssue
precedingSiblingsOrIssues
precedingSiblingsOrSelf
prev_sibling
prev_sibling_or_issue
prev_sibling_or_issues
prev_sibling_or_self
prev_siblings
prev_siblings_or_issue
prev_siblings_or_issues
prev_siblings_or_self
previous_sibling
previous_sibling_or_issue
previous_sibling_or_issues
previous_sibling_or_self
previous_siblings
previous_siblings_or_issue
previous_siblings_or_issues
previous_siblings_or_self
previousSibling
previousSiblingOrIssue
previousSiblingOrIssues
previousSiblingOrSelf
previousSiblings
previousSiblingsOrIssue
previousSiblingsOrIssues
previousSiblingsOrSelf
prevSibling
prevSiblingOrIssue
prevSiblingOrIssues
prevSiblingOrSelf
prevSiblings
prevSiblingsOrIssue
prevSiblingsOrIssues
prevSiblingsOrSelf
root
roots
self
self_or_ancestor
self_or_ancestors
self_or_child
self_or_children
self_or_descendant
```

```
self_or_descendants
self_or_following_sibling
self_or_following_siblings
self_or_next_sibling
self_or_next_siblings
self_or_parent
self_or_parents
self_or_preceding_sibling
self_or_preceding_siblings
self_or_prev_sibling
self_or_prev_siblings
self_or_previous_sibling
self_or_previous_siblings
self_or_sibling
self_or_siblings
self_or_sub_issue
self_or_sub_issues
selfOrAncestor
selfOrAncestors
selfOrChild
selfOrChildren
selfOrDescendant
selfOrDescendants
selfOrFollowingSibling
selfOrFollowingSiblings
selfOrNextSibling
selfOrNextSiblings
selfOrParent
selfOrParents
selfOrPrecedingSibling
selfOrPrecedingSiblings
selfOrPreviousSibling
selfOrPreviousSiblings
selfOrPrevSibling
selfOrPrevSiblings
selfOrSibling
selfOrSiblings
selfOrSubIssue
selfOrSubIssues
sibling
sibling_or_issue
sibling_or_issues
sibling_or_self
siblingOrIssue
siblingOrIssues
siblingOrSelf
siblings
siblings_or_issue
siblings_or_issues
siblings_or_self
siblingsOrIssue
```

```
siblingsOrIssues
siblingsOrSelf
sub_issue
sub_issue_or_issue
sub_issue_or_issues
sub_issue_or_self
sub_issues
sub_issues_or_issue
sub_issues_or_issues
sub_issues_or_self
subIssue
subIssueOrIssue
subIssueOrIssues
subIssueOrSelf
subIssues
subIssuesOrIssue
subIssuesOrIssues
subIssuesOrSelf
|
||
```

1.12.3 structure() JQL function

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.You can use this function in any place in JIRA where you can use JQL: in the Issue Navigator, in a Saved Filter, as an Agile Board query etc. For more information, see JIRA documentation on Advanced Searching and Advanced Searching Functions.

> If a user does not have access to structure (see page 306), 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.

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 (see page 12) will be searched.

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 (see page 252).

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.

Function arguments need to be quoted if they contain spaces or non-letters

As dictated by the syntax of JQL, you'll need to enclose structure name or structure query in 'single quotes' or "double quotes" if they contain spaces or non-letters.

What if structure name or structure query itself contains quotes?

If structure name or structure query contains quotes of one kind, you need to enclose them with a different kind of quotes. That is, if structure query contains double quote, you'll need to enclose it in single quotes. Alternatively, you can escape quote with a backslash: `\"`.

Example 1

Suppose you need to find all issues that are directly under issues in status *Awaiting Deployment.*

In plain JQL, issues in this status can be found via this query: `Status = "Awaiting Deployment"`. Note that since status name contains spaces, JQL requires us to enclose it in quotes.

According to S-JQL Reference (see page 258), the corresponding Structure query would be `child of [Status = "Awaiting Deployment"].`

That means that you need to enclose this Structure query with single quotes:

```
issue in structure("My personal structure", 'child of [Status =
"Awaiting Deployment"]')
```

Note that the following will **not** work:

```
issue in structure("My personal structure", "child of
[Status = "Awaiting Deployment"]")
```

Example 2: escaping with backslash

In the following example, the query returns issues that are directly under issues assigned to fix version named *3.0 "Armageddon"*.

