

Jira Integration

Introduction

This topic describes integration between Admin By Request and Jira for cloud-based Jira implementations. The integration is relatively simple, comprising three mechanisms: one to create issues in Jira from requests made in ABR, one to handle approved requests and one to handle declined requests. This is the extent of our Jira integration; it is up to the customer to further automate the updating of tickets through Jira automation rules.

Nevertheless, we include in this topic two simple automation rules to illustrate how some things might be done. Customers can use these as the basis for their own automation workflows.

IMPORTANT:

- The integration covered here is available for Jira Cloud, but not on-premise installations like Jira Data Center Edition.
- Disclaimer - Admin By Request is not a Jira consultancy and we do not have extensive experience working with Atlassian tools, including Jira. Our integrations and examples are provided for the customer's convenience; they do not carry any warranty whatsoever and must be used entirely at the customer's own risk. We recommend testing everything in a non-production environment and we are not liable for any downtime or loss of productivity or data that might result from using the integrations.

In this article

["Prerequisites" on the next page](#)

["1. Creating custom fields" on page 5](#)

["2. Creating a custom Request Type" on page 8](#)

["3. Integrating Jira with Admin By Request" on page 12](#)

["Updating Jira issues" on page 15](#)

["Creating Jira automation rules" on page 16](#)

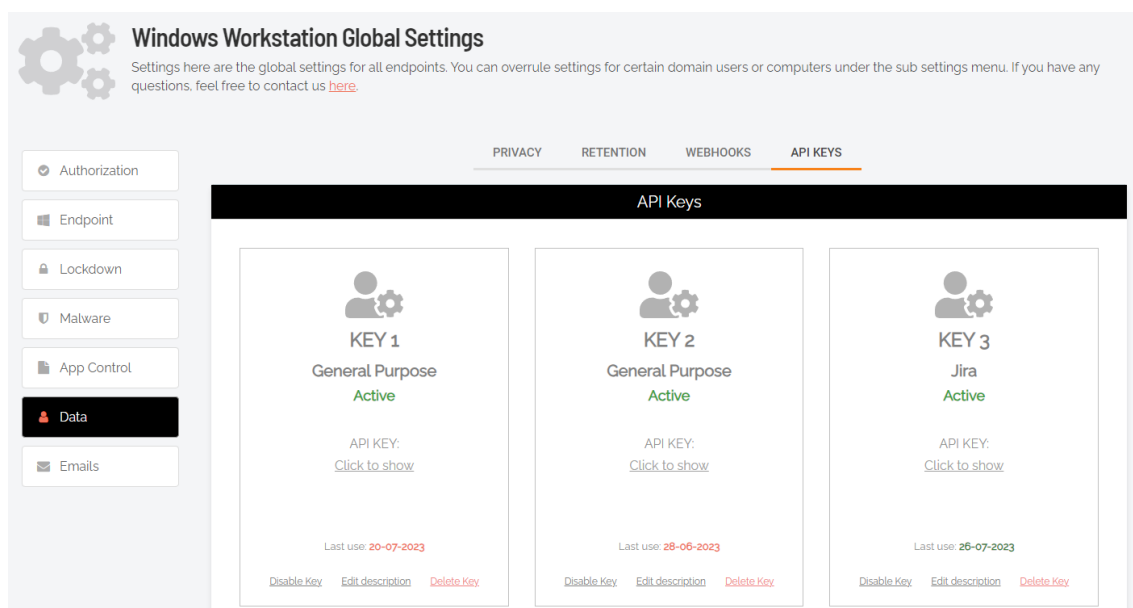
Prerequisites

To complete the setup, you will need the following:

- A. A Jira Service Management username and an Admin By Request Portal username.
You must have administrator access to both Jira Service Management and the ABR Portal.
- B. A Jira Service Management API token associated with your username.
For information on managing API tokens in Jira, refer to <https://support.atlassian.com/atlassian-account/docs/manage-api-tokens-for-your-atlassian-account/>.
- C. An API key in your ABR Portal:

Create an API key

1. Log in to the portal and navigate to **Settings > Windows Settings**.
2. In the left menu, click **Data**, followed by tab **API KEYS**.
3. To create a new API key, click button **Add New**.
4. If you want to name this key, click **Edit description** and name it accordingly.
5. Click **Save** to save the new API key:



Copy the key to the clipboard, ready for pasting in the next step.

Test the API key

The following test uses VS Code to send a curl statement to the ABR server's API interface.

A successful test returns **200 OK**, plus a list of ABR requests in JSON format.

Note that if there are no ABR requests in *any* category (pending, approved, denied or quarantined) then the server will still return **200 OK**, but no JSON data:

