Mastering JIRA

JIRA is an issue-tracking tool from Atlassian and has gained immense popularity in recent years due to its ease of use and, at the same time, its customization abilities and finely grained control over various functions. JIRA offers functionalities for creating tasks and assigning them to users and many useful add-ons can be added such as JIRA Agile for Agile tracking and Groovy scripts, a powerful tool for administering customizations for customizations.

This book explains how to master the key functionalities of JIRA and its customizations and add-ons, and is packed with real-world examples and use cases. You will first learn how to plan JIRA installation. Next, you will be given a brief refresher of fundamental concepts and learn about customizations in detail. Next, this book will take you through add-on development to extend JIRA functionality. Finally, this book will explore best practices and troubleshooting, to help you find out what went wrong and how to fix it.

Who this book is written for
If you are a JIRA administrator managing small-to-medium JIRA instances and want to learn how to manage enterprise-scale instances, then this book will help you expand your knowledge and equip you with advanced skills. Prior understanding of JIRA core concepts is required.

What you will learn from this book
- Plan the installation and upgrade of your JIRA instance
- Configure issue type schemes, create customized workflows, modify permission schemes, and change notification schemes
- Customize JIRA for test management, helpdesk, and requirement management
- Deploy JIRA for Agile tracking using the Scrum and Kanban techniques
- Manage users and groups and integrate them with LDAP for single sign-on
- Develop JIRA add-ons to extend JIRA's functionality
- Migrate data from external systems using CSV import
- Generate reports directly from the JIRA database and to use custom JavaScript and CSS

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 11 'JIRA Administration with Groovy Script Runner'
- A synopsis of the book’s content
- More information on Mastering JIRA
Ravi Sagar is a JIRA trainer, consultant, and Drupal expert with several years of experience in web development and business analysis. He has done extensive work implementing and customizing big JIRA instances for project tracking, test management, support tickets, and Agile tracking.

Ravi founded Sparxsys Solutions Pvt. Ltd. (www.sparxsys.com) in 2010, a start-up company that provides consultancy and training services on Atlassian tools and Drupal. He has created accessible websites for blind people, adhering to WCAG guidelines. Ravi's areas of interest include project management and Agile methodologies.

His areas of focus in customizing JIRA include topics, such as issue schemes, workflow schemes, field configuration schemes, screen schemes, permission schemes, and notification schemes. He has also worked on Agile tracking projects, such as Scrum and Kanban. He contributed immensely towards setting up JIRA for helpdesk, test case management, bug tracking, and support ticket management. His other areas of expertise include JIRA training, Drupal training, business analysis, project management, and JIRA Agile.

Ravi has extensive experience in JIRA installation and configuration and has also worked on Linux and Windows Server. He understands clients' requirements and suggests best solutions to save cost.

Ravi has been involved in JIRA support and maintenance and training, including regular upgrades of JIRA and installed plugins, migration from legacy-defect tracking tools to JIRA, splitting and merging JIRA instances apart from bulk actions, such as uploading issues, editing, and user creation.

He has also worked on implementing JIRA Agile and its integration with other tools, such as Confluence, Crucible, and Fisheye and has hands-on experience in JIRA REST and SOAP.

You can connect with him at http://www.linkedin.com/in/ravisagar or e-mail him at ravi@sparxsys.com.
JIRA is an issue-tracking tool from Atlassian, which has gained immense popularity in recent years due to its ease of use, its customization abilities, and finely grained control over various functions. Out of the box, JIRA offers issue and bug tracking capabilities to create tasks, assign it to users, and generate useful reports. However, the real power of JIRA lies in the customization that it offers.

Experienced JIRA administrators looking to learn advanced topics and expand their knowledge will benefit from this book.

Packed with real-world examples and use cases, you will first learn how to plan the JIRA installation. Then, you will be given a brief refresher of the fundamental concepts. You will also understand the customizations in detail, along with a sample data for various use cases. Several aspects of JIRA administration, such as user management, groups, roles, and security levels, will be covered keeping in mind the applications for enterprises. Next, this book will take you through the add-on development to extend JIRA functionalities. It will also give you insights on how to build applications on top of JIRA using the REST API. Various aspects of the migration process from other tools using the CSV file will also be discussed.

The implementation of Scrum and Kanban techniques, along with Agile reports, will be discussed. We will take a look at the Groovy script, which is a great tool that empowers JIRA administrators with tremendous flexibility. We will also take a look at some of the common database tables to fetch useful results and discuss the possibilities to add custom CSS and JavaScript in our JIRA instance. Finally, we will conclude the book by going through the best practices and troubleshooting steps to help you find out what went wrong and how to fix it.
What this book covers

Chapter 1, Planning Your JIRA Installation, covers planning of the JIRA installation to ensure longevity of the installation so that it can accommodate more users and data in the future; the installation and update process is also discussed briefly in this chapter.

Chapter 2, Searching in JIRA, has detailed explanation on how data can be fetched from JIRA using the Basic search feature, as well as by writing advanced queries using JQL.

Chapter 3, Reporting – Charts to Visualize the Data, covers various built-in project reports that come with JIRA. It also covers how to present them in the Dashboards.

Chapter 4, Customizing JIRA for Test Management, explains how to modify the configurations to implement new Issue Types for Test Campaign and Test Case. The procedure to implement a new workflow with conditions, along with new permission schemes, will be discussed in detail in this chapter.

Chapter 5, Sample Implementation of Use Cases, has a lot of examples of different implementations, such as a helpdesk system and requirement management, which readers can leverage in their company.

Chapter 6, User Management, Groups, and Project Roles, explains how to manage users in JIRA and the way to organize them in various groups.

Chapter 7, Configuring JIRA User Directories to Connect with LDAP, Crowd, and JIRA User Server, discusses how to integrate your JIRA instance with LDAP and Crowd for external user management.

Chapter 8, JIRA Add-on Development and Leveraging REST API, explains how to start developing add-ons for JIRA to extend its functionalities. The JIRA REST API that enables accessing JIRA’s functionalities from external tools is also discussed with examples.

Chapter 9, Importing and Exporting Data in JIRA, talks about how data from external tools can be imported using the CSV import and Project Import feature. The importance of taking regular backups is explained in this chapter, along with the procedure to restore JIRA from the backup file.

Chapter 10, Working with JIRA Agile, explains how to implement the Scrum and Kanban technique in JIRA. The planning of your Sprints in the Scrum and various customizations that one can perform in these boards is discussed in detail, along with Burndown and Velocity charts to track the progress of the project.
Chapter 11, *JIRA Administration with Groovy Script Runner*, introduces the add-on that administrators can install and various additional features using scripting that it brings, which helps JIRA administrators with various customizations that were otherwise not possible.

Chapter 12, *Accessing the Database*, explains fetching the data directly from JIRA's database. This chapter has various useful queries to retrieve information from the database. The way to access data from embedded HSQL database has also been explained.

Chapter 13, *Customizing Look and Feel and Behavior*, talks about how to perform extreme changes in the JIRA design using custom style sheets. This chapter also discusses the possibility to control the HTML fields using JavaScript.

Chapter 14, *JIRA Best Practices*, discusses various points that JIRA administrators should keep in mind not only before implementing JIRA, but also various practices that they should employ on an ongoing basis.

Chapter 15, *Troubleshooting JIRA*, is the last chapter where various ways to identify the problems in the instance is discussed. Common problems that people face in JIRA are listed in this chapter.

Appendix, *Integrating JIRA with Other Tools*, has details on how various tools, such as Git, Bitbucket, and Confluence can be integrated with JIRA.
JIRA Administration with Groovy Script Runner

When JIRA is used to implement a complex use case with lots of conditions, validations, and workflows, there are certain configurations that are not doable using the existing set of features. For instance, you want to autocalculate a custom field value after a workflow transition. Using the Script Runner plugin, many such advanced configurations can be achieved. This chapter will discuss this useful plugin in detail along with examples.

Installing Script Runner

Just like any other add-on, Script Runner can be installed from the JIRA Administration interface. Perform these steps to install Script Runner on your JIRA instance:

1. Navigate to JIRA Administration | Add-ons | Find new add-ons (under ATLASSIAN MARKETPLACE).
2. In the search box, enter *Script Runner* and click on the *Enter* key. The Script Runner add-on will appear in the search result list:

![Atlassian Marketplace for JIRA](image)

3. Click on the **Install** button and the Script Runner add-on will begin downloading:

![Download and installing](image)
4. Script Runner will then be downloaded and installed in your instance.
5. In the popup window that appears confirming the add-on installation, click on the Close link at the bottom-right corner:

<table>
<thead>
<tr>
<th>Installed and ready to go!</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Script Runner v 3.0.9" /></td>
</tr>
<tr>
<td>by Jamie Echlin Ltd</td>
</tr>
</tbody>
</table>

This add-on has been installed. If you need help getting started, click the link to the add-on documentation from the Manage add-ons screen.

---

Script Runner will now be installed in your JIRA instance.

**Built-in scripts for administration**

The Script Runner add-on allows users to write and run their own scripts. It comes with plenty of nice scripts that allow JIRA administrators to perform various activities that are otherwise difficult to perform using existing JIRA features, or not possible at all from the UI. Let's take a look at some of these scripts.

**Accessing Built-in Scripts**

To access scripts that come with the Script Runner add-on, perform these steps:

1. Navigate to JIRA Administration | Add-ons | Built-in Scripts (under SCRIPT RUNNER).
2. In this section, the list of all the **Built-in Scripts** can be found:

<table>
<thead>
<tr>
<th>Built-in Scripts</th>
<th>Removed scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select them one of the built-in scripts. If it takes parameters order them in the form that will be shown.</td>
<td>Some scripts like Condition Tester and Send Email preview have been removed, they can now be previewed as a workflow function. Create a draft workflow to preview them.</td>
</tr>
<tr>
<td>Clicking preview will give more information what this script will do.</td>
<td></td>
</tr>
<tr>
<td>Note: Different scripts are available for workflow functions and listeners.</td>
<td></td>
</tr>
</tbody>
</table>

- Bulk Field Resolutions
  - modify/resolution field in bulk, without creating a change entry. Useful after a bad import.
- Bulk import custom field values
  - bulk import custom field values
- Change dashboard or filter ownership
  - This will change the ownership of the selected dashboards/filters, and for dashboards anything that they use
- Clear the groovy internal cache
  - clear the groovy_rerun cache if it is not working automatically, or the jira cache if you have changed something in the database

- Expect a delay after executing this.
- Copy custom field values
  - Copy custom field values from one field to another in bulk, to support changing a field type without using SQL or restarting.
- Copy project
  - This tool will create a new project, from the configuration of another project.

This includes: Schemas, Role memberships, Custom field configurations, and optionally, issues, versions and components.

- Recalculate schema
  - Recalculate多么多 issues based on a JQL query, for instance to change the state of issues if they have been inactive for 2 weeks. BETA.
- Generico events
  - Generico issue: Created events to be consumed by listeners.
- Resolves
  - Resolves issues
- Resolves issues for a project or corresponding to a filter, perhaps after an indexing problem or editing the database.

- Script registry
  - View all your groovy scripts in workflows, listeners, db.

- Split custom fields contexts
  - This script splits projects out a custom field context, duplicating option values and updating issues with the new option values.
  - Switch to another user to deal with support problems and so on.
- Unit test runner
  - Unit test runner

3. Click on any of these links to run that particular built-in script. This will further ask you to enter parameters relevant for that script.

Let's take a look at some of these Built-in Scripts.

### Copying a project

Only JIRA Administrators have the permission to create a project in JIRA and change the configurations too. Whenever there is a need to have an additional project in JIRA, the administrator needs to manually create projects and then change their schemes. Although, it's not a difficult task and usually takes 10 minutes of time, sometimes there is a need to create ten projects and creating them manually could take hours.

There is a built-in script in Script Runner to copy the project along with its configurations, with or without its issues. Perform these steps:
1. Click on the Copy project link:

- **Copy project**
  - This tool will create a new project, from the configuration of another project.
  - This includes Schemes, Role memberships, Custom field configurations, and optionally, issues, versions and components.
  - **Source project**: Helpdesk
  - **Target project key**: SST
  - **Target project name**: System Support Ticket
  - **Copy versions**: ✓
  - **Copy components**: ✓
  - **Copy issues**: □
  - **Copy Classic GreenHopper configuration**: □
  - **Copy project specific dashboard and filters**: □

2. On the next screen, select Helpdesk as Source project, enter Target project key and Target project name.

3. Tick the Copy versions and Copy components checkbox if you want to copy the project versions and project components as well.

4. Click on the Run button to initiate the Copy project script.

Once done, the message is displayed at the bottom of the screen stating that the project has been copied with a link to the new project. If you check the new copied project, you will notice that it has all the configurations of the source project. This tool takes less than a minute to run and JIRA administrators can save a lot of time using it.
Escalation service

This is an excellent Built-in Script that helps JIRA administrators to perform periodic actions on a certain set of issues. Let's take an example of a support ticket configuration in which we have a workflow state called Waiting for Client in our project. This is used to signify that further information is required from a client to act further on a ticket. These issues need to be resolved automatically when the ticket is not updated in the past 2 weeks:

1. Click on Escalation service:

2. Click on New Service to create a new service:
3. Enter No updates in 2 weeks as **Description**.

4. In **JQL Query**, enter `status= "Waiting for Client" AND updated >= -14d`.

5. Enter your **User ID** and specify **Interval/CRON Expression**.

   ![Image](image.png)

   As we want to resolve the issues that qualify the criteria in JQL Query, select **Additional issue actions** as **Set resolution to Fixed**; this will automatically insert the `issueInputParameters.setResolutionId('1')` code.

6. Click on the **Add Service** button to save this service.

    ![ './escalation.service](image.png)

Now, we have a service added that will run after every 24-hour period and will resolve the issues that were not updated in the past 14 days, that is, 2 weeks.

**Switching to a different user**

Imagine a situation when a user reports a problem in JIRA. As a JIRA administrator, you need to log in with his/her ID to understand the problem that this user might be facing. You can either ask this user his/her password or create a similar user with the same set of permissions. Instead, wouldn't it be better to be able to log in to JIRA using that user's username without asking for the password? There is a Built-in Script to perform just that:
1. Click on **Switch to a different user**.
2. Enter the **User ID** of the user and click on the **Run** button:

   ![Switch to a different user](image)

3. Click on the **here** link that appears at the bottom of the screen. You will then be logged in with a different user:

   ![Click here to continue](image)

This script is one of my favorites because I can log in with any username I want without asking their password and it also saves a lot of time.

### Modifying the JIRA workflow with conditions, validators, and post functions

The best part of the Script Runner add-on is the additional features it brings in the JIRA workflow. Out of the box, there are various conditions, validators, and post functions that can be configured in the workflow, but it offers limited functionalities. Script Runner simply gives you more options that you can control in the workflow. Let's take a look at them.

### Conditions

Script Runner brings a set of additional conditions that you can add in the workflow; it gives you an amazing control over a lot of things that was not possible earlier. Perform these steps:

1. Modify the workflow of your choice. For any transition, navigate to **Add Condition To Transition**.
2. You will find a new condition called **Script Condition**; select it and click on the **Add** button:
3. On the next screen, you will get a list of scripts that you can add as a workflow condition:

Let's discuss these scripts.
JIRA Administration with Groovy Script Runner

All subtasks must be resolved
If you want all the subtasks of a particular issue to be resolved with a specific resolution, then this condition can be added:

Simply select the Resolution that you want your subtasks to match and click on the Add button.

Allowing the transition if this query matches a JQL query
While performing a workflow transition, you can use a custom JQL in the workflow condition and allow the transition only when that JQL returns the issue that you will transition:
For instance, if you want the transition to happen only when the assignee of the issue is the currently logged in user and the due date is today, then add the `assignee = currentUser()` and `due = now()` JQL Query and click on the Add button. Optionally, you can also enter the specific issue id to preview this condition.

**Checking if the issue has previously been in a status**

The workflow can have numerous states and transitions between them. A workflow state can have more than one transition. For some reason, if you want the transition to be from a particular state only, then this condition can be added:

![Select script](image)

Just select **In Progress** from the drop-down list for **Previous Status** and check whether this status is **Immediately previous only**, uncheck this option if it is any other status and click on the Add button.
Simple scripted condition
Script Runner has some ready-made simple scripts that can be added quickly as a condition:

```
 tercer.getAttachments(Issue).any(lt.filename.endsWith(".pdf"))
```

Just click on any of the example links and it can be added as a condition. For instance, I added a condition that will check whether one of the issue attachments is a PDF file or not. Click on the Add button to continue.

Validators
Just like additional conditions, Script Runner brings a set of additional validators that you can add in the workflow; it gives you an amazing control over lots of things that were not possible earlier. Perform these steps:
1. Modify the workflow of your choice and for any transition, navigate to **Add Validator To Transition**.

2. You will find a new validator called **Script Validator**. Just select it and click on the **Add** button:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Compare</td>
<td>Compares two date fields during a workflow transition.</td>
</tr>
<tr>
<td>Date Expression Compare</td>
<td>Compares a date field to a date expression during a workflow transition.</td>
</tr>
<tr>
<td>Date Window</td>
<td>Compares two date fields, by adding a time span in days to one of them.</td>
</tr>
<tr>
<td>Fields Required</td>
<td>Fields required during a workflow transition.</td>
</tr>
<tr>
<td>Permission Validator</td>
<td>Validates that the user has a permission.</td>
</tr>
<tr>
<td>Regular Expression Check</td>
<td>Validates field contents against a regular expression during a workflow transition.</td>
</tr>
<tr>
<td>Script Validator</td>
<td>Runs a script to check validation, or a button script.</td>
</tr>
<tr>
<td>User Permission Validator</td>
<td>Validates that the user has a permission, where the OOBWorkflow variable holding the username is configurable. Obsolete.</td>
</tr>
</tbody>
</table>

3. On the next screen, you will get a list of scripts that you can add as a workflow validator:

<table>
<thead>
<tr>
<th>Select script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click on a script and add parameters. For running your own code select Custom script post-function.</td>
</tr>
<tr>
<td>Run your own groovy script from a file or entered into JIRA.</td>
</tr>
<tr>
<td>Field(s) changed validator</td>
</tr>
<tr>
<td>Enforces that one or more fields are changed as part of the transition.</td>
</tr>
<tr>
<td>Require a comment on transition</td>
</tr>
<tr>
<td>This validator enforces a comment is provided on transition.</td>
</tr>
<tr>
<td>Simple scripted validator</td>
</tr>
<tr>
<td>Runs a simple embedded script to find out whether to allow the transition or not</td>
</tr>
</tbody>
</table>

Let's discuss these scripts.
Field(s) changed validator

It's possible that you will use a transition view in the workflow transition that pops up a window to the user to capture additional input. These transition views are nothing but a screen containing one or more fields. Use this validator to validate whether these fields present in the transition view have changed:

Select all the Fields that you want to check for change and click on the Add button. The fields visible here are only the ones that are part of the transition view for the transition you are working on in the workflow.

Require a comment on transition

In the transition view, there is usually a comment field as well. Use this validator to validate whether a comment has been added or not:

This validator doesn't require any parameters to configure. Just click on the Add button.
Simple scripted validator

Script Runner has some ready-made simple scripts that can be added quickly as a validator:

Just click on any of the example links and it can be added as a validator. For instance, if you want to enforce that the issue should have at least four subtasks, then click on Has greater than two subtasks and modify Condition from 2 to 4, so finally, it's issue.subTasks.size() > 4. Click on the Add button to continue.
Post Functions

Just like additional conditions and validators, Script Runner brings a set of additional post functions that you can add in the workflow; it gives you an amazing control over a lot of things that was not possible earlier. Perform these steps:

1. Modify the workflow of your choice and for any transition, navigate to the **Add Post Function To Transition**.
2. You will find a new post function called **Script Post-Function**; select it and click on the **Add** button:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign to Current User</td>
<td>Assigns the issue to the current user if the current user has the 'Assignable User' permission.</td>
</tr>
<tr>
<td>Assign to Lead Developer</td>
<td>Assigns the issue to the project component lead developer</td>
</tr>
<tr>
<td>Assign to Reporter</td>
<td>Assigns the issue to the reporter</td>
</tr>
<tr>
<td>Clear Field Value</td>
<td>Clear value of a given field.</td>
</tr>
<tr>
<td>Copy Value From Other Field</td>
<td>Copies the value of one field to another, either within the same issue or from parent to sub-task.</td>
</tr>
<tr>
<td>Create Perform Job Function</td>
<td>Creates a Perform Job (if required) after completing the workflow transition.</td>
</tr>
<tr>
<td>Notify HipChat</td>
<td>Send a notification to one or more HipChat rooms.</td>
</tr>
<tr>
<td>Script Post-Function</td>
<td>Runs a script in a post-function, or a built-in script.</td>
</tr>
<tr>
<td>Trigger a Webhook</td>
<td>If this post-function is executed, JIRA will post the issue content in JSON format to the URL specified.</td>
</tr>
<tr>
<td>Update Issue Custom Field</td>
<td>Updates an issue custom field to a given value.</td>
</tr>
<tr>
<td>Update Issue Field</td>
<td>Updates a simple issue field to a given value.</td>
</tr>
</tbody>
</table>
3. On the next screen, you will get a list of scripts that you can add as a workflow post function:

- Adds a comment to all blocked issues when this issue is transitioned.
- Useful for alerting participants of other issues that a blocker is resolved, etc.
- This function should be put on the Resolve transition (or similar).
- Adds the current user as a watcher
- Adds the user performing the action as a watcher, if condition applies
- Assign to first member of role
- Assign to first member of the specified role
- Assign to last role member
- Assign this issue to the last user from the specified role who this issue was assigned to previously.
- Clones an issue and links
- Clones this issue to another issue, optionally in another project, and optionally a different issue type.
- Create a sub-task
- Create a sub-task. Will optionally reopen a matching sub-task.
- Custom script post-function
- Run your own groovy script from a file or entered into JIRA.
- If the condition is met, automatically transition this issue to another status
- If an event occurs when condition is true
- File an event that can be picked up by a notification scheme, in order to send mail only under certain conditions, eg Priority is Blocker
- Send a custom email
- Send an email based on the provided template if conditions are met
- Set issue security level depending on provided condition
- Set issue security if the provided condition evaluates to true
- Transition parent when all sub-tasks are resolved
- This will do the given action on the parent when all sub-tasks are resolved

Let's discuss some of these scripts.
Adds the current user as a watcher

Consider a scenario when the priority of the issue is Major and the user who is currently logged in and making the workflow transition should be added as a watcher of the issue; in this case, use this post function:

- Adds the current user as a watcher

Add the user performing the action as a watcher, if condition applies

<table>
<thead>
<tr>
<th>Condition</th>
<th>Issue {priority} name == 'Major'</th>
</tr>
</thead>
</table>

Enter the condition for which this function will fire. Example will evaluate to “true”. You can click one of the examples:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority is Major</td>
<td></td>
</tr>
<tr>
<td>Issue type is Defect</td>
<td></td>
</tr>
<tr>
<td>Reporter is current user</td>
<td></td>
</tr>
<tr>
<td>Has greater than two subtasks</td>
<td></td>
</tr>
<tr>
<td>Has string custom field value equal to</td>
<td></td>
</tr>
<tr>
<td>Has select list value equal to</td>
<td></td>
</tr>
<tr>
<td>Multi select list contains value</td>
<td></td>
</tr>
<tr>
<td>Cascading select list first value equal to</td>
<td></td>
</tr>
<tr>
<td>Cascading select list both values equal to</td>
<td></td>
</tr>
<tr>
<td>Has at least one outward &lt;i&gt;duplicate&lt;/i&gt; link</td>
<td></td>
</tr>
<tr>
<td>Has one or more attachments</td>
<td></td>
</tr>
<tr>
<td>Current user is member of Administrators role</td>
<td></td>
</tr>
<tr>
<td>Reporter is in a particular group</td>
<td></td>
</tr>
<tr>
<td>Has at least one PDF attachment</td>
<td></td>
</tr>
<tr>
<td>Assignee changed (Workflow functions only)</td>
<td></td>
</tr>
<tr>
<td>Priority changed to Major (Listeners only)</td>
<td></td>
</tr>
<tr>
<td>Priority changed to Major (Workflow functions only)</td>
<td></td>
</tr>
</tbody>
</table>

From the list of examples, click on the Priority is Major link and a Condition will be added. Now, whenever this condition is true during the workflow transition, the post function will be executed. Click on the Add button to add the post function.

Transitioning the parent when all subtasks are resolved

If your issue has a lot of subtasks, then it's possible to move the parent to a new state in the workflow when all its subtasks are resolved:
Select Parent action as Done (41) and Resolution as Fixed. Click on the Add button to add the post function.

There are numerous other post functions that can be used in the workflow. Using the Script Runner add-on, a lot of flexibility and control can be added in the workflow to perform actions that were not possible earlier.

**Accessing powerful JQL functions**

We have already discussed searching issues in JIRA with the basic and advance search using JQL. However, there are some limitations of JQL. JIRA administrators often try to fetch the information directly from the database, which is difficult to do because it requires a good knowledge of the JIRA database schema.

Script Runner introduces new JQL functions. You can use these functions in your instance. After installing this add-on, just perform the re-indexing to enable the new JQL functions.

Let's discuss some of these JQL functions.

**Returning issues with a number of comments**

Use the following JQL queries to return issues with the exact number of comments:

- `issueFunction in hasComments(3)`

  The following query will return an issue with more than four comments:

- `issueFunction in hasComments('+5')`
Returning issues based on comments attributes

Use the following query to return issues commented on by project role administrators:

```
issueFunction in commented("role Administrators")
```

This query returns issues with comments from a specific user in the past 7 days:

```
issueFunction in commented("after -7d by ravisagar")
```

Returning issues based on attachments

Use the following query to fetch issues with PDF as an attachment:

```
issueFunction in hasAttachments ("pdf")
```

This query finds issues in the file that was attached by a specific user in the past 7 days:

```
issueFunction in fileAttached("after -7d by ravisagar")
```

Comparing dates

Issues can also be fetched by comparing their date fields, such as resolution date and due date.

Use this query to return issues that were resolved later than their due date:

```
issueFunction in dateCompare("", "resolutionDate > dueDate")
```

The following query finds issues that were resolved within 1 week of their creation:

```
issueFunction in dateCompare("", "created +1w > resolutionDate ")
```

These are just some of the examples of additional JQL functions that you can use. For the full list, I recommend you to refer to https://jamieechlin.atlassian.net/wiki/display/GRV/Scripted+JQL+Functions.

Script Runner is personally my favorite add-on that I use with all of the JIRA instances that I manage. It just gives so much power and control to effectively manage various administrative tasks in JIRA. Apart from various built-in scripts, which give administrators access to ready-to-use features, one can also write his/her own script and use it to perform more advanced and complex tasks. This ability to write scripts opens up a lot of possibilities to enhance the workflow and to add more features in JIRA without developing an add-on.
In this chapter, we discussed Groovy Script Runner, which is an amazing add-on to perform complex customizations in the workflow, access powerful JQL functions, and run various scripts that can be used by JIRA administrators to maintain the instance efficiently. Script Runner is by far the most popular admin tool used by JIRA administrators.

In the next chapter, we will discuss how to access the JIRA database directly to fetch data. JIRA offers lots of good reports, but sometimes they are not good enough and more insight is required. If you know the JIRA database schema and how to access the database, then any data can be retrieved for further reporting purposes.
Where to buy this book

You can buy Mastering JIRA from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.