```
issue in structure("My personal structure", "child of [fixVersion
= '3.0 \"Armageddon\"']")
```

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.

1.13 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.

1.13.1 Keyboard Shortcuts (PC)

Navigation

Action	Shortcut
Select Issue	*Left-Click*
Show/Hide Issue Details	o
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	--

Changing Structure

Action	Shortcut
Move Up	Ctrl+
Move Down	Ctrl+
Indent	Ctrl+
Outdent	Ctrl+
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
Finish & Save	Enter *or* Ctrl+Enter

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
Switch Structure	ss
Add Issue	`Ctrl+Enter`

Standard JIRA Actions

Action	Shortcut
Operations Dialog	`.`
Edit Issue	`e`
Comment on Issue	`m`
Assign Issue	`a`
Assign to Me	`i`
Edit Issue Labels	`l`
Actions Drop-Down	`Alt+`

Action	Shortcut
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	`\`
View Full-Size Image (see page 63)	`ii`
Show/Hide Issue Details without Switching Panel	`Shift+o`
Show Automation	`~`

1.13.2 Keyboard Shortcuts (Mac)

Navigation

Action	Shortcut
Select Issue	*Left-Click*
Show/Hide Issue Details	o
Previous Issue	*or* k

Changing Structure

Action	Shortcut
Move Up	
Move Down	
Indent	

Action	Shortcut
Next Issue	*or* `j`
Expand Sub-Issues	
Collapse Sub-Issues	
For Large Structure	
Add Column	`tt`
Expand All	`++`
Collapse All	–
Change Structure	xx

Structure Views

Action	Shortcut
Switch View	`vv`
Save View	`vs`
Save View As	`vss`
Revert Changes to View	`vr`

Standard JIRA Actions

Action	Shortcut
Operations Dialog	.
Edit Issue	e

Action	Shortcut
Outdent	
Drag and Drop	Drag
New Issue	
New Sub-Issue	
Remove from Structure	

Changing Issues

Action	Shortcut
Edit Field	*Double-Click*
Edit Summary	`tab`
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

Action	Shortcut
Comment on Issue	`m`
Assign Issue	`a`
Assign Issue to Me	`i`
Edit Issue Labels	`l`
Actions Drop-Down	

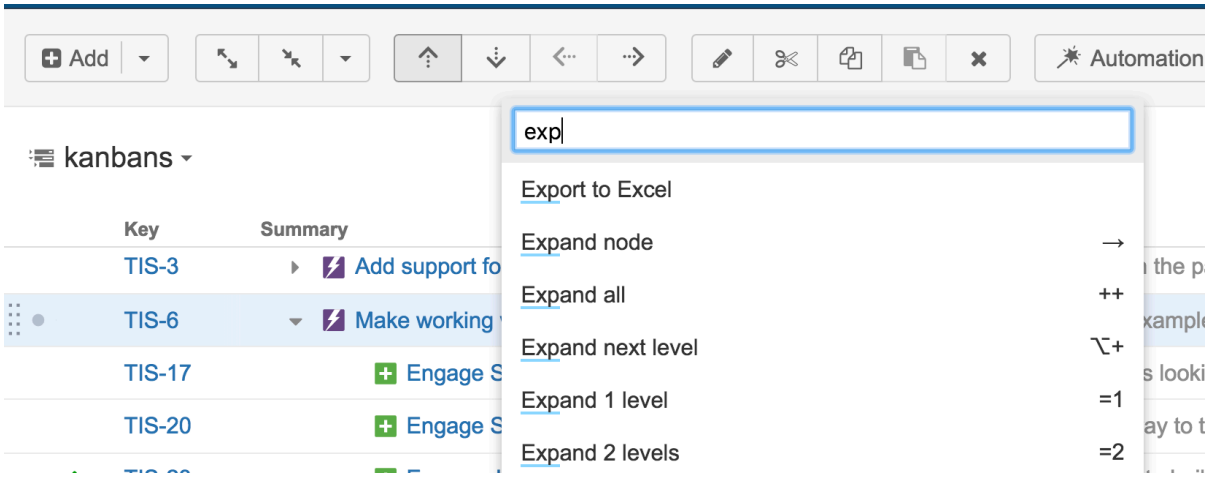
Action	Shortcut
Select All Sub-Issues	
Deselect All Sub-Issues	
Expand Selection Down (Up)	()
Bulk Selection	
Cancel Selection	`esc`

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	`.`
View Full-Size Image (see page 63)	`ii`
Show/Hide Issue Details without Switching Panel	`o`

1.13.3 Quick Action Lookup

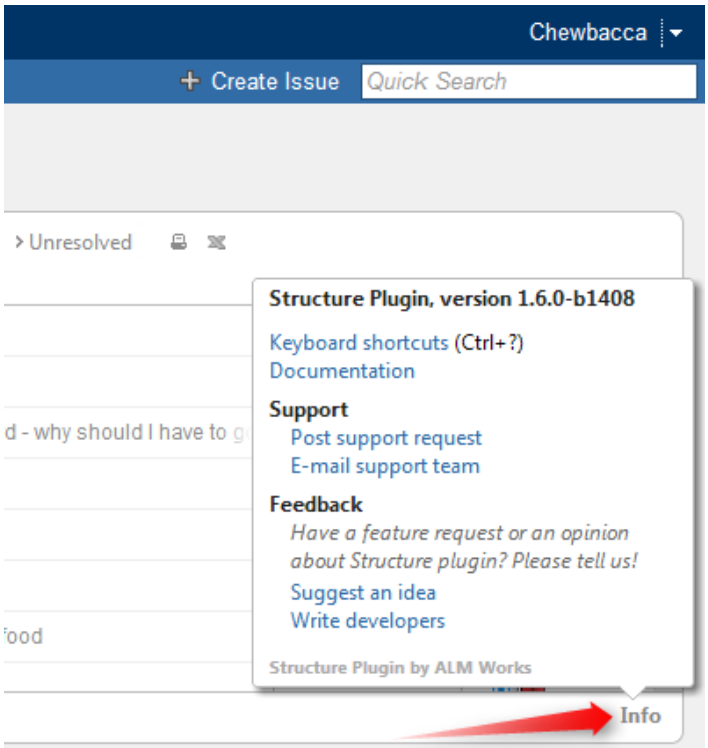
You can use a special keyboard combination – **`s,q`** – to pull up the "Action Lookup" input box. In that input box you can start typing what you need to do and it will suggest available "actions" that match the description.



It also shows keyboard shortcut associated with the action.

1.14 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.

2 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 (see page 292)
2. Setting Up Structure License (see page 299)
3. Getting Started with Structure (see page 304)

Contents:

2.1 Installing Structure

Structure is installed like most other plugins.

1. Before installing Structure in production, make sure your JIRA meets the Memory Guidelines (see page 294).
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 (see page 299).

Congratulations! You can now spread the word and help users get started with Structure – see Getting Started with Structure (see page 304).

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 (see page 450) article for possible causes and solutions.

Next: Set up Structure license key (see page 299)

2.1.1 Migrating Data from Structure 2 to Structure 3

Unlike previous versions, Structure 3.0 uses the main JIRA database to store its data. You need to migrate the data from Structure 2 in order to continue working with it in Structure 3. Additionally, this feature can be used to restore structures from a backup made with Structure 2.

Structure 2 had a separate Backup / Restore functionality, because Structure data was kept separately. With Structure 3, all data is backed up with the usual JIRA backup.

However, we plan to reinstate Backup / Restore / Migrate feature in the future versions of Structure 3.

Creating a Backup of Structure 2.x Data

- If you still have Structure 2.x installed, create backup of current Structure data. You can either use **Administration | Structure | Structure Backup** menu or do a cold backup by copying the whole `structure/` sub-directory under JIRA home while Structure plugin is disabled. See Backing Up Structure (see page 314) for details.
- If you already have Structure 3.x installed, use **Administration | Structure | Export Structure 2.x Data** page. It allows you to create a backup zip with all Structure 2.x data, and then opens **Restore Structure** page, allowing you to immediately import the backup into Structure 3.x database.

Restoring Structure Data from 2.x Backup

1. Use **Administration | Structure | Restore Structure** menu and use any Structure 2.x backup made earlier. Note that it should be placed in the `import/` directory on your server.
2. If you used "Export Structure 2.x Data" menu, you will be taken to the restore automatically.

After Data Migration

Upgrade Testy

If you have Structure.Testy installed, download and install the latest version of Structure.Testy, compatible with Structure.

Upgrading "Global Structure"

If you're using "Global Structure" structure, which was created by default in Structure 2.x, you need to make sure that there's an "owner" of that structure. Otherwise, Automation (see page 176) will not work there.

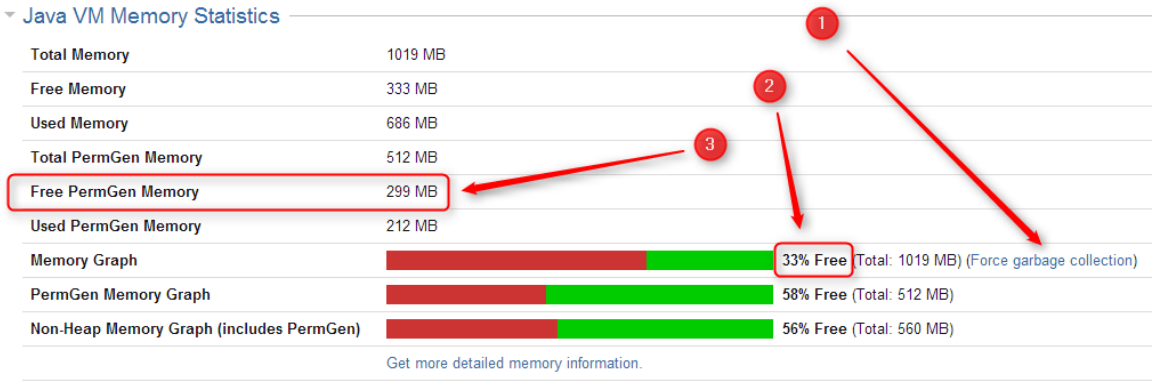
1. Open **Structure | Manage Structure**.
2. Find Global Structure and check if it has non-empty Owner.
3. If it doesn't have an owner, click **Configure**, and set yourself as the owner.

2.1.2 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 (prior to Java 8)	100 – 200 MB	`JIRA_MAX_PERM_SIZE`

If you run JIRA on Java 8, PermGen memory is not a factor.

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.

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**

2.1.3 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 the JIRA's database.

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.

2.1.4 Upgrading and Downgrading

Upgrading

General upgrade procedure is simple:

1. Create backup of Structure data. Use **Administration | Structure | Backup Structure**. See Backing Up Structure (see page 314) for details.
2. Install the new version of the plugin.
3. Check Structure extensions. If you are using Structure.Testy, Structure.Pages, or other extensions, they will most likely become disabled. You need to either upgrade them too (it might be a compatibility requirement), or enabled them manually in the Add-on Manager. If they fail to enabled, reinstall them (uninstall and install again).
4. Check plugins that integrate with Structure, such as Colors or Gantt Chart. Like with extensions, see if they are enabled and maybe upgrade or reinstall them.
5. Monitor `catalina.out` or `jira-application.log` for warnings or errors.

For more specific instructions, please check the Release Notes for the version to which you wish to upgrade.

Downgrading

Reverting the plugin to an older version is not always possible because newer versions can modify the database so it becomes incompatible with older versions.

Simplified Downgrade

Simple downgrade is possible if the database schema hasn't changed. Check the Release Notes for the version you are downgrading from and look for downgrade advisory. Proceed only if you have indications that it is safe to downgrade to the specific version you have in mind.

1. Uninstall Structure plugin. This step is required because Add-on Manager will not install an earlier version over a later version.
2. Install the version that you need.
3. Check Structure extensions and integrating add-ons. See the steps in the Upgrading section above.
4. Monitor `catalina.out` or `jira-application.log` for warnings or errors. **This is especially important with this kind of downgrade, because some errors may be subtle and not visible to the users!**

Reliable Downgrade

Reliable downgrade requires Structure backup file and manual access to the database.

1. Create Structure backup using **Administration | Structure | Backup Structure**.
    a. Backup files are backward / forward compatible along Structure 3.x series. You cannot use Structure 3.x backup to downgrade to Structure 2.
    b. You can use a previously created backup file. **Note that all data will be rolled back to the state when the backup file was created.**
2. Uninstall Structure add-on.
3. **Double-check you have the backup! You are about to delete all Structure data.**
4. Manually access your database using database tools. Drop all tables that start with `AO_8BAD1B_`. If after that you have other objects starting with that prefix, drop them too.
5. Install previous version of Structure add-on.
6. Use **Administration | Structure | Restore Structure** to populate the data from the backup file.
7. Check Structure extensions and integrating add-ons. See the steps in the Upgrading section above.
8. Monitor `catalina.out` or `jira-application.log` for warnings or errors.

Creating a backup and restoring from backup may require considerable time. If you want to speed up the process and you don't need the history of structure changes, turn off the option "Include History" when creating a backup.

2.2 Setting Up Structure License

Unless your JIRA runs on one of the free licenses (see page 303), 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.

2.2.1 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 (see page 303).

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.

2.2.2 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**
Purchased at	ALM Works website	Atlassian Mar
Managed at	The license key is sent to you by email	Manage with
License key looks like this:	`-----BEGIN CERTIFICATE----- MIIEYTCCAkmgAwIBAgIGAT2oPFqOMA0GCSqGSIb3DQE... ... at least 20 lines of symbols ... -----END CERTIFICATE-----`	`AAABEA0ODA ... at lea`
Installation Instructions	1. If you have a license from Atlassian installed, first remove it in the **Plugin Manager**.	1. Open **P** 2. Locate

	License from ALM Works	License from
	2. Open **Administration | Structure | License Details**. 3. Copy and paste the key to the **Install License** section and click **Install License**.	3. Copy th click **Up**
Uninstallation Instructions	1. Open **Administration | Structure | License Details**. 2. See the details of installed license and click **Uninstall**.	1. Open **P** 2. Locate 3. Clear th click **Up**
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.	• Purcha certain

2.2.3 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.

2.2.4 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 (see page 305)

2.2.5 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	

Parameter	Meaning
	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.

2.2.6 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.

2.2.7 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

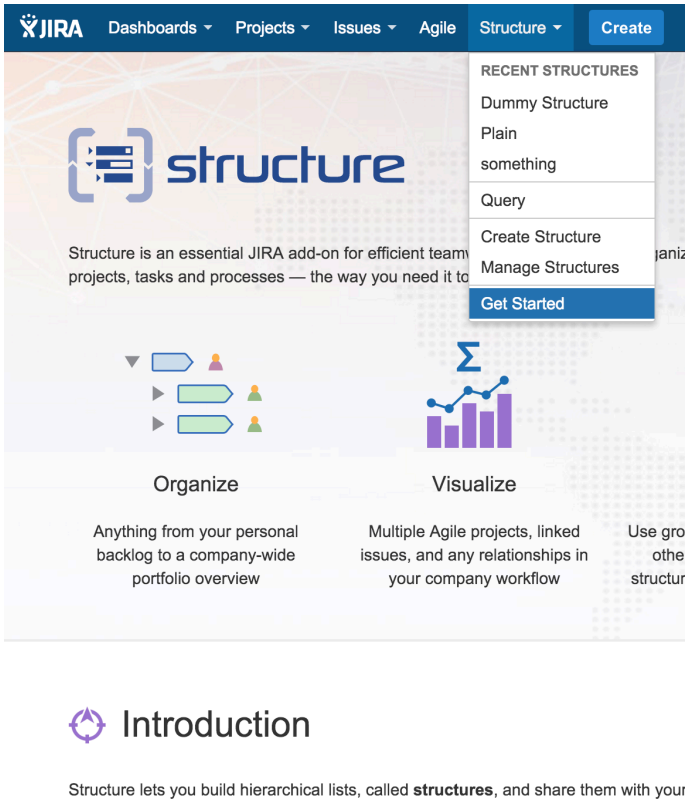
If the currently used license becomes invalid (for example, because it is expired, or because you've upgraded to a version of Structure that's not covered by the current license's maintenance), then Structure plugin will function in read-only mode.

The users will be able to view structures, but they won't be able to make any changes until a valid license is installed.

2.3 Getting Started with Structure

Structure comes with a short tutorial that is recommended for everyone who starts working with Structure and for those who have previous experience with Structure 2.11 or earlier. The tutorial is available under **Structure | Get Started** menu.

As it takes some (reasonable) effort to learn Structure before starting to use it efficiently, consider sending out a link to this page to every user in your company who might have use for Structure.



2.4 Selecting Structure-Enabled Projects

Structure can be enabled for any selection of the JIRA projects, or for none of them. (In the latter case no one can use Structure.)

> ⚠ By default, Structure is enabled for all projects. To limit users' exposure to Structure, pick specific projects to be enabled for Structure.

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 (see page 306)

2.5 Global Permissions

2.5.1 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 (see page 305).
- Structure is enabled for this user (see page 306):
    - 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 (see page 305).

Users who have *JIRA Administrators* global permission always have access to Structure.

2.5.2 Restricting User Access to Structure

By default, Structure is accessible to anyone who has *Browse* permission on structure-enabled projects (see page 305). 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 (see page 305).
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 (see page 306).

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 (see page 194)). 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.

2.5.3 Changing Permission to Create New Structures

By default, any logged-in user with access to Structure (see page 306) 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 (see page 305).
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 (see page 306) 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 (see page 194)). 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.

2.5.4 Changing Permission to Manage Synchronizers

By default, any logged-in user with Control access level (see page 194) for a structure can manage that structure's Synchronizers (see page 219). 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 (see page 305).
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.

2.5.5 Changing Permission to Access Automation

By default, any user with Edit Generators access level (see page 194) for a structure can add and configure generators (see page 176). You can restrict this ability to one or more user groups or project roles.

To select who can edit generators:

1. Navigate to **Administration | Structure | Configuration**.
2. Click **Select Who Can Access Automation** .
3. Select whether generators can be changed by **Anyone with Edit Generators permission** 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 (see page 305).

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 Edit Generators access level for a structure to be able to add or change generators in it.

Users who have *JIRA Administrators* global permission are always allowed to change generators.

2.6 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.

2.6.1 Initial Configuration

When Structure plugin is installed, the defaults are configured as follows:

System Default Structure	None
Project Default Structure	None
Default Views Menu	Preinstalled views **Basic**, **Planning**, **Tracking**, **Triage**, **Entry** (on all pages)
Default View	**Basic View**
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.

2.6.2 Changing Default Structure

Default structure (see page 12) is selected when the user opens Structure Board (see page 15) for the first time, or when the Auto-switch option (see page 21) 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 (see page 194). 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.

2.6.3 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 (see page 196).
4. Click Apply

2.6.4 Changing Default Options for the Issue and Project Pages

A number of options define how Structure Panel behaves on the issue page (see page 21) and on the project, component and version pages (see page 30). 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 (see page 21)

Option	Description	See Also
Auto-switch (Project Page)	Lets you automatically select structure displayed on the Project, Component and Version pages.	Structure on the Project Page (see page 30)
Keep Structure Selection When Navigating	When turned on, clicking an issue in the Structure Widget (see page 15) opens that issue's page and shows the same structure on that page initially.	Structure Options for the Issue Page (see page 21)
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 (see page 21)

2.7 Structure Backup, Restore and Migration

Structure data can be backed up and restored separately from other JIRA data. Structure data includes structures, hierarchies (forests), synchronizers, generators, folders - everything added to JIRA by the Structure add-on. Structure backup does not include issue or other items data (except for some attributes that are added to enable migration.)

You need the *JIRA System Administrators* global permission to back up, restore or migrate Structure data.

Starting with Structure 3, when you fully back up JIRA, Structure data is also backed up – it is stored in the same database with JIRA data. However, you can use the separate backup:

- To be able to restore only Structure data, not changing JIRA data
- To be able to migrate structures to other servers (following Project Import in JIRA, for example)
- To export Structure data to some other tool by parsing the backup XML

2.7.1 Using Structure Backup

Structure add-on can use a backup file 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 and other items 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.

2.7.2 Backing Up Structure

Backing up Structure saves the existing structures, their configuration, hierarchies and other Structure data. Structure backup does not save the issues themselves or other JIRA data - see Structure Backup, Restore and Migration (see page 313).

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 *.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. Use **Backup History** checkbox to include full change history in the backup file.

4. Click **Backup**
5. If the file already exists, you will be given an option to overwrite the file or cancel the operation.
6. You will see the **Process Status** page where you can track if the backup is going on or is finished. Once it's finished, click **Show Results** to see the full name of the backup file.

2.7.3 Restoring Structure from Backup

Restoring structure from backup brings back the structures, synchronizers, views and other data 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 (see page 313).

> ⚠ The issues and other items 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 (see page 316).

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. You will see the Process Status page that will tell you if Restore is going on or is finished. Once it's finished, you'll be able to see the result and possibly some warning messages.

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, including Structure 2.

2.7.4 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 (see page 313).

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 (see page 190) 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 (see page 219) 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 (see page 196) for the selected structures will be restored.

6. Click **Restore Selected Structures**.
7. You will see the Process Status page that will tell you if migration is going on or is finished. Once it's finished, you'll be able to see the result and possibly some warning messages.

After structures have been migrated, open **Structure | Manage Structure** page to see if new structures are there.

> ⚠ As of version 3.3, Migrate Structure does not support Structure.Testy or Structure. Pages data.

2.8 Automatic Structure Maintenance

2.8.1 Automatic Structure Maintenance

Automatic Structure maintenance runs daily and performs Structure backup and database optimization. The 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.

2.8.2 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. If you have a lot of changes in structures this setting may cause the backup to take some time and the backup file to be large. If you don't need history of structure changes, it is advised to turn this option off. It is advised to have separate Structure backups even though Structure data is backed up with JIRA's normal backup, because you will be able to Restore from that data without rolling back changes in JIRA.
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 (see page 12), Structure plugin will keep this mark even if the user is later deleted from JIRA. Popularity number of the structure (see page 12) 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 from structures references to deleted issues and references to other items that have become permanently unavailable.

Optimize view settings	If a view (see page 209) is deleted, some structure view settings (see page 196) 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 add-on may keep data related to synchronizers of a structure which was already deleted. This task removes such data.
Delete old synchronizer audit log records	This removes old records from Synchronizer Audit Log, clearing up space in the database. Parameters: • Keep records for the last *X* days. If you set X to 0, maintenance procedure will remove all records.
Reindex change history	Currently does nothing. This task has remained as an option since Structure 2. Its purpose will be restored later when Structure 3 gets more maintenance options for structure histories.
Optimize structure perspectives	Removes old perspectives (see page 217) 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
Reindex structures	Clears and recalculates issue-to-structure index, used to define which structures contain a specific issue. (Issues added with Automation (see page 176) are not counted.)
Delete old change history	The task removes old records from change history. A history record is considered old if the change was made earlier than X days ago (X is specified by the first parameter) **and** it is not among Y latest history records for the structure where the change was made (*Y* is specified by the second parameter). Parameters:

	• Always keep change history for the last X days • Always keep Y latest changes per structure

2.8.3 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.

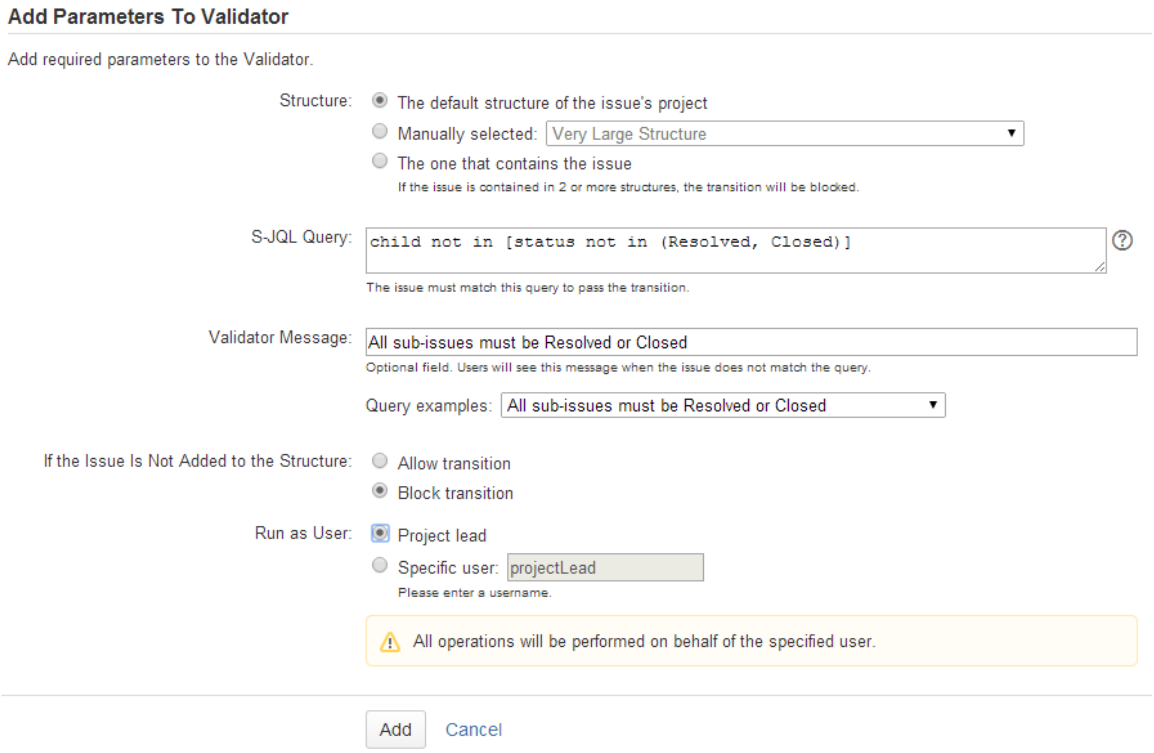
2.9 Workflow Integration

2.9.1 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 (see page 252). 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).

   In Structure 3, the option that picks a structure that contains the issue being validated is no longer available.

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.

2.9.2 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.

2.10 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

2.10.1 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**

2.10.2 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.

2.11 Structure Files

2.11.1 $JIRA_HOME/structure

Structure keeps most of its data in the `structure` sub-directory under the JIRA home directory .

On JIRA Data Center, a local filesystem is used.

Cache files

Structure uses file system to temporarily store some of the internal runtime database, involved in Automation feature. These files may be stored in "rows0", "rows1" and similar directories.

2.12 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 (see page 333).

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 Add-on Manager by navigating to *Administration* | *Add-ons* | *Manage Add-ons*.
2. Locate Structure add-on and expand its row.
3. Click the link that looks like the following: "309 of 310 modules enabled." (Numbers may vary.)
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 (see page 249), • Activity tab on the issue page, • Activity tab on the user page, • Activity tab on the project page.
		Structure section is removed from the issue page.

Feature	Module name	Effect of disabling this module
Structure on the Issue Page (see page 17)	`web-resource: Issue Page Decorator (adjustIssue)`	
Structure on Agile Boards (see page 31)	`web-panel: GreenHopper tab (greenhopper-tab)`	Structure tab is removed from the issue details panel on the Agile board.
Synchronizers (see page 219)	`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.)

2.13 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.

2.13.1 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.

2.13.2 Setting System Properties with Script Runner

If you don't want to restart JIRA, you may use the 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 after you restart the Structure, but the properties will be reset to their default values when you restart JIRA. In some cases for settings to take effect you have to reinstall the Structure. But If you want the changes to be permanent, please use the `-D` startup option as described above.

2.13.3 Synchronizer Cycle Guard

The cycle guard (see page 229) 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

Property	Default	Explanation
		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. 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

Property	Default	Explanation
		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.

2.13.4 Structure size limit

Property	Default	
`com.almworks.jira.structure.AOBasedStructureManager.forestSizeLimit`	`100000`	

2.14 System Requirements

2.14.1 Atlassian Platform

JIRA Versions Supported	7.1 – 7.3 (by the latest version) See also: Platforms supported by JIRA
JIRA Editions Supported	JIRA Core, JIRA Software, JIRA Service Desk
JIRA Data Center	Supported See Server Requirements below
Confluence Versions (Structure.Pages)	5.9 – 6.2

2.14.2 Databases

Databases used by JIRA are also supported by Structure. We test the product with the following databases:

- Oracle 12C
- MySQL 5.1, 5.5, 5.6
- PostgreSQL 8.4, 9.0,9.1,9.2,9.3
- Microsoft SQL Server 2012, 2008, 2005
- HSQL (not for production use)

2.14.3 Browsers

Structure Plugin is compatible with the following browsers:

Browser	Supported versions	Versions known to NOT work
Mozilla Firefox	All recent versions	
Chrome	All recent versions	

Internet Explorer	11	8 Internet Explorer 9 and 10 are partially supported. There are a couple of low-impact known issues.
Safari	All recent versions on OS X	Safari for Windows is not supported.
Edge	All recent versions	
Other browsers	Unsupported, but may work	

2.14.4 Server Requirements

- At least 100MB of free disk space is needed on the server. See Structure Files Location (see page 324) for details.
    - On JIRA Data Center, each node must have sufficient free disk space in the local home.
- Java process running JIRA needs at least additional 200 MB of heap memory. If running on Java 7 or earlier, ensuring sufficient free PermGen space is recommended. See Memory Guidelines (see page 294) for details.
- JIRA process must have read/write permissions to JIRA (local) home directory to create `structure` sub-directory automatically.

2.14.5 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.

2.15 Best Practices

We have collected several guidelines for common situations.

If you have your own best practice to suggest, please let us know!

2.15.1 Backup Strategy

Prior to version 3.0, Structure used to store data separately from JIRA data and it was not included in the general System Backup. That called for a separate backup strategy. With version 3.0 and later, this matter is simplified.

General Approach

Structure data is backed up along with JIRA data when you make full system backup.

However, Structure can back up and restore its data separately. This allows you to roll back Structure-related changes without affecting other JIRA data and generally safeguards your structures.

The following backup strategy is sufficient in most cases.

Option 1. Automatic XML Backup + Export Directory Backup

This strategy involves two processes:

- Automatic Structure Maintenance (see page 318) 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.

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. Manual / API-Triggered XML Backup

You can manually back up structure through Structure Backup (see page 314) 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 (see page 335) (use `StructureBackupManager` interface).

Restoring from XML Backup

See Restoring Structure from Backup (see page 315) 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.

2.15.2 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 (see page 306). 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.
5. Don't forget to advise everyone to check the Getting Started (see page 304) page.

In the same way, you can gradually enable Structure project-by-project. See Who Has Access to the Structure (see page 306) for details.

Turning Optional Functionality Off

Some Structure features can be turned off – see Turning Off Optional Features (see page 325).

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.

Automation (see page 176) is the newest feature that can potentially place considerable load on the server. You can limit the access to it by changing permission to access Automation (see page 309).

2.16 Dark Features

Dark features are additional features or behavior modifications that are usually hidden from the user. However, JIRA administrator can turn them on for their instance.

2.16.1 Alternative initial values for project/type when creating an issue in dialog

Normally, when the user creates new issues through dialog, Structure remembers the selected project and issue type and offers them next time by default. This dark feature enables a different algorithm, which used to work in a previous version of Structure: the initial project and issue type are taken from the issue that was focused when "+Create" or "+Next Issue" was pressed.

System property	`structure.feature.altInitialValuesInDialog`
Options to add in setenv.sh / setenv.bat	`-Dstructure.feature.altInitialValuesInDialog=true`
Internal feature name	`altInitialValuesInDialog`
Introduced in version	2.11.0

3 Structure Developer's Guide

3.1 Structure Developer Documentation

Structure for Developers

Structure add-on provides APIs that allow you to access structures, integrate your add-on with Structure and extend Structure functionality. 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 (see page 337) and Java API reference (see page 380).

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 JIRA Plugin (see page 337) .

Extending Structure

You'd like to extend Structure, adding functionality to the plugin itself – read documentation about extending Structure functionality with additional plugins (see page 356).

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 (see page 379) and Structure REST API (see page 389).

3.2 Structure Concepts, Developer's Perspective

This article provides an introduction to the main concepts used in Structure. Before starting your work on integration with Structure, please familiarize yourself with these concepts.

3.2.1 1. Basic Concepts Overview

Concept	Short Definition	API Classes to Check
Structure	A named container for a hierarchical list.	`Structure`, `StructureManager`
Forest	A hierarchical list.	`Forest`, `ForestService`
Row	A row is a unique, atomic element of a forest.	`StructureRow`, `RowManager`
Item	An item is a user-level object (like Issue) that is displayed in a row.	`ItemIdentity`, `CoreIdentities`
Attribute	An attribute provides values of a certain type and meaning for forest rows.	`AttributeSpec`, `StructureAttributeService`
Column	A column loads one or more attributes and displays information about forest rows.	`ViewSpecification`
View	A view is a named collection of columns.	`StructureView`, `StructureViewManager`

Important points:

- **Structures** are the main entities provided by Structure add-on. A structure has name and other attributes, like description, and it also has content, represented by a **forest.**
- A forest represents a structure's content. But it can also represent a result of a query or a hierarchical list received or stored somewhere else.

- Forest contains **rows**. Forest content is actually a list of pairs (`row ID, depth`).
- A row has a numeric ID that uniquely identifies it in a forest. A forest may not contain the same row twice. (Although a row may be present in different forests.)
- When users look at a structure, they see a grid – each row in that grid is represented by a Structure's row.
- A row refers to an **item**. An item is an abstraction for everything that can be placed into a forest – issues, folders, projects, users are all items, from Structure's perspective.
- An item has **item identity** – something that uniquely identifies that item on a JIRA instance.
- An item also has **attributes** – some values with associated meaning, which Structure and its extensions can provide and that can be shown to the user.

3.2.2 2. A Note on Extensibility

Structure is built with extensibility in mind. It is possible for a separate add-on to add new item types, attributes, columns and other extensible elements to Structure, at runtime.

3.3 Accessing Structure from JIRA 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.

Check out the articles below for details.

3.3.1 Setting Up the Integration

To start using Structure in your plugin:

1. Add dependency to your pom.xml

Figure out the version of the API (see page 380) 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>16.0.0</version>
  <scope>provided</scope>
```

```
</dependency>
```

Note that there are Additional Libraries Used in Structure API (see page 338)

2. Import StructureComponents

In your `atlassian-plugin.xml`, use `<component-import>` module to import `StructureComponents` service. This service provides access to all other Structure services.

Alternatively, you can import specific services.

```
<component-import key="structure-components" interface="com.
almworks.jira.structure.api.StructureComponents"/>
```

3. Have Structure API service injected into your component

```
public class MyClass {
  private final StructureManager structureManager;

  public MyClass(StructureComponents structureComponents) {
    structureManager = structureComponents.getStructureManager();
  }

  ...
}
```

This is it! Continue to the list of Structure Services (see page 342) to see which service you need to work with. Other articles in this section provide examples for specific use cases.

For a production plugin, consider Controlling Compatibility (see page 339). For a standalone plugin, which can work without Structure, read about Making Structure Dependency Optional (see page 340).

Additional Libraries Used in Structure API

Structure API has dependencies on a few open-source libraries that are transitively included in your project when you add a dependency on Structure API.

You don't need to explicitly add dependencies on these libraries.

Integers and HPPC

The open source library Integers provides collections of primitive types with `java.util`-like interfaces. When working with `Forest`, you will typically use `LongList` and `LongArray` (an implementation of `LongList`).

It comes with another primitive type collection library, HPPC, which provides specific implementations of these collections.

See API Usage Samples (see page 437) to get the idea how to work with those interfaces.

JetBrains Annotations

Annotations library from JetBrains provides `@Nullable` and `@NotNull` annotations, used throughout the API.

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 (see page 380) page explains how version numbers change based on how compatibility is affected. Say, you develop your code using Structure API version 16.2.0 – your code will work with any version of the API starting from 16.2.0 and up to, but not including version 17.0.0.

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 (see page 380) table and modify your `pom.xml` to contain the following:

```
<plugin>
  <groupId>com.atlassian.maven.plugins</groupId>
  <artifactId>maven-jira-plugin</artifactId>
  ...
  <configuration>
    <instructions>
      <Import-Package>
        com.almworks.jira.structure.api*;version="[16,17)",
        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 (see page 437) for a more complete example.

Next: Making Structure Dependency Optional (see page 340)

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.

1. 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 (see page 339) if you don't have this declaration yet.)

```
<Import-Package>
  com.almworks.jira.structure*;version="[16,17)";resolution:
=optional,
  com.almworks.integers*;version="0";resolution:=optional,
  org.jetbrains.annotations;version="0";resolution:=optional
</Import-Package>
```

2. 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("co
m.almworks.jira.structure")) {
      return false;
    }
    try {
      Class.forName("com.almworks.jira.structure.api.
StructureComponents");
```

```
    } catch (Exception e) {
      return false;
    }
    return true;
  }

  public static ForestAccessor getForest(long structureId) {
    if (!isStructurePresent()) return null;
    StructureComponents structureComponents;
    try {
      structureComponents = ComponentAccessor.
getOSGiComponentInstanceOfType(StructureComponents.class);
    } catch (Exception e) {
      return null;
    }


    try {
      return new ForestAccessor(structureComponents.
getForestService().getForestSource(ForestSpec.structure
(structureId)));
    } catch (StructureException e) {
      return null;
    }
  }
}
```

3.3.2 Structure Services

This page lists public services provided by Structure API. All these services are available from StructureComponents instance.

Services to Start With

Use ...	to ...
StructureManager	Create and delete structures, modify structure properties such as name or permissions. (But not to work with the structure's content.)
ForestService	Access forests for reading or changing.
StructureAttributeService	Retrieve attribute values for given rows in a given forest.
RowManager	Extract item information for rows read from a Forest.

Use ...	to ...
FolderManager	Create folders or change folder properties.
GeneratorManager	Create generators or change generator properties.

More Power

Use ...	to ...
StructureConfiguration	Change global Structure add-on configuration.
StructureViewManager	Create and manipulate views.
StructureSyncManager	Manage synchronizers.
StructureBackupManager	Backup complete Structure data to a file or restore it back.
StructureFavoriteManager	Read or change which structures are favorite of which users.
PropertyService	Store arbitrary properties.
StructurePropertyService	Store arbitrary per-structure properties.

Extreme Power

Use ...	to ...
ItemTracker	Track recorded changes that happened to items (in JIRA Data Center – on all nodes of the cluster).
ItemResolver	Convert `ItemIdentity` into an object representing that item.
IssueEventBridge	Listen for or report issue events.
StructureQueryParser	Parse an S-JQL query.

Use ...	to ...
StructureQueryBuilderFactory	Build an S-JQL query via Builder pattern.
ProcessHandleManager	Manage feedback page for asynchronous processes.
SyncAuditLog	Access or manage Synchronization Audit log.
StructureJobManager	Run a job asynchronously.
ScheduledJobManager	Schedule a periodical job to run asynchronously (only on a single node in a cluster).

3.3.3 Building Forest Specification

A forest specification, or `ForestSpec`, is a way for your code to identify the forest that you'd like to access. The forest may come from different sources – it could be a structure, it could be a transformed (see page 133) structure, it could be a result of query or some other types of forest source.

So the first step before you read or update a forest is to create an instance of ForestSpec. Here are some examples of how you can do that.

Desired forest	ForestSpec expression
Base Content	
Structure #123	`ForestSpec.structure(123)`
Result of a JQL query	`ForestSpec.sQuery("jql", "priority = Blocker")`
Result of a text query	`ForestSpec.sQuery("text", "text to find")`

Desired forest	ForestSpec expression
Adjusted Content	
Structure #123, sorted by Priority	`ForestSpec.structure(123).transform( CoreStructureGenerators. SORTER_ATTRIBUTE, ImmutableMap.of( "attribute", (Object) ImmutableMap.of("id", IssueFieldConstants.PRIORITY, "format", "order") "desc", true ) );`
Structure #123, skeleton only (without dynamic content)	`ForestSpec.skeleton(123)`
Structure #123, with title row	`ForestSpec.skeleton(123).withTitle()`

More details are available in Javadocs for ForestSpec.

3.3.4 Reading Structure Content

Let's say you need to access a structure's content and export the hierarchy into your custom format or use for displaying the hierarchy in your way. This scenario walks you through from having just a structure name to iterating through the forest and learning which items are there.

We assume that your code has `StructureComponents` instance injected into `myStructureComponents` field.

1. Figure out Structure ID

To address a structure, you need to know its ID. If you just have a name you can do the following:

```
List<Structure> structures = myStructureComponents.
getStructureManager().getStructuresByName("My Structure",
PermissionLevel.VIEW);
long structureId;
if (structures.size() == 1) {
  structureId = structures.get(0).getId();
} else {
  // no structures or too many structures -- error?
}
```

Now you have `structureId` or an error situation where the name does not uniquely identify your structure.

2. Create a ForestSpec

You need a forest specification to get a ForestSource. You can read more about this in the section about Building Forest Specification (see page 344).

```
ForestSpec forestSpec = ForestSpec.structure(structureId);
```

Note that this forest spec is going to be "secured" for the current user, which means that the resulting forest will exclude the sub-trees that only contain items not visible to the user.

3. Retrieve ForestSource

A `ForestSource` is an interface that produces some specific forest and that provides versioning for it.

```
ForestSource forestSource = myStructureComponents.
getForestService().getForestSource(forestSpec);
```

Note that this call may produce `StructureException` in case a structure cannot be found and in some other cases. A robust code would have some exception handling.

Do not store a `ForestSource` in memory for a long time, longer than a single user request. Structure has internal caching engine that efficiently manages forest sources and their dependencies. Request forest source from `ForestService` in every new request.

4. Retrieve Forest and its version

Forest source can provide you with the latest version of the forest, or with an incremental update, based on the version you already have.

To get the latest forest:

```
VersionedForest versionedForest = forestSource.getLatest();
DataVersion latestVersion = versionedForest.getVersion();
Forest forest = versionedForest.getForest();
```

Note that `latestVersion` variable contains the version of the forest that you got. You can later use it to call `forestSource.getUpdate(latestVersion)` and receive only information about how did the forest change since the last time you've seen it.

You cannot really use `latestVersion` for anything else besides getting updates later. The numbers in that version bear no meaning regarding structure's history. For history queries, you'll need to use `HistoryService`.

5. Iterate through Forest and get StructureRow instances

A `Forest` is just two parallel arrays, one containing row IDs, the other containing depths. (Or, one can say that it is a list of pairs `(rowId, depth)`.) You can iterate through it via simple cycle.

For each row, you'll need more information than just row ID. We use `RowManager` to retrieve other properties of a row.

```
RowManager rowManager = myStructureComponents.getRowManager();
for (int i = 0; i < forest.size(); i++) {
  long rowId = forest.getRow(i);
  int depth = forest.getDepth(i);
  StructureRow row = rowManager.getRow(rowId);
  ...
}
```

Note that `row` is never `null`, because Row Manager would through an unchecked exception if a row is not found – this situation is considered a developer's error.

6. Analyze the row and process data

Finally, you get `ItemIdentity` from the row to understand which item does the row show. The items could be anything – issues, folders, users. So even if your structure only contains issues, it is advised to do an extra check.

```
ItemIdentity itemId = row.getItemId();
if (CoreIdentities.isIssue(itemId)) {
  long issueId = itemId.getLongId();
  // process the row!
  ...
}
```


A structure with dynamic content will also contain generators. If you take all the rows, regardless of the item type and use them somewhere, you might stumble upon a generator. To eliminate them from the analyzed forest, add a condition. The same is usually done for "loop markers", which are special items added by extenders to indicate that there's a loop (like cyclic issue links).

```
  ItemIdentity itemId = row.getItemId();
  if (!CoreIdentities.isGenerator(itemId) && !
CoreIdentities.isLoopMarker(itemId)) {
    ...
  }
```

Congratulations! You've successfully implemented forest read-out.

You can adjust this walkthrough for your needs – for example, read a query result, or read only a portion of a forest.

3.3.5 Changing Structure Content

Updating a structure can be done through the same `ForestSource` interface that was used for Reading Structure Content (see page 345). In this article, we're assuming that you've got `forestSource` local variable that you've created according to instructions in the previous article.

Forest Coordinates

To make a change to a forest, you need to be able to point to a specific part of a forest. This is done by using row IDs, which uniquely identify forest rows.

- To point to a specific row in the forest, which you'd like to move or delete, you just use this row's ID.
- To point to a specific position in the forest, where you'd like to insert or move rows to, you need to use row IDs of its neighbors, or *coordinates*:
  - "Under" coordinate is the row ID of the future parent of the inserted row, or zero if the row is placed at the top level.
  - "After" coordinate is the row ID of the future preceding sibling of the inserted row under the same parent, or zero if the row is placed as the first child.
  - "Before" coordinate is the row ID of the future succeeding sibling of the inserted row under the same parent, or zero if the row is placed as the last child.

Applying Forest Action

To make a change, you need to call `forestSource.apply()` method, passing a specific `ForestAction` that you want to apply.

Adding a single row

To add a single row to the forest, use `ForestAction.Add` constructed with the `ItemIdentity` of the item associated with that row.

```
forestSource.apply(new ForestAction.Add(CoreIdentities.issue(10000
), under, after, before))
```

Adding a sub-forest

To add multiple rows in one action, use `ForestAction.Add` that receives an `ItemForest`.

`ItemForest` is a special container that is used to build a temporary forest with temporary rows, having negative row IDs. The class provides information both about the hierarchy of inserted temporary rows (via `Forest`) and a mapping from the temporary row ID to the inserted `ItemIdentity`.

To create an `ItemForest`, you need to use either `ImmutableItemForest` or `ItemForestBuilderImpl`.

```
ItemForest itemForest = new ItemForestBuilderImpl()
  .nextRow(CoreIdentities.textFolder("My Issues"))
  .nextLevel()
    .nextRow(CoreIdentities.issue(10000))
    .nextRow(CoreIdentities.issue(10001))
    .build();
forestSource.apply(new ForestAction.Add(itemForest, under, after,
before));
```

Removing a sub-tree

To remove a row, use `ForestAction.Remove` and pass the row ID being removed.

```
forestSource.apply(new ForestAction.Remove(LongArray.create(100, 1
01, 102)));
```

> ⚠ All sub-rows of the removed rows will be removed as well. If you need to keep them, apply `ForestMove` on them first.

Moving a sub-tree

To move a row with its sub-rows, use `ForestAction.Move`.

You can specify one or more row IDs, which can be from the different parts of the forest. Those rows will be placed one after another at the specified position.

```
forestSource.apply(new ForestAction.Move(LongArray.create(100, 101
, 102), under, after, before));
```

Inspecting the Results

A call to `ForestSource.apply()` will finish successfully if the operation has been completed and throw a `StructureException` otherwise.

You can inspect the returned `ActionResult` to get information about the *effects* of the action (more on effects below).

You can also use `ActionResult.getRowIdReplacements()` – it is a mapping from the temporary row IDs, used when adding rows, to the newly assigned real row IDs, which are now part of the structure.

Effects and Changing Dynamic Structures

You may have noticed that you can apply actions to any forest source, not necessarily a simple structure. It can be a transformed structure, or even a transformed query. A structure can also contain dynamic parts, created or adjusted by generators, and you can try to apply the actions that would affect these parts.

A successful action would produce one or more *Effects* (represented in the `ActionResult` as `AppliedEffect`). In simple case of changing a non-dynamic structure, it would be, unsurprisingly, a structure change. In case the action involves dynamic content, the effects may differ – but the general concept is that, after the effect takes place, the updated (re-generated) structure will reflect the desired action's result.

Here are some examples of the possible effects.

Action	Effect
Adding rows to a static structure	✅ Structure is modified
Moving item X from group A to group B, where groups are provided by a grouper by field F	✅ The value of F for X is changed from A to B
Removing issue X from under issue Y, when previously X was added automatically by a Links Extender using link type L	✅ Link L: YX is deleted
Moving issue upwards when structure is sorted by Agile Rank	✅ Issue's Rank is changed
Adding an issue to an arbitrary JQL query result	⛔ StructureException is thrown – no way to force an issue to be part of a JQL result
Adding issue X under issue Y within the scope of a Links Extender and when issue Y is "static" (not added by the extender)	⛔ StructureInteractionException is thrown – there are two ways to interpret this action

As generators are extensible and can be added by other plugins, the range of possible effects is not limited.

Note that in the last two examples the action is not successful. In the last example, you need to use `ForestSource.apply()` with parameters, which would define whether a generator should process the action or if the issue should be inserted into the static structure.

Concurrency and Atomicity

Each `ForestAction` can be viewed as a separate transaction. It is atomic, meaning that it is either fully successful or fully failed.

There's no way to make a transaction larger. In other words, if you apply two actions to a forest source, it is possible that a concurrent action, done from another thread, is executed in between your two actions.

Permissions

All actions are executed under the "current" user and with all necessary permission checks. Updating a structure requires `EDIT` permission on the structure. Other effects, like changing issue fields, would require `EDIT_ISSUE` permission on the subject issues.

When permissions are insufficient, the action will not succeed and a `StructureException` will be thrown.

When it comes to effects applied by generators, it is a generator's responsibility to check permissions before applying an action. All generators bundled with Structure have strict permission checks.

The current user is generally managed by JIRA and is the same as the user who makes the request. However, you can use `StructureAuth` class to "sudo" to another user or to bypass permission checks altogether.

3.3.6 Loading Attribute Values

You may need to load the same values that Structure shows on the Structure Board, especially if it's a total value, progress value or other Structure-specific value. This is done via `StructureAttributeService`.

About Attributes

One of the core concepts in Structure is the Attribute abstraction. An attribute is something that can provide a value of specific type and meaning for any row in a forest.

For example, a "Summary" attribute would produce the value of Summary field for issues, the name of a folder for folders and a person's full name for users. Some attributes may be applicable only to certain item types and would provide empty value for all other items.

Besides item-based attributes, which provide values that depend only on the item in the forest, there are forest-based attributes – aggregates and "propagates", which are calculated based on the whole forest and items in it.

Forests and Attributes are two main concepts that make up the Structure grid. Looking at the Structure Board, you see Forest in the vertical direction – rows and hierarchy are taken from Forest, and you see Attributes in the horizontal direction – all columns load Attributes from the server and display those values.

General Approach to Loading Values

Let's assume that, after Reading Structure Content (see page 345), you have `StructureComponents` instance and an instance of `ForestSpec` for a forest. We can read a number of attributes for a number of rows by going to `StructureAttributeService.`

1. Figure out which Attributes do you need

The service accepts multiple attribute specs in one request. If you need several attributes calculated – it's better to do that in one request.

```
List<AttributeSpec<?>> attributeSet = new ArrayList<>();
attributeSet.add(CoreAttributeSpecs.KEY);
attributeSet.add(CoreAttributeSpecs.SUMMARY);
attributeSet.add(CoreAttributeSpecs.TOTAL_REMAINING_ESTIMATE);
```

`CoreAttributeSpecs` class contains some of the most popular attributes. However, it's likely that you'll need to build you own attribute specification. For example, to address a numeric JIRA custom field and calculate total of that field based on sub-issues, you'll need the following.

```
AttributeSpec<Number> customField =
  AttributeSpecBuilder.create("customfield", ValueFormat.NUMBER).
params().set("fieldId", 10000).build();

AttributeSpec<Number> customFieldTotal =
  AttributeSpecBuilder.create(CoreAttributeSpecs.Id.SUM,
ValueFormat.NUMBER).params().setAttribute(customField).build();

attributeSet.add(customFieldTotal);
```

2. Figure out which Rows do you need to calculate the Attributes for

For example, this could be all rows in that structure.

```
LongList rows = myStructureComponents.getForestService().
getForestSource(forestSpec).getLatest().getForest().getRows();
```

If you need to create a `LongList` manually, use `LongArray` implementation.

3. Call StructureAttributeService

This service calculates a matrix of values for each row and attribute you specify.

```
VersionedRowValues values = myStructureComponents.
getAttributeService().getAttributeValues(forestSpec, rows,
attributeSet);
```

There is a variation of `getAttributeValues()` method that accepts a `Forest`, rather than `ForestSpec`. It is recommended to use the variant that accepts `ForestSpec` whenever possible, because that variant uses caching.

4. Read out the result

The returned object contains values for all pairs of requested row and requested attribute.

```
for (LongIterator ii : rows) {
  String key = values.get(ii.value(), CoreAttributeSpecs.KEY);
  Number total = values.get(ii.value(), customFieldTotal);
  ...
}
```

3.3.7 Creating and Adding Folders

You may need to create a new folder and add it to a structure.

Folders and generators are items that are managed entirely by Structure add-on, so you'll need to use Structure's services to create the item first, giving you the item identify, and then insert a row into a forest.

Read more about Changing Structure Content (see page 348) for general ideas about updating a structure.

1. Create the Folder entity

```
long folderId = myStructureComponents.getFolderManager().
createFolder(Folder.named("My Stuff").build());
```

The folder is now stored in the database.

2. Define folder's identity

```
ItemIdentity itemId = CoreIdentities.folder(folderId);
```

3. Add folder to structure

```
forestSource.apply(new ForestAction.Add(itemId, 0, 0, 0));
```

3.3.8 Creating Dynamic Structures

Structures may have dynamic content, produced by generators.

Generators can be added to structure and moved around in the same way other items are added, as described in Changing Structure Content (see page 348). A generator will have effect on the whole sub-tree under its parent.

Generators are a separate entities, managed by Structure add-on. So to create a dynamic structure, we need to create a generator first and then insert it into the structure.

1. Create generator instance

You create a generator instance by calling `GeneratorManager`.

```
long generatorId = myStructureComponents.getGeneratorManager().
createGenerator(
  CoreStructureGenerators.SORTER_AGILE_RANK,
  ImmutableMap.of(CoreGeneratorParameters.SORT_DESCENDING, false),
  structureId);
```

Note the third parameter – the generator is "owned" by a structure, so we should pass the ID of the owning structure.

2. Insert generator into the forest

Find parent row under which you'd like the forest to be automated. To apply generator to the whole forest insert generator at the top level by making "under" coordinate zero.

Do not use "after" and "before" coordinates unless you are adding an Inserter.

```
forestSource.apply(new ForestAction.Add(CoreIdentities.generator
(generatorId), under, 0, 0));
```

This is it! Next time you read the contents of this forest source, it will have the results of this generator applied.

3.4 Extending Structure Functionality

You can extend Structure add-on's functionality with your own add-on by using one of the available extension points.

Structure plugin has a lot of extension points. More extensive documentation is coming with the future versions. It will cover the following topics:

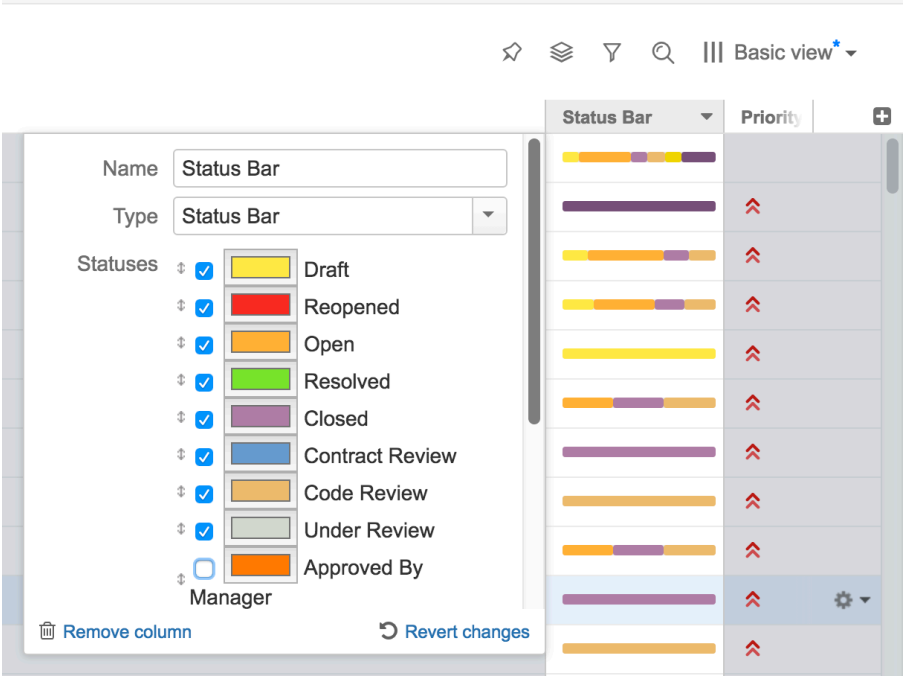
- Adding new item types, which can be used in a structure
- Adding new generators, which can build dynamic structures (Inserters, Extenders, Filters, Groupers and Sorters)
- Adding new attributes, displaying them in the Structure grid or using for sorting or grouping
- Adding new structure templates
- Adding new constraint function to S-JQL
- Adding actions to Manage Structure page
- Adding toolbar elements to the Structure Board

If you're interested in these topics but cannot find documentation or need help, please write to support@almworks.com and we'll provide advice.

3.4.1 Creating a New Column Type

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 (see page 437).



1. 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 attributes 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 attributes provided by Structure (see page) are enough for your column, you can skip the server-side attribute provider. You can also skip the components related to export, if this functionality is not critical. In that case, you can jump straight to the client-side part (see page 362), consulting the other chapters as necessary. For the complete treatment, please continue reading from top to bottom.

2. The Attributes

Before we begin, let's decide which attributes 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" attribute, and because Structure does not provide such an aggregate out of the box, we'll need to write our own.

Secondly, 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.

Thirdly, 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.

3. AttributeSpec for Status Bar

Once we understood which attributes will our JavaScript code need, we have to define or find the appropriate attribute specifications for it.

Our status bar is going to be a new attribute, so we need to create an AttributeSpec. The ID for this spec should be something unique to our add-on. And the format should be a generic `JSON_OBJECT`, because we're going to transfer a bunch of data back to the client rather than just a single value.

```
public static final AttributeSpec<Map<String, Integer>> STATUS_BAR
  = new AttributeSpec("com.almworks.statusbar", ValueFormat.
JSON_OBJECT);
```

We don't need any parameters for this attribute specification – regardless of column configuration, we'll always load the same attribute.

The value will be the map from the Status ID to the number of cases that status is encountered in the sub-tree, including the parent issue.

As for the status ID of the current row, we'll use `CoreAttributeSpecs.STATUS_ID`.

4. Status Bar Attribute

Now that we know which attribute we need to implement, let's write a loader of that attribute. A loader is an instance of AttributeLoader that loads specific attributes for a specific request.

We need to start by looking for the most convenient base class for our loader. It seems that `AbstractDistinctAggregateLoader` is the best, because:

- It is already a loader for an aggregate,
- It addresses the problem of having multiple issues in the same sub-tree more than once – obviously, we don't want to count such issue's Status twice.

As the loader does not have any other parameters, we'll only need a single instance, which we'll keep in a `static final` field.

```
private static final AttributeLoader<Map<String, Integer>> LOADER
= new StatusBarLoader();
```

Our loader will have a dependency on the `CoreAttributeSpecs.STATUS` attribute. Structure will guarantee that the dependency attributes are loaded before our loader is asked to do its calculation.

> It is recommended that aggregates and propagates did not access items directly, but rather declared dependency on other attributes. In this way, if another developer extends the applicability of those dependency attributes to a new type of items, they will immediately get a working aggregate attribute that you wrote, even though you didn't know about the new item type at development time.

The calculation of the result is pretty straightforward. The base class, `AbstractDistinctAggregateLoader`, defines two methods for building recursive value: `getRowValue()` provides a single value for a single row and `combine()` accumulates the provided values.

- As a result for a single row, we create a map with just one record: the issue's status is mapped to 1. If status is missing (as would be the case for non-issues), we just return null.
- As a combination function we will implement map merge that combines counters.
- Finally, we return an immutable copy of the `result` map.

StatusBarAggregate.java

```
  private static class StatusBarLoader extends
AbstractDistinctAggregateLoader<Map<String, Integer>> {
    public StatusBarLoader() {
      super(STATUS_BAR);
    }

    public Set<? extends AttributeSpec<?>>
getAttributeDependencies() {
      return Collections.singleton(STATUS);
    }

    protected Map<String, Integer> getRowValue
(AggregateContext<Map<String, Integer>> context) {
      Status value = context.getValue(STATUS);
      return value == null ? null : Collections.singletonMap
(value.getId(), 1);
    }

    protected Map<String, Integer> combine(Collection<Map<String,
Integer>> values, AggregateContext<Map<String, Integer>> context)
{
      HashMap<String, Integer> r = new HashMap<>();
      for (Map<String, Integer> map : values) {
        if (map != null) {
          for (Map.Entry<String, Integer> e : map.entrySet()) {
            Integer count = r.get(e.getKey());
```

```
            if (count == null) {
              count = 0;
            }
            r.put(e.getKey(), count + e.getValue());
          }
        }
      }
      return r;
    }
  }
```

5. Attribute Provider

Attribute providers are registered as modules in the plugin descriptor, and their instances are created by the JIRA module system. If the attribute provider "recognizes" the attribute specification and can serve it, it must return a non-null `AttributeLoader` instance. Because our `StatusBarLoader` 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 loaders for each call. The returned loader will then be called once for each item needed to display the Structure grid (or its visible part).

StatusBarDataProvider.java

```
public class StatusBarAttributeProvider implements
AttributeLoaderProvider {
  private static final AttributeSpec<Map<String, Integer>>
STATUS_BAR = new AttributeSpec("com.almworks.statusbar",
ValueFormat.JSON_OBJECT);
  private static final AttributeLoader<Map<String, Integer>>
LOADER = new StatusBarLoader();

  public AttributeLoader<?> createAttributeLoader(AttributeSpec<?
> attributeSpec, @NotNull AttributeContext context)
    throws StructureProviderException
  {
    if (STATUS_BAR.getId().equals(attributeSpec.getId())) {
      return LOADER;
    }
    return null;
  }
}
```

When the data provider is ready, we register it in the plugin descriptor.

atlassian-plugin.xml

```
<structure-attribute-loader-provider key="alp-sbcolumn" name="attr
ibute-loader:Status Bar Column"
                                    class="com.almworks.jira.
structure.sbcolumn.StatusBarAttributeProvider"/>
```

6. 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 (see page 437) 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.

6.1. API Overview

The whole API is accessible through the `window.almworks.structure.api` (see page 422) 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` (see page 434)	The **column type** is the gateway between Structure and your code. The column type is registered with Structure and has the following responsibilities: • 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` (see page 424)	

Class or Function	Description
	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 one or more attributes to be downloaded from the server for the rendered rows.
`api.ColumnConfigurator` (see page 430)	The **configurator** is responsible for the column configuration panel as a whole. Its most important task is to create `ColumnOption` instances.
`api.ColumnOption` (see page 432)	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.

6.2. 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).

Do not confuse column specifications with attribute specifications. A column is a higher-level concept and may require multiple attributes (as is the case with our Status Bar column).

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.

6.3. 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 `AttributeContext` 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 (see page 139) 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()`	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.
`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)`	

Method	Description
	Returns the metadata object associated with the given `key`. See the section below for the description of metadata.

In our column we'll use `context.getMetadata()`.

6.4. 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 attributes. 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 for 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.

6.5. Column

The `api.Column` class is responsible for rendering the cells of the Structure grid. Please refer to the Column class reference (see page 424) 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.

`collectRequiredAttributes()` always requests the status bar aggregate data from `StatusBarAttributeProvider`. If the "Include itself" option is off, it additionally requests the status ID of the current issue, which is provided by Structure as `{id:'status', format:'id'}`. The main attributes are also available from `require('structure/widget/attributes/Attributes')` object.

`getSortAttribute()` is used to specify the attribute for sorting when the user clicks on the column header.

`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.

6.6. ColumnConfigurator

The `api.ColumnConfigurator` (see page 430) 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` (see page 371) and column groups (see page 373) 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.

6.7. ColumnOption

Each `api.ColumnOption` (see page 432) 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.

6.8. ColumnType

The `api.ColumnType` (see page 434) 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 (see page 437) in your favorite editor and scroll to the `StatusBarType` class definition.

The `getMetadataRequests()` method declares the column-level metadata request to load the available issue statuses from JIRA. See Requesting and Using Metadata (see page 366) 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.

6.9. 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.

6.10. Web Resources and Contexts

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"/>
  <context>structure.widget</context>
</web-resource>
```

We use `structure.widget` web resource context to make our JavaScript (and CSS, if we had any) load on Structure Board. It also works for the Structure's Dashboard Gadget. However, if you'd like your column to work on other pages – Project page, Issue page or Agile Board page, you need to include other web contexts too – see Loading Additional Web Resources For Structure Widget (see page 378).

7. Export Renderers

Any structure 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 different 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 javadocs 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 would recognize the column specification and return an **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 **export context** and **export row** instances will provide all the data, including the required attributes.

Speaking of the interfaces that must be implemented, `ExportRendererProvider` is analogous to `AttributeLoaderProvider`, and `ExportRenderer` is a mixture of `AttributeLoader` and the client-side `Column` (see page 424).

7.1. 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.

7.2. 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.
- Creates and returns an instance of the `StatusBarRenderer` inner class, passing it the column name and parameters.

The renderer has `prepare()` method that lets it specify which attributes it will need loaded to do the export. Like in `StatusBarColumn`, we request our histogram-based custom attribute and status for the current row.

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 attribute values from the context.
- 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="expor
t-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>
```

7.3. 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 `prepare()` and `configureColumn()` methods are 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>
```

3.4.2 Creating a New Synchronizer

Structure comes with a number of bundled synchronizers (see page 219), but you can add another synchronizer to the system, allowing Structure users to install it on structures and run export / import.

1. Implement StructureSynchronizer

Create your implementation of StructureSynchronizer interface. Use AbstractSynchronizer as the base class.

2. Define structure-synchronizer Module

Add structure-synchronizer (see page 382) module to your `atlassian-plugin.xml`, referring to your implementation of the `StructureSynchronizer`.

3. 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.

You can download the sources zip with the sample synchronizers at API Usage Samples (see page 437) page.

3.4.3 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 (see page 15) is displayed, use `structure.widget` web resource context and possibly a few others.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 (see page 357). You'll need to use the Structure JavaScript API (see page 422) and register the web resource with your JavaScript code as a widget extension.

Using Web Resource Contexts

You can add JavaScript or CSS to the Structure widget by adding 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 following table lists all web resource contexts related to pages where Structure Widget can possibly be shown.

Web Resource Context	Used on...
`structure.widget`	Structure Board, Structure Gadget
`jira.view.issue`	Issue Page, Issue Navigator in details view
`gh-rapid`	Scrum and Kanban boards from JIRA Software
`jira.browse.project`	Project page (including Structure tab)
`structure.printable`	Printable Structure page

To have your code present on every page where a Structure widget can possibly be shown, include all these resources. You usually don't need to include `structure.printable` though, unless you have some special rules for printing.

Sample snippet from `atlassian-plugin.xml`:

```
<web-resource key="custom-field-resource" name="My Custom Field
Web Resource">
  <resource type="download" name="custom-field-resource.js"
location="js/myplugin/custom-field-resource.js"/>
  <context>structure.widget</context>
  <context>jira.view.issue</context>
  <context>gh-rapid</context>
  <context>jira.browse.project</context>
</web-resource>
```

3.5 Accessing Structure Data Remotely

Structure plugin provides REST API, which is primarily used by the structure widget (see page 33). 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 (see page 389).

3.6 Reference

3.6.1 Structure Developer Reference

3.6.2 Structure Java API Reference

Structure API is work in progress. You will find that some of the packages are documented less than others, and some are not documented yet.

We're constantly working on the API improvements and documentation and will make the javadocs and other parts of the documentation more complete with every release.

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 (see page 380) 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
16.4.0 Javadocs	JIRA 7.1+	4.1.0+	"[16.4,17)"	2017-06-19
16.3.0 Javadocs	JIRA 7.1+	4.0.0+	"[16.3,17)"	2017-04-26
16.2.0 Javadocs	JIRA 7.0+	3.6.0+	"[16.2,17)"	2017-04-03
	JIRA 7.0+	3.5.0+	"[16.1,17)"	

Version	Supported JIRA Versions	Supported Structure Versions	OSGi Import Version	Release Date
16.1.0 Javadocs				2017-01-26
16.0.0 Javadocs	JIRA 7.0+	3.4.0+	"[16,17)"	2016-12-07
Structure API version 16.0.0 is the first public API version for Structure 3.x. For older API versions compatible with Structure 2.x, see Previous API Versions.				

Javadocs for the Latest Version — Java API documentation for the latest API version.

To see how to include the API in your project dependencies, read about Accessing Structure from JIRA Plugin (see page 337).

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 -Dfile=structure-api-16.4.0.jar -
DpomFile=structure-api-16.4.0.pom
```

Name	Version	Date
structure-api-16.4.0-javadoc.jar	1	2017-06-20 00:02
structure-api-16.4.0-sources.jar	1	2017-06-20 00:02
structure-api-16.4.0.jar	1	2017-06-20 00:02
structure-api-16.4.0.pom	1	2017-06-20 00:02

3.6.3 Structure Plugin Module Types

The following module types are added by the Structure plugin:

- structure-synchronizer (see page 382) defines a new synchronizer.
- structure-attribute-loader-provider (see page 384) lets you provide the data for new column types in the Structure widget.
- structure-export-renderer-provider (see page 384) lets you export new column types to printable HTML and Excel files.
- structure-item-type (see page 385) lets you define a new type of items, which can be used in structures.
- new-structure-template (see page 386) lets you add templates for new structures.
- structure-query-constraint (see page 387) allows adding new functions to S-JQL language.
- Generator Modules (see page 387) let you add generators to the Automation subsystem. The following module types are included:
  - `structure-inserter`
  - `structure-extender`
  - `structure-filter`
  - `structure-grouper`
  - `structure-sorter`

structure-synchronizer

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"/>
  </structure-synchronizer>
```

Element	Required	Description
`structure-synchronizer`	Yes	The module descriptor.
`@key`	Yes	Unique module key within the plugin.
`@order`	Yes	Order of the synchronizer among other synchronizers, whenever a list of synchronizers is present.
`@class`	Yes	The class that implements the synchronizer. Must implement StructureSynchronizer. It is recommended to extend AbstractSynchronizer.
`label`	Yes	The name of the synchronizer.
`description`	No	Description of the synchronizer.
`rules`	No	The text that is shown at the top of the synchronizer configuration page. Could be a large text.
`resource [@name="form"]`	Yes	A velocity template that contains the form for the synchronizer parameters.

structure-attribute-loader-provider

You can use this module to add your support for attributes, either new or already existing, to Structure. The attributes are used by Structure Widget columns, by exporters and by generators.

Example

```
<structure-attribute-loader-provider key="provider-key"
    class="com.company.your.plugin.attribute.MyAttributeProvider"/
>
```

Element	Required?	Description
`structure-attribute-loader-provider`	Yes	The module descriptor.
`@key`	Yes	The unique identifier of the plugin module.
`@name`	No	The human-readable name of the plugin module.
`@class`	Yes	The class that implements the data provider. Must implement AttributeLoaderProvider.

structure-export-renderer-provider

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	Required?	Description
`structure-export-renderer-provider`	Yes	The module descriptor.
`@key`	Yes	The unique identifier of the plugin module.
`@name`	No	The human-readable name of the plugin module.
`@class`	Yes	The class that implements the renderer provider. Must implement ExportRendererProvider.
`column-key`	No	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`	No	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 considered generic – such a provider will be consulted for every column not served by a format-specific provider.

structure-item-type

This module type lets you declare a new item type. Items of that type can then be used in structures.

Example

```
<structure-item-type key="type-book" name="itemtype:Book"
    class="com.mycompany.structure.books.BookItemType"/>
```

Element	Required?	Description
`@key`	Yes	The unique identifier of the plugin module. Full module key will define the `itemType` part of ItemIdentity.
`@name`	No	The human-readable name of the plugin module.
`@class`	Yes	The class that implements the support for the item type. Must implement StructureItemType.

new-structure-template

New Structure Template module allows you to add templates to the Create Structure dialog.

Example:

```
<new-structure-template key="big-template"
    class="com.mycompany.structure.template.bigtemplate"
    name="New Structure Template: Big Template">
  <label key="com.mycompany.template.big-template.label"/>
  <description key="com.mycompany.template.big-template.
description"/>
  <resource type="download" name="icon.png" location="css
/structure/templates/big@2x.png"/>
  <resource type="velocity" name="step1" location="templates
/structure/big/step1.vm"/>
  <resource type="velocity" name="step2" location="templates
/structure/big/step2.vm"/>
</new-structure-template>
```

Element	Required?	Description
`@key`	Yes	Module key.
`@name`	No	The name of the module for JIRA administrators.
`@class`	Yes	The class that implements the template, must implement NewStructureTemplate.
`label`	Yes	

Element	Required?	Description
		The name of the template as it appears in the Create Structure dialog.
`description`	No	Description of the template.
`resource [@type=velocity]`	No	Any number of HTML templates used by your code to render wizard steps.
`resource [@type=download]`	No	Any number of downloadable images or other resources used by your template.

structure-query-constraint

Structure Query Constraint module allows you to define an additional constraint function that can be used in S-JQL.

For example, `folder()` function explained in S-JQL Reference (see page 258) is implemented with a `structure-query-constraint`.

Example:

```
<structure-query-constraint key="constraint-foo"
                            class="com.mycompany.structure.
FooConstraint"
                            name="Structure Query Constraint: foo"
                            fname="foo"/>
```

Element	Required?	Description
`key`	Yes	Module key.
`name`	No	Module name for the JIRA administrator.
`class`	Yes	Class that implements StructureQueryConstraint.
`fname`	Yes	Function name, must be unique throughout the system.

Generator Modules

There are five modules, one for each type of generators, that work in the same way:

- `structure-inserter`
- `structure-extender`
- `structure-filter`
- `structure-grouper`
- `structure-sorter`

Each module allows declaring a generator of a specific type. When a plugin with a generator module is installed, you get the ability add those generators to structures.

Example

```
<structure-extender
    key="extender-examples" name="extender:Examples" description="
Examples extender"
    class="com.mycompany.structure.examples.ExamplesExtender">
  <label key="com.mycompany.examples.extender.label"/>
  <icon spanClass="s-fa s-fa-link"/>
  <dialog-title key="com.mycompany.examples.extender.dialog-title"
/>
  <resource type="velocity" name="form" location="/templates
/example/extender-examples.vm"/>
  <resource type="velocity" name="summary" location="/templates
/example/extender-examples-summary.vm"/>
</structure-extender>
```

Other types of generators are declared in the same way.

Element	Required?	Description
`@key`	Yes	The unique identifier of the generator. Full module key will a part of generator specification, defining the automation.
`@name`	No	The name of the module for the JIRA administrator.
`@description`	No	Description of the module for the JIRA administrator.
`label`	Yes	The name of the generator as it appears to the user.

Element	Required?	Description
`icon`	No	The icon for the generator that will be shown whenever the generator row is displayed. See below for details.
`dialog-title`	No	The title of the dialog that is used to edit the generator
`resource [@name=form]`	No	Form template that will be used for editing the generator's parameters
`resource [@name=summary]`	No	Form template that will be used to display the generator as a row in a structure

Generator Icons

The icon for the generator is defined using CSS classes. If you're using your own icons, make sure the appropriate CSS styles are loaded everywhere Structure can be used (see Loading Additional Web Resources For Structure Widget (see page 378)).

You can also use the standard icons used by bundled generators:

Generator Type	Icon Classes
Inserter	`s-fa s-fa-plus`
Extender	`s-fa s-fa-link`
Filter	`alm alm-group`
Grouper	`s-fa s-fa-filter`
Sorter	`alm alm-sort-asc`

3.6.4 Structure REST API Reference

Structure REST API is under development. The functionality available through REST is sometimes not complete, but it allows to work with the structures.

API version 2 is also not stable, although we're not seeing major changes coming to the main resources.

> Both version 1 and version 2 of the REST APIs have been driven by the needs of Structure Widget. We're currently developing a higher-level API specifically for integrations rather than for the product itself. Let us know at support@almworks.com if you'd like to contribute or get preliminary access to that API.

General Notes

API Versions

As of Structure version 3.4, there are two versions of the REST API – `1.0` and `2.0`. Some of the REST resources are exposed through version 1.0 and some through version 2.0.

Version 1.0 is stable and we don't plan to change it. It comes from Structure 2 and largely remains the same as in Structure 2.x versions. Some of the resources may become deprecated as we replace them with the newer versions.

Version 2.0 is not stable and is being developed along with the product. That means that you can use it, but you need to test your integration every time you upgrade. We are also going to publish API changes in the release notes.

REST Resource Addresses

Structure REST API resources have the URL

```
BASEURL/rest/structure/VERSION/NAME
```

where BASEURL is the base JIRA address (`http://localhost:2990/jira` being standard base URL for development environment), VERSION is the version of the API (either 1.0 or 2.0) and NAME is the name of the resource. For each documented resource there's an indication about its API version.

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

- Structure Resource (see page 391) is used to create and manage structures (but not the content)
- Forest Resource (see page 411) is used to retrieve and update forests (a structure's content)
- Item Resource (see page 413) is used to create and update items (issues, folders and, possibly, items of other types)
- Value Resource (see page 418) is used to retrieve attribute values for a given forest

Structure Resource

This page describes resources with which you can list (see page 396), create (see page 400), read (see page 403), update (see page 406), and delete (see page 409) structures. Structures contain general information (see page 192) such as name and permissions, but not the hierarchy itself. Issue hierarchy is accessed through the Forest Resource (see page 411). This page also documents structure shape (see page 391) and its fields (see page 392), and the error entity (see page 395) that may be returned in case of the REST API user error.

Structure resource belongs to **version 2.0** of the API.

`/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 (see page 391)
- Structure Fields (see page 392)
- Permission Rules (see page 394)
- Error Entity (see page 395)

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 (see page 391)

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 (see page 192):

`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 (see page 194) access level to the structure, otherwise absent.
`editRequiresParentIssuePermission`	`true` if the Require Edit Issue Permission on Parent Issue (see page 195) flag is set on this structure, otherwise absent.
`permissions`	The list of structure permission rules (see page 194). Present only if the user has Control (see page 194) 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 (see page 192) 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 content of a structure, i.e. its hierarchy of items, can be read or modified using Forest Resource (see page 411).

Top (see page 391)

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 (see page 194) access level to the referenced structure.

Top (see page 391)

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 (see page 391)

Structure Resources

GET /structure

```
GET $baseUrl/rest/structure/2.0/structure
GET $baseUrl/rest/structure/2.0/structure?
name=$name&permission=$permission&withPermissions=$withPermissions
&withOwner=$withOwner&limit=100
```

A list of all structures visible to the REST API user. Optionally, the result can be filtered by name or user's access level. 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 (see page 306). The returned list contains only structures to which the REST API user has at least View (see page 194) access level.

Request

Query parameters:

`name`	If present, the returned list will contain only structures which names contain the specified string (case insensitive).
`permission`	If present, the returned list will contain only structures to which the REST API user has the specified access level (see page). 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.

`withPermissions`	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`.
`archived`	If `true`, the returned list can also contain archived structures. Default is `false`.
`limit`	If specified, must be a number. Defines the maximum number of structures to return.

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, sorted by name.	`application/json`, `application/xml`

Example 1: all structures

```
GET $baseUrl/rest/structure/2.0/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/2.0/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: 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 4: 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
```

```
<?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 (see page 313) operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top (see page 391)

POST /structure

```
POST $baseUrl/rest/structure/2.0/structure
```

Create (see page 193) an empty structure by POSTing to this resource.

Who can access this resource

Only logged in users who have access to the Structure Plugin (see page 306) and a permission to create structures (see page 307).

Request

Request entity should contain the new structure (see page 392). 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 (see page 394).

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/2.0/structure
```

Request entity	Response entity
`{ "name":" Test plan"`	`{ "id": 104,`

Request entity	Response entity

<pre>}</pre>	<pre> "name": "Test plan",
	  "description": "",
	  "permissions": [],
	  "owner": "user:admin"
	}</pre>

Example 2: structure with some permissions

```
POST $baseUrl/rest/structure/2.0/structure
```

Request entity	Response entity

<pre>{
  "name":"Structure with some permissions",
  "editRequiresParentIssuePermission":"true",
  "permissions":[
  {
  "rule":"apply",
  "structureId":102
  }
  ]
}</pre>
	<pre>{
	  "id": 105,
	  "name": "Structure with some permissions",
	  "description": "",
	  "editRequiresParentIssuePermission": true,
	  "permissions": [
	  {
	  "rule": "apply",
	  "structureId": 102
	  }
	  ],
	  "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`	`application/json`, `application/xml`

	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.	
`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 (see page 313) operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top (see page 391)

GET /structure/{id}

```
GET $baseUrl/rest/structure/2.0/structure/$id
GET $baseUrl/rest/structure/2.0/structure/$id?
withPermissions=$withPermissions&withOwner=$withOwner
```

This resource allows to obtain structure details (see page 392) 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 (see page 306). To access the particular structure, the user has to have at least View (see page 194) access level.

Request

Path parameter:

`id`	the ID of the structure

Query parameters:

`withPermissions`	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 (see page 194) 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/2.0/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/2.0/structure/102?withOwner=true&withP
ermissions=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/2.0/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 (see page 194) 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`
	If an internal error has occurred while processing this request.	

`500 Internal Server Error`		
`503 Service Unavailable`	If Structure Plugin is stopped at the time of request. For example, the Restore (see page 313) operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top (see page 391)

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 (see page 306) and Control (see page 194) permission on this structure.

Request

Request entity should contain those structure fields (see page 392) 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 (see page 394).

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",
      "level":"view"
    },
    {
      "rule":"set",
      "subject":"group",
      "groupId":"jira-users",
      "level":"edit"
    },
    {
      "rule":"set",
      "subject":"group",
      "groupId":"jira-administrators",
      "level":"admin"
    }
  ]
}
```

Example 2: changing permission rules

```
POST $baseUrl/rest/structure/1.0/structure
```

Request entity

```
{
  "permissions":[
    {
      "rule":"set",
      "subject":"group",
      "groupId":"jira-users",
      "level":"edit"
    },
    {
      "rule":"apply",
      "structureId":101
    }
  ]
}
```

Response entity

```
{
  "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"
}
```

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	`application/json`, `application/xml`.

	rule that fails validation. Response entity contains error. Responses to problems with an *apply* rule usually have `structureId` to indicate the invalid reference.	
`403 Forbidden`	If REST API user is not logged in, does not have permissions to access Structure Plugin, or does not have Control (see page 194) 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 (see page 313) operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top (see page 391)

DELETE /structure/{id}

Deletes (see page 208) the designated structure.

Who can access this resource

Only logged in users who have access to the Structure Plugin (see page 306) and Control (see page 194) 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 (see page 194) 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 (see page 194) 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 (see page 313) operation may be in progress.	

Other return codes are possible under the normal rules of HTTP communication.

Top (see page 391)

Forest Resource

Forest Resource is responsible for serving forests and forest updates and receiving the forest actions (change commands) from the client.

Retrieving Forest

Request

```
GET $baseUrl/rest/structure/2.0/forest/latest?s=$forestSpec
POST $baseUrl/rest/structure/2.0/forest/latest
```

Returns the hierarchical issue list (forest) of the specified structure.

Parameters:

$forestSpec	*required*	The URL-encoded JSON representation of ForestSpec. See also: RestForestSpec.
POST content	*required*	While GET method is preferred, POST is more robust because there's no risk of exceeding URL length with large forest specifications. The content is the same JSON object (but not URL-encoded, obviously).

Example:

```
GET /rest/structure/2.0/forest/latest?s={%22structureId%22:113}
```

Retrieves latest forest for structure #113.

Response

```
{
  "spec":{"structureId":113},
  "formula":"10394:0:4/356,10332:0:14707,10374:1:5/240,10348:2:
14717",
  "itemTypes":{
    "4":"com.almworks.jira.structure:type-generator",
    "5":"com.almworks.jira.structure:type-folder"
  },
  "version":{
    "signature":-1659607419,
    "version":1
  }
}
```

In this reply, the most important part is "formula", which contains serialized information about the forest.

Each component (delimited by comma) represents a row and looks like this: 10374:1:5/240. In this example, the numbers are:

- 10374 is the row ID,
- 1 is the row depth,
- 5/240 is the item identity.

If the row contains an issue, it's just issue ID, otherwise it has the format of <item type>/<long item id>, or <item type>//<string item id>. Item type is a number, which is expanded in the "itemTypes" map in the reply.

Changing Forest

To change a forest, you POST one or more change actions to `/forest/update` resource. Each action is a serialized version of `ForestAction` – for more information about the actions, see Changing Structure Content (see page 348).

```
POST $baseUrl/rest/structure/2.0/forest/update
```

Parameters:

```
{
```

```
  "spec": { "structureId": <id> }, // use structure ID
  "version": { "signature": <signature>, "version": <version> },
// use last seen signature and version
  "actions": [
     {
         "action": "add",
         "under": 0,        // at the top level
         "after": 123,      // after row ID 123 (not issue iD!)
         "before": 456,     // before row ID 456
         "forest": "-100:0:10001"   // insert issue 10001, -100
is the temporary row ID which will be mapped into the real row ID
when the method returns
     },
     {
         "action": "move", // works like previously, only row IDs
instead of issue IDs
         "rowId": 123,
         "under": 456,
         "after": 0,
         "before": 124
     },
    {
       "action": "remove",
       "rowId": 442
     }
  ]
}
```

Item Resource

Item resource is used to create new items and update existing items.

Creating a New Item

The following request is used to create a new item (issue, folder or other type) and insert it into a forest.

```
POST $baseUrl/rest/structure/2.0/item/create
```

This request should upload a specification of the creation action and coordinates of where to put the result.

Example

```
{
```

```
  "item": {
    "type": "com.almworks.jira.structure:type-folder",
    "values": { "summary": "New folder name" }
  },
  "forest": {
    "spec": { "structureId": 128 },
    "version": {
      "signature": 0,
      "version": 0
    }
  },
  "items": {
    "version": {
      "signature": 0,
      "version": 0
    }
  },
  "rowId": -100,
  "under": 0,
  "after": 0,
  "before": 0,
  "parameters": {}
}
```

Parameters

Parameter (see example above)	Meaning
`item`	Defines the item being created.
`item.type`	Item type (complete key of the module that provides this item's main functionality.) Use `com.almworks.jira.structure:type-folder` for folders and `com.almworks.jira.structure:type-issue` for issues. See also: CoreItemTypes
`item.values`	A set of values for the new item. The specific fields depend on the item. For a folder, it is "summary". For other items, see examples below.

Parameter (see example above)	Meaning
`forest.spec`	Forest specification of the forest that will receive the new item. See ForestSpec and RestForestSpec.
`forest.version`	Last known version of the forest. The reply to this call will contain the update to that version. Use zero version (as in example) to receive full forest.
`items.version`	Last known version of instance items set. The reply to this call will contain an update to the known items. Use zero version (as in example) to receive full update.
`rowId`	Temporary ID assigned to the created issue. Must be negative. You can use `-100` in most cases.
`under` / `after` / `before`	Forest coordinates to insert the new item into. See Forest Resource (see page 411).

Specific parameters for main item types

Folder

This is the example of `item` parameter for a new folder:

```
"item": {
  "type": "com.almworks.jira.structure:type-folder",
  "values": { "summary": "New folder name" }
}
```

The only parameter sent is the folder name.

Issue

This is the example of `item` parameter for a new issue:

```
"item": {
  "type": "com.almworks.jira.structure:type-issue",
```

```
  "values": {
    "issue": {
      "summary": "issue summary"
    },
    "pid": 10000,
    "issuetype": "3",
    "mode": "new",
  }
}
```

The above are the minimal fields needed to create a new issue. Note that `pid` is a number, but `issuetype` is a string.

Reply Example

The following is an example of a reply.

```
{
  "successfulActions": 1,
  "itemId": "com.almworks.jira.structure:type-issue/10100",
  "oldRowIds": [-100],
  "newRowIds": [61],

  "forestUpdates": [...],
  "itemsUpdate": {...}
}
```

Most important fields are `itemId` and `newRowIds`. More on the return fields:

Field	Explanation
`successfulActions`	A number of actions successfully performed by the server. In this case, it's either 0 or 1.
`itemId`	The ID of the newly created item. See ItemIdentity.
`oldRowIds` / `newRowIds`	Provides mapping from the temporary row IDs used for uploading the action and the real row IDs obtained after the item was inserted.
`forestUpdates`	Changes to the forest since the version passed in the request.
`itemsUpdate`	Changes to the items set since the version passed in the request.

Updating an Existing Item

The following request is used to update an existing item (issue, folder or other type).

```
POST $baseUrl/rest/structure/2.0/item/update
```

Example of the request:

```
{
  "item": {
    "itemId": "10000",
    "values": {
      "summary": "New Summary"
    }
  },
  "items": {
    "version": { "signature": 0, "version": 0 }
  },
  "forest": {
    "spec": {
      "type": "clipboard"
    },
    "version": { "signature": 0, "version": 0 }
  }
}
```


Note that although the update does not depend on the forest, the low-level API in the current version requires the request to specify a forest spec and known version of items stream. If you don't need to maintain up-to-date items cache and not interested in updates to a forest where the item is located, just use empty version in **items** field and "clipboard" forest spec – like in this example.

Parameters

Parameter (see example above)	Meaning
	The ID of the item.

Parameter (see example above)	Meaning
`item.itemId`	If it is just a number, like in the example, it is an issue ID. Note that it is still a **String** value that contains issue ID. Instead of a number, it can be a canonical notation of an ItemIdentity. For example, to update a folder, use `"com.almworks.jira.structure:type-folder/123"` where 123 is the folder ID.
`item.values`	A map of values to be updated. The keys are the same as when the item is created. For updating a folder, use `"summary"`.
`items.version`	Known version of the items stream. The response will contain an update based on that number. Use zeroes, as in example, when updated is not needed.
`forest.spec` and `forest.version`	Monitored forest spec and known version of that forest. The response will contain a forest update based on those values. When not needed, use a simple forest (like clipboard in this example) and zeroed version.

Reply

The reply is similar to the reply from calling `/create` method, defined above. A positive HTTP status tells that the item has been updated. There is no `"itemId"` in the response.

Value Resource

Value Resource is used to retrieve values of attributes for rows in a given forest.

To learn more about attributes, see Loading Attribute Values (see page 352).

To retrieve values from Structure, you need a few things first:

- A forest specification (`forestSpec`) for the displayed forest – same as the one used in Forest Resource (see page 411). Forest specification is needed even if the values do not depend on the forest.

- A list of row IDs for which the values should be loaded. Row IDs can be retrieved from Forest Resource before calling Value Resource.
- A list of attribute specifications. Some examples are given below.

Loading Values

To load values use the following call

```
POST $baseUrl/rest/structure/2.0/value
```

The request should come with JSON payload that specifies which values you are interested in.

Example

```
{
  "requests": [
    {
      "forestSpec": {
        "structureId": 123
      },
      "rows": [
        1820,
        1842,
        2122
      ],
      "attributes": [
        {
          "id": "summary",
          "format": "text"
        },
        {
          "id": "key",
          "format": "html"
        },
        {
          "id": "progress",
          "format": "number",
          "params": {
            "basedOn": "timetracking",
            "resolvedComplete": true,
            "weightBy": "equal"
          }
        }
      ]
    }
```

```
    ]
}
```

As you see in this example, a request body may contain one or more request, each for a specific matrix of several rows and several attributes. A value for each pair of a row and an attribute will be calculated.

Parameters

Parameter	Meaning
`requests[i].forestSpec`	Forest specification that produces the forest from which the rows are taken.
`requests[i].rows`	Array of row IDs for which values should be loaded.
`requests[i].attributes`	Array of attribute specifications that should be loaded for each row.

The example shows three attributes being loaded – plain text Summary, html-formatted Key and Progress based on time tracking. For more information about available system attributes, see javadocs for AttributeSpec and CoreAttributeSpecs.

There is a simple way to learn the attribute spec that you need.

1. Configure a column that shows the needed value on the Structure Board.
2. Use your browser's Developer Tools and open Network tab.
3. Reload structure.
4. Look for a request to `/value` URL and see its input. Use JSON formatters for convenience.

Response

The response will contain one or more matrices with values for each pair of requested row and attribute. A list of rows is given separately. Then, for each requested attribute, a list of values is given.

```
{
```

```
  "responses": [
    {
      "forestSpec": {
        "structureId": 123
      },
      "rows": [
        1820,
        1842,
        2122
      ],
      "data": [
        {
          "attribute": {
            "id": "summary",
            "format": "text"
          },
          "values": [
            "Issue 1",
            "Folder 2",
            "Some Other Item 3"
          ],
          "trailMode": "INDEPENDENT",
          "trails": [ "", "", "" ]
        }
      ],
      "forestVersion": {
        "signature": -1385959428,
        "version": 1
      }
    }
  ],
  "itemTypes": {},
  "itemsVersion": {
    "signature": -558220658,
    "version": 1
  }
}
```


Parameters

Parameter	Meaning
`responses[i].forestSpec`	Requested forest spec, from which the rows are taken.
`responses[i].rows`	A list of row IDs for which the values are provided.

Parameter	Meaning
`responses[i].data[j].attribute`	The attribute specification for which the following values are calculated.
`responses[i].data[j].values`	Array of values. The value at k-th place corresponds to the row at k-th place in `responses[i].rows`.

⚠ If you are receiving value in any format other than `html`, you need to html-escape that value before adding it to the web page.

3.6.5 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.

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 (see page 424).

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 (see page 209) specification, along with the type-specific parameters and column name.

Parameters

`type`	*Object*	A `ColumnType` instance, implementing a specific column type – see ColumnType Class (see page 434)
`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() (see page) method.

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 (see page 364) 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 (see page 363) 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.getAttributeValue()	returns current row's attribute value
renderingParameters.getRowId()	returns current row's id
renderingParameters.getItemId()	return current row's item id

Example

```
var Template = require('almworks/util/Template');
var cellTemplate = new Template('<span class="acme-field">
{awesomefield}</span>');
getCellValueHtml: function(rp) {
  return cellTemplate.renderHtml({ awesomefield: rp.
getAttributeFieldValue({id: 'com.acme.awesome-data', format: 'text
'}) });
}
```

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.getAttributeValue()	returns current row's attribute value
renderingParameters.getRowId()	returns current row's id
renderingParameters.getItemId()	return current row's item id

collectRequiredAttributes(attributeSet)

Lets column request attributes that are needed for rendering. The attributes are provided on the server side by AttributeLoaderProvider.

Parameters

attributeSet.requireAttribute (attributeSpec,forestSpec)	Method for collecting required attributes. Parameters are: • `attributeSpec` is the attribute specification object • `forestSpec` is the forest specification for the forest, from which attribute should be loaded **(optional)**

About Attribute Specs

AttributeSpec defines the attribute and format to be loaded. See Loading Attribute Values (see page 352) for more information on attributes.

Some of the attributes are shown below. You can also define your own attribute, calculate it on the server side and request from your column.

About Forest Spec

Forest specification is optional. When used, it allows you to get attribute value from a different forest – however, it must be related to the forest being displayed, otherwise it will not have the same rows.

For example, you can specify a forest specification with some transformation to display values from there in the untransformed forest. There are also two special values for `forestSpec`:

- `'displayed'` is the default value, meaning "use the forest that is being displayed"
- `'unfiltered'` means "use the same forest, but remove all filters that are coming at the end of transformation chain"

Example

```
collectRequiredAttributes: function(attributeSet) {
  attributeSet.requireAttribute({id: 'key', format: 'text'});
  attributeSet.requireAttribute({
    id: 'sum',
    format: 'number',
    params: {
      id: 'customfield',
      format: 'number',
      params: {
        fieldId: 10010
      }
    }
  }, 'unfiltered');
  attributeSet.requireAttribute({id: 'com.mycompany.work-stats',
format: 'json'});
}
```

Some of the attributes provided by Structure:

Attribute Spec	Example	Description
`{id: <jira-field-id>, format: 'html'}`		The HTML representation of a JIRA issue field value, as seen on the issue page or in the Issue Navigator. Structure allows non-issue items also have these values. `<jira-field-id>` is the common name for the JIRA's standard field id. This attribute does not load custom fields.

Attribute Spec	Example	Description
`{id: 'customfield', format: 'html', params: { fieldId: <field-numeric-id> }}`		HTML representation of a custom field value.
`{id: 'project', format: 'id'}`		Project ID for the issues. The `id` format means either a string or a number, depending on what is being used for identifying the object.
`{id: 'editable', format: 'boolean'}`		Boolean value telling whether the item can be edited by the user.

See also CoreAttributeSpecs for examples of bundled attributes.

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 (see page 366) 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
    }
  };
}
```

getSortAttribute()

Returns attribute specification for sorting when the user clicks on the header. If `null` is returned (the default), the clicking this column header does not result in added sorting transformation.

isSortDescendingByDefault()

If returns `true` the initial direction of the sorting will be descending.

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() (see page 435) 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() (see page 424) for reference.

getMetadataRequests()

Returns a JavaScript object specifying the metadata needed by this configurator to set up the UI. See Requesting and Using Metadata (see page 366) 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 (see page 432).

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() (see page 431) 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 || '4
2') : '');
    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$) (see page 433) 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 (see page 432) 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 (see page 366) 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 (see page 366) 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 (see page 366) 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 (see page 366) 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
    }
  };
}
```

3.6.6 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 (see page 15), Structure Gadget (see page 22) See Loading Additional Web Resources For Structure Widget (see page 378) for the recommended way of extending the widget.
`structure.printable`	Printable page (see page 171)

For details about how to use web resource contexts, see Atlassian Developer Documentation.

3.7 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.

3.7.1 Download

Name	Version	Date
labels-extender-1.0.0.jar	1	2017-04-05 02:22
scheduled-sync-1.0.0.jar	1	2017-04-05 02:22
status-bar-column-2.0.0.jar	1	2017-04-05 02:22
structure-api-examples-3.0.0.zip	1	2017-04-05 02:22

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.

3.7.2 Example List

Sample Plugin	Description
simple-plugin	Very basic demo of using `StructureManager`.
scheduled-sync	A plugin that allows to schedule periodical full synchronization (resync).
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.
labels-extender	A plugin that adds Labels Extender, which includes issues in the structure based on issue key mentioned in Labels field of the parent issue.

3.8 Structure 3 API Changes

3.8.1 1. State of the API

In Structure 3 we had to change API in an incompatible way because the underlying architecture of the product had changed. If you have integration with Structure 2, most likely it won't work with the new Structure and some effort is needed to migrate the code.

As of Structure versions 3.0 – 3.1, the new API is not yet finalized and thorough documentation is not yet published. We plan to spend additional effort on making the APIs simple, stable and well-documented and publish the final documentation then.

Until that time, it's possible to use the current non-published API with Structure 3, however:

- There’s no public documentation on it. The sources of the API artifact are published, but they mostly don’t have javadocs yet.
- There will be backwards-incompatible changes while we finalize the API. The concepts and interfaces will stay mostly the same, but some classes may be moved and optimized. This is less likely to impact REST API, although we plan to introduce new REST APIs that would be simpler than the low-level API we have now.

- There will be new interfaces that would make it easier to deal with the new concepts. Right now it may be a little “low-level” and somewhat more complicated than it needs to be.

Although the documentation about the new API is not available, Structure team will be happy to assist you in migrating your code to work with Structure 3.

This article lists some of the most frequently used API calls. If you need to do something that is not covered by this article or have any questions, please write us at support@almworks.com

3.8.2 2. Conceptual Changes

2.1. Forests and Rows

In Structure 2, a structure's content was called a ***forest*** . That is still the case, however, the forest now contains ***rows*** rather than issues. A row has a Row ID – a long integer primary key for the row. Given row ID, you can retrieve information about the item displayed in that row.

The data structure that represents a forest didn't change. Previously a forest was represented by an array of pairs `(issueID, depth)`. Now the forest is represented by an array of pairs `(rowID, depth)`.

The concept of a row may seem superfluous, but it's actually required for uniquely identifying a specific position in a forest. Issue ID (or Item ID) is not sufficient because an issue can be located at multiple places in the forest.

2.2. Items

Each row has an associated ***item*** – an issue, a folder, a project or any other type of items. In Structure 3, item types are extendable and an add-on may provide additional types of items to Structure. An item is identified by ***Item Identity*** , which consists of:

- Item Type, represented by a complete module key of a JIRA plugin module that provides the item type, and
- Item ID, represented by *either* a long integer (for example, issue ID for issues) or by a string (a user key for users).

Sometimes item type is omitted; in that case the implied item type is "issue".

Some of the most popular item types are:

Item Type	Module Key	Meaning of Item ID	Comments
Issue	`com.almworks.jira.structure:type-issue`	Issue ID (long)	Default when item type is not specified
Folder	`com.almworks.jira.structure:type-folder`	Either folder ID (long) or folder i18n name (string)	Folders are introduced in Structure 3
User	`com.almworks.jira.structure:type-user`	User key (string)	
Generator	`com.almworks.jira.structure:type-generator`	Generator ID (long)	A generator is an automation rule embedded in the structure.
Page	`com.almworks.structure.pages:type-confluence-page`	Page ID (ID modulo 1e9)	Confluence pages are added as a type by Structure.Pages extension

2.3. Attributes

Attributes are a generalization of a JIRA's issue fields. An attribute is something that can be calculated for an item. For example, an issue has such attributes as "summary", "key", "priority". But purely Structure-related values are also attributes, such as "sequential number", or "aggregate progress", or "sum of story points". The attributes can also be retrieved for any types of items – for a Confluence page (provided by Structure.Pages extension), "summary" would be the title of the page, "labels" would be the labels, and a new attribute "author" would provide the initial author of the page.

Attributes are identified by an attribute specification, or ***attribute* spec**. It is usually represented as a JSON object with an ID and parameters.

2.4. Concept Comparison

	Structure 2	Structure 3
A structure's content is ...	Forest	Forest
Things that can be placed into a structure are ...	Issues	Items (including issues)
Forest consists of ...	Issues	Rows
A position in a forest is identified by ...	Issue ID	Row ID
A value in the Structure grid is displayed by ...	Column	Column
A column requests from the server ...	Fields	Attributes

3.8.3 3. REST API

3.1. Retrieving Structure Forest

```
GET /rest/structure/2.0/forest/latest?s={%22structureId%22:$id}
```

This method retrieves a content of a structure. If structure has generators, the generated content is returned. Generators are preserved in the forest.

Parameters:

- `$id` – structure ID

Return value sample:

```
{
  "spec":{"structureId":171},
  "formula":"10394:0:4/356,10332:0:14707,10374:1:5/240,10348:2:
14717",
  "itemTypes":{
    "4":"com.almworks.jira.structure:type-generator",
    "5":"com.almworks.jira.structure:type-folder"
  },
  "version":{
```

```
    "signature":-1659607419,
    "version":1
  }
}
```

In this reply, the most important part is "formula", which contains serialized information about the forest, much like in Structure 2.Each component (delimited by comma) represents a row and looks like this: `10374:1:5/240`. In this example, the numbers are:

- `10374` is the row ID,
- `1` is the row depth,
- `5/240` is the item identity. If the row contains an issue, it's just issue ID, otherwise it has the format of `<item type>/<long item id>`, or `<item type>//<string item id>`. Item type is a number, which is expanded in the "itemTypes" map in the reply.

3.2. Updating a Structure Forest

```
POST /rest/structure/2.0/forest/update
```

Parameters:

```
{
  "spec": { "structureId": <id> }, // use structure ID
  "version": { "signature": <signature>, "version": <version> },
// use last seen signature and version
  "actions": [
     {
         "action": "add",
         "under": 0,        // at the top level
         "after": 123,      // after row ID 123 (not issue iD!)
         "before": 456,   // before row ID 456
         "forest": "-100:0:10001"   // insert issue 10001, -100
is the temporary row ID which will be mapped into the real row ID
when the method returns
     },
     {
         "action": "move", // works like previously, only row IDs
instead of issue IDs
         "rowId": 123,
         "under": 456,
         "after": 0,
         "before": 124
     },
    {
```

```
        "action": "remove",
        "rowId": 442
      }
  ]
}
```

3.3. Creating a structure

```
POST /rest/plugins/structure/2.0/structure
```

Parameters:

```
{
   "name": "my structure",
   "description": "my description",
   "permissions": [   ]    // same format you see when you GET
structure
}
```

3.4. Deleting a structure

```
DELETE /rest/plugins/structure/2.0/structure/<id>
```

3.8.4 4. Java API

4.1. Versions

As Structure 3 API is finalized, it's getting a lot of refactoring and version changes. A new Structure version may have a backward-incompatible API, although incompatibilities may be isolated and your code has a good chance to work fine. However, the major version is promoted every time a backward-incompatible change is made, therefore you need to carefully set up the version of imported API packages – either set them optimistically (for example, `[12,15)` – up to version 15) and test your integration with a new release to see that there are no errors; or set the version as usual – for example, `[12,13)` – but then you might need to recompile with each new release of the API. The latter approach is recommended for in-house customizations.

Version	Supported JIRA Versions	Introduced in Structure Version	OSGi Import	OSGI Import (Optimistic)
12.0.0	JIRA 6.3+	3.0.0	"[12,13)"	"[12,15)"
12.1.0	JIRA 6.3+	3.0.1	"[12.1,13)"	"[12.1,15)"
13.0.0	JIRA 6.3+	3.1.0	"[13,14)"	"[13,16)"
13.0.1	JIRA 6.3+	3.1.1	"[13,14)"	" [13,16)"

The API versions and sources are available from the public Maven repositories – http://mvnrepository.com/artifact/com.almworks.jira.structure/structure-api

4.2. Retrieving Structure's Forest

To get the content of a structure, you need to use `ForestService` interface, which can be injected. It has `getForestSource()` method that will return a `ForestSource` given `ForestSpec`, which is a specification of what kind of forest you are retrieving. For getting just a content of a structure, use `ForestSpec.structure(structureId)`. Once you have a `ForestSource`, you can use `forestSource.getLatest().getForest()` to retrieve an instance of `Forest` – which should be familiar from the Structure 2 API.

But now `Forest` contains row IDs, not issue IDs, so to get information about what issues (or other items) are in the forest, you need to "dereference" each row ID.

4.3. Working with Rows

For working with rows, use `RowManager`. To get an item ID from a row ID, use `rowManager.getRow(rowId).getItemId()`. This gives you `ItemIdentity` instance. To see if it is an issue, use `CoreIdentities.isIssue(itemId)` and to get issue ID in that case, use `itemId.getLongId()`.

To get all row IDs for a given issue ID (for example, to find an issue in a forest), you can use `rowManager.findRows(CoreIdentities.issue(issueId))`. These row IDs may be from multiple forests, so you need to see if the forest that you have contains some of those IDs.

4.4. Getting Totals and Other Values

To calculate totals or other Structure-calculated values, you need to use `StructureAttributeService`.

`StructureAttributeService.getAttributeValues()` has the following parameters:

- `ForestSpec` – use the same forest spec that you use to retrieve the forest;
- row IDs – you need to specify for which rows (not issues!) the values are requested;
- a collection of `AttributeSpec` – specify which attributes are requested.

You need to build a list of attribute specs to specify what to calculate. There are several ways to get a correct attribute spec:

- Some specs are defined in `CoreAttributeSpecs`.
- You can build a spec using `AttributeSpecBuilder`.
- You can parse a JSON representation of a spec into a `Map`, then extract "id" and "params".

Examples:

Attribute	Spec
Story Points	`AttributeSpecBuilder .create(“customfield", ValueFormat.NUMBER) .params() .set("fieldId", 10000) // 10000 – the id of "Story Points" custom field .build()`
Story Points	`AttributeSpecBuilder .create("sum", ValueFormat.NUMBER) .params() .setAttribute(storyPoints) // storyPoints = the attribute spec for Story Points .set("distinct", true) // exclude duplicates .build()`

4.5. Changing Structure

To change a structure, you need to use `UpdatableForestSource.apply()` method. Each update is a separate transaction – the concept of a ForestTransaction used in Structure 2 has been removed.

To get an instance of `UpdatableForestSource` you need to cast `ForestSource` retrieved from `ForestService`.

Examples:

Operation	Code
Add an issue with ID 10200 to structure, under parent row with ID 1040, after row with ID 1900 and before row with ID 2000	`forestSource.apply(new UpdatableForestSource.Update.Add( CoreIdentities.issue(10200), 1040, 1900, 2000 ))`
Remove rows with IDs 10100 and 10102	`forestSource.apply(new UpdatableForestSource.Update.Remove( LongArray.create(10100, 10102)))`
Move row with ID 1010 as the first row under parent row with ID 1040, before a row with ID 1060	`forestSource.apply(new UpdatableForestSource.Update.Move( LongArray.create(1010), 1040, 0, 1060))`

4 Structure FAQ

4.1 Frequently Asked Questions

4.2 Cannot Create an Issue With +Next Issue (+Sub-Issue) Because of the Required Fields

4.2.1 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.

4.2.2 Answer

You can enter other fields when creating a new issue.

1. Use "+" button (see page 43) 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 (see page 285).

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 (see page 46) for details.

4.3 Plugin Manager Says Structure Is Unlicensed

4.3.1 Question

I have a valid license installed. Why do I see Structure as **Unlicensed** or having **Action Required** in the Plugin Manager?

4.3.2 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 (see page 299).

4.4 No Check Mark Displayed for a Resolved Issue

4.4.1 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?

4.4.2 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 (see page 65), Filtering (see page 129), Progress Column (see page 49)

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".

4.5 Structure plugin won't start

4.5.1 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?

4.5.2 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.

1. 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 (see page 324)

2. 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.

3. 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.

4. 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.

4.6 After an Issue is Moved to Another Project, It Cannot Be Found in the Structure

4.6.1 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?

4.6.2 Answer

Please check that the project where the issue was moved to is enabled for Structure (see page 305). 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 (see page 305) or move the issue to an already Structure-enabled project, e.g., to the original project.

4.7 User Cannot Access Structure, Although Permissions Have Been Granted

4.7.1 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 (see page 306) or not enabled in any project (see page 305). 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?

4.7.2 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.

4.8 Issues Not Added to a Structure when Using Links Synchronizer or Import

4.8.1 Question

I'm trying to use Links synchronizer (Import) with link type X but the issues are not added to the structure.

4.8.2 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 (see page 226) 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 (see page 126) or Filter Synchronizer (see page 231), 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 (see page 235)

4.9 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.

4.10 Integration with JIRA Agile (Greenhopper)

4.10.1 Question

We're using JIRA Agile (GreenHopper) - are there any conflicts with Structure? Can we see the Structure's hierarchy in JIRA Agile?

4.10.2 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 (see page 239) 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 (see page 239), Structure on Agile Boards (see page 31)

4.11 Using Subtasks and Structure

4.11.1 Question

Should I disable sub-tasks to use Structure?

4.11.2 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 (see page 230), which makes sure that JIRA sub-tasks are positioned under their JIRA parent issues.

4.12 Difference from Sub-tasks

4.12.1 Question

How is issue hierarchy provided by Structure plug-in different from the standard sub-tasks?

4.12.2 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 (see page 33)

4.13 Some Link Synchronizer Operations Are Not Written to the History

4.13.1 Question

Link creation and removal operations, when performed by link synchronizers, are not written to the issue history and activity streams. Why?

4.13.2 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

4.14 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 (see page 219). 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 (see page 325) to reduce the load (for example, the Structure panel on the Issue Page) and limit the group of users (see page 333) Structure is exposed to.

4.15 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 (see page 166) 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.

4.16 Can I export a structure to Microsoft Word so that it can be emailed as a document?

Exporting a structure to MS Word directly is not supported at this time. The nature of Structure plugin is such, that the format and the presentation of data is much closer related to MS Excel than that of MS Word. To export to Excel just click a drop-down arrow on the **Export** button located on the right of the *Structure Toolbar*, and choose **Export to Excel**. Structure will create an MS Excel file with the same name as the structure that you are exporting and will save it in your browser's *Download* folder.

Once you have the file, you can open it, copy the data you need and paste it into any MS Word file for further formatting.

Another way to convert a structure into a document that can be shared with a customer, is to use **Export** | **Printable Page** and then use any "PDF Printer" to save it as a PDF File.

5 Structure Troubleshooting

5.1 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* 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. After you've collected the support zip, you can go back to **Administration | System | Logging and Profiling page** and set the logging level for *com.almworks* to WARN - it's the default level.

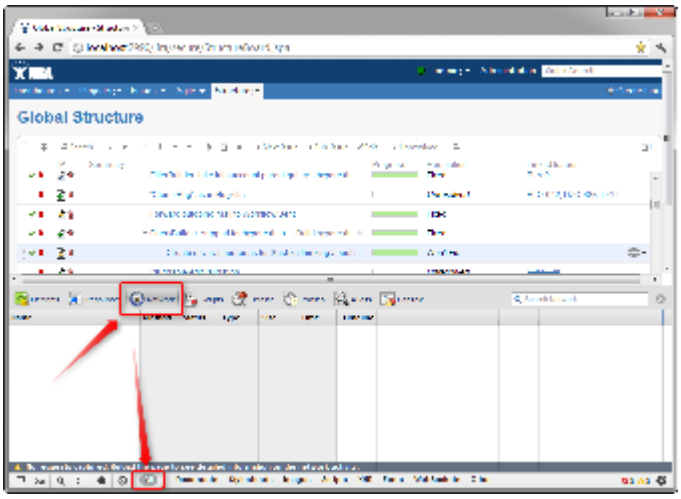
5.2 HAR Network Report

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.)

5.2.1 Collecting HAR Report with Google Chrome

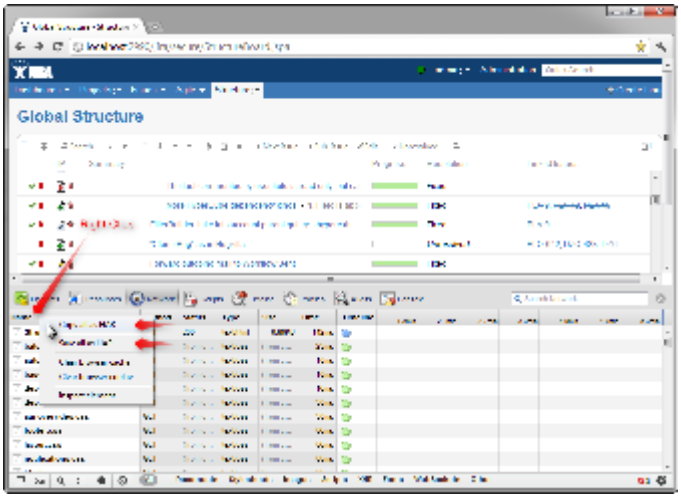
1. Open a new Chrome window and navigate to the page where the problem happens.
2. Press **Ctrl+Shift+I** or use menu **Wrench | Tools | Developer Tools** to open a section with developer tools. Switch to the **Network** tab there. Make sure **All** tab is selected below.



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.

5.3 Troubleshooting Synchronizers

Structure synchronizers (see page 219) 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.

5.3.1 Structure Audit Log

Starting with Structure 3, all standard synchronizers record all actions they have taken in the database and allow the administrator to undo the changes. Navigate to **Administration | Structure | Support | Synchronizer Audit Log** to query history or apply undo.

5.3.2 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 (see page 461) (until the next JIRA restart) or permanently (see page 462).

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
```

5.4 Structured JQL Troubleshooting

If a Structured JQL (see page 252) 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.

5.5 Collecting Performance Snapshots

Performance snapshots allow ALM Works support team to analyze performance-related problems on your JIRA server without direct access to it.

5.5.1 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	2017-04-05 02:22

5.5.2 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.

5.5.3 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.

5.5.4 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.

5.5.5 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 (see page 470).

5.5.6 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.)

5.5.7 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.

1. 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 (see page 466).

1.1. 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`.

2. 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,onlylocal,dir=%
~dp0..\profiler\snapshots,delay=20000 -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 (see page 469).

3. 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.
```

4. 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.

4.1. 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.

4.2. 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
```

5. 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.

6. 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.

5.6 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).

5.6.1 1. Attach to the Support Request in ALM Works Service Desk (Preferred)

File size limit: 20 MB

If the files pertain to a Support Request on https://support.almworks.com, please use Service Desk to upload and attach the files to the ticket. Size limit is 20 MB per upload.

5.6.2 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.

5.6.3 3. Upload Files Directly to Our Server

File size limit: 5 GB

If you need to send us files larger than 20 MB, please let us know. We will send you a custom link that will allow you to upload such files directly to our secure server.

> The files you have uploaded are safe – they cannot be accessed by anyone except ALM Works support.

5.7 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.

5.7.1 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	Uni
6.0–6.0.8	`gadget:Structure (IE 8-9 Compatible, Works with JIRA 6.0.x)`	str
6.1–6.1.7	`gadget:Structure (IE 8-9 Compatible, Works with JIRA 6.1.x)`	str

   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)".

5.8 Troubleshooting Performance and Stability Issues

In cases when JIRA's performance deteriorates or if the system becomes unstable or unresponsive, it is important to achieve two goals:

1. Bring system back to normal in the shortest amount of time.
2. Collect information that would help analyze the problem and make sure it does not appear again.

The second goal is strategically very important, however, it might get overlooked in a rush to make things work "now". For example, JIRA administrator may be inclined to restart a stuck JIRA instance quickly in order for it to get back to working state as fast as possible. But if thread dumps are not collected, the developers will never know where JIRA was stuck, so the same problem may happen again.

The first goal is of course also very important. Sometimes JIRA administrator manages to restore system functioning, sometimes help from Atlassian and ALM Works support teams is needed. Support engineers and developers would typically take into account all information they have, analyze it and try to pinpoint the source of the problem. Often additional information is required from the JIRA administrator, and sending requests and replies back and forth takes precious time.

This article is intended to provide JIRA administrators with advice about how to collect maximum information about a performance or stability problem, when that problem happens. The list is not intended to be complete, additional information may still be needed, however, providing all listed information gives a good chance that a support engineer will be able to identify a problem and provide advice sooner.

5.8.1 1. Thread Dumps

Thread dumps are the most important information when system is unresponsive or has performance issues. They allow to peek into what's going on inside JIRA's JVM process.

- Please refer to Atlassian documentation on generating a thread dump for instructions of manually capturing a thread dump on the server.
- Thread dumps are also a part of the Support Zip (3 dumps are generated in one zip), however, generating a support zip might be unavailable if JIRA is hanging.
- For best diagnosis, please collect 5-6 thread dumps with 3-4 second interval.

Please collect 5-6 thread dumps with 3-4 second interval.

5.8.2 2. Verbose Logging

If the problem has temporary but reproducible manner, you can turn on verbose logging so that the engineers can gather more information from the logs. To do so:

1. Open **Administration | System | Logging and Profiling** page.
2. Enter STRUCTURE TROUBLESHOOTING into the Optional Message field, turn on Log Rollover and press Mark.
3. Scroll down and click **Configure logging level for another package**, enter package name **com.almworks** then select logging level **DEBUG** and click **Add**.

Then you can try to reproduce the problem and collect the support zip.

Do not forget to turn off the DEBUG logging after the problem has been resolved, otherwise you may get too many messages in the logs during normal operation.

5.8.3 3. Support Zip

Support zip is the most important thing after thread dumps. It allows engineers to have full understanding of the environment and retrospect using the logs into what was going on. If you had Verbose Logging on before problem appeared, it gives even more details.

To collect a support zip:

1. Open **Administration | System | Atlassian Support Tools**, switch to **Support Zip** tab.
2. Open **Administration | System | Support Tools**, switch to **Create Support Zip** tab. Select all options. Click **Create**.
3. Download the resulting ZIP file and send it to the support teams: either attach it to the ticket, or, if the file is large, request a URL for uploading.

On JIRA Data Center, collect Support Zips on each node.

5.8.4 4. Browser Console Log

If the problem seems to be on the client side, in the browser – if there are errors or if the browser is hanging or some button or link does not respond, check out the browser's error console. Depending on the browser type, the console may be opened with different menus or keyboard shortcuts.

1. Reproduce the problem
2. Copy all contents from the console and send it to support.

Also, please include browser type and version, as well as the information about operating system.

5.8.5 5. HAR Report

HAR report is also taken on the browser and contains logs of network communications with the server. Use this log to provide information that can help troubleshoot issues with slow loading of data or general slow responsiveness on the client side.

1. Use Google Chrome
2. Open menu, More Tools | Developer Tools.
3. Switch to Network tab

4. Reproduce the problem
5. Right click in the table and select "Save as HAR with content..." or "Copy All as HAR".
6. Paste or save HAR as a file.

HAR with content provides more information but it may contain your JIRA's data. Review the contents before sending it out to support.

5.8.6 6. Screenshots or Video

When there's a visible and informative behavior demonstrated by the system, a screenshot or a video showing the problem would go a long way in getting the support engineers understand the issue.

You can use operating system's native tools to capture a video, or install a third-party tool for that. Feel free to ask ALM Works Support for recommendations if you don't have preferable screen capturing tool.