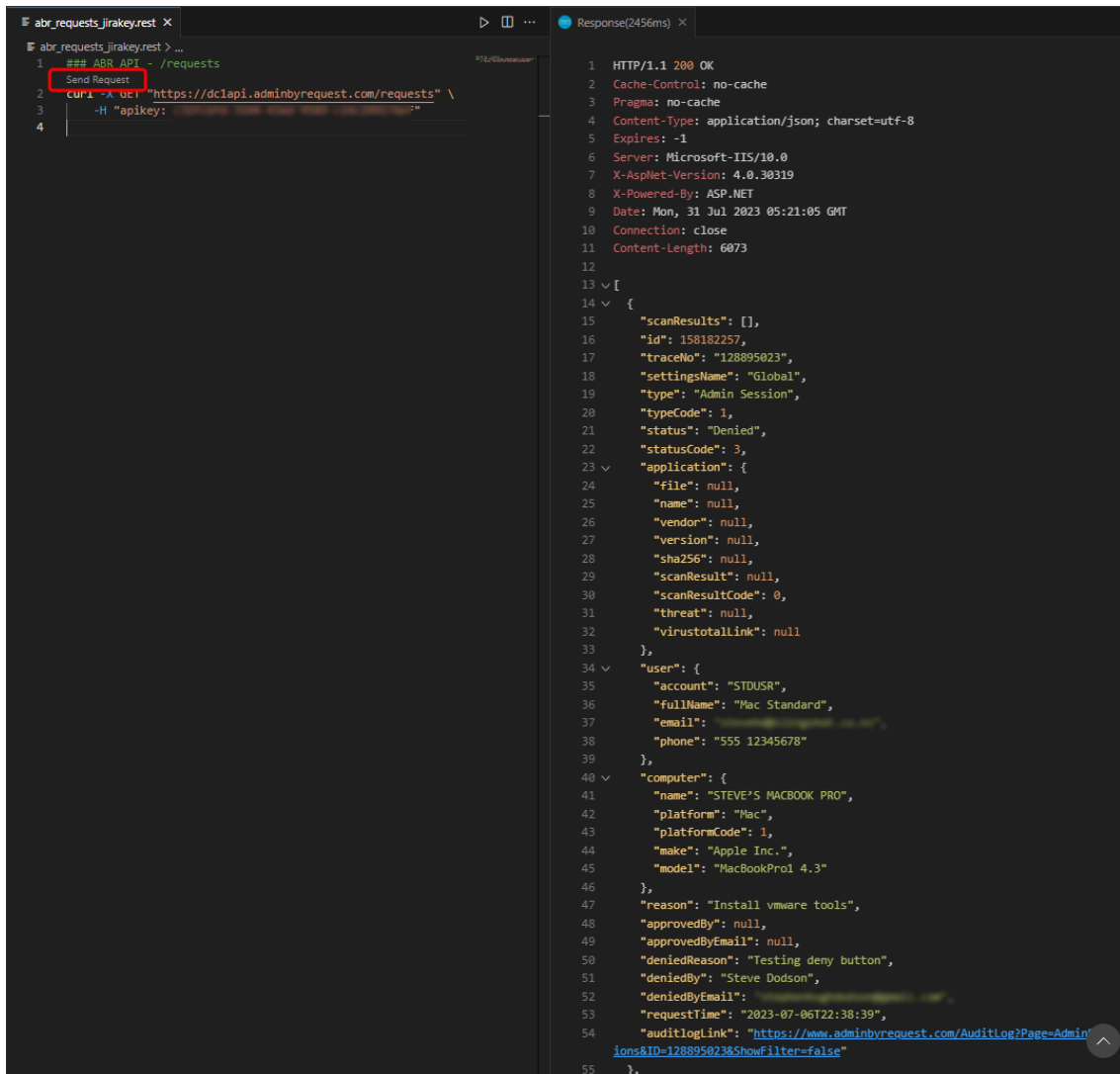
1. In your portal, identify the URL to use for API requests. The example below shows **<https://dc1api.adminbyrequest.com/requests>**, but your URL might be slightly different.

2. Enter the following curl statement, using your URL and your API key:

```
curl -X GET "https://dc1api.adminbyrequest.com/requests" \  
-H "apikey: <your API key>"
```

3. Send the curl statement. The following examples show the response in VS Code from the ABR server using firstly the REST client and secondly the Thunder client.

REST client



```
abr_requests_jirakey.rest x  
#11/11/2023 05:21:05 GMT  
1 ## ABR API - /requests  
2 Send Request  
3 curl -X GET "https://dc1api.adminbyrequest.com/requests" \  
4 -H "apikey: <your API key>"  
Response(2456ms) x  
1 HTTP/1.1 200 OK  
2 Cache-Control: no-cache  
3 Pragma: no-cache  
4 Content-Type: application/json; charset=utf-8  
5 Expires: -1  
6 Server: Microsoft-IIS/10.0  
7 X-AspNet-Version: 4.0.30319  
8 X-Powered-By: ASP.NET  
9 Date: Mon, 31 Jul 2023 05:21:05 GMT  
10 Connection: close  
11 Content-Length: 6073  
12  
13  
14 {  
15   "scanResults": [],  
16   "id": 158182257,  
17   "traceNo": "128895023",  
18   "settingsName": "Global",  
19   "type": "Admin Session",  
20   "typeCode": 1,  
21   "status": "Denied",  
22   "statusCode": 3,  
23   "application": {  
24     "file": null,  
25     "name": null,  
26     "vendor": null,  
27     "version": null,  
28     "sha256": null,  
29     "scanResult": null,  
30     "scanResultCode": 0,  
31     "threat": null,  
32     "virustotalLink": null  
33   },  
34   "user": {  
35     "account": "STDUSR",  
36     "fullName": "Mac Standard",  
37     "email": "mac@adminbyrequest.com",  
38     "phone": "555 12345678"  
39   },  
40   "computer": {  
41     "name": "STEVE'S MACBOOK PRO",  
42     "platform": "Mac",  
43     "platformCode": 1,  
44     "make": "Apple Inc.",  
45     "model": "MacBookPro1 4.3"  
46   },  
47   "reason": "Install vmware tools",  
48   "approvedBy": null,  
49   "approvedByEmail": null,  
50   "deniedReason": "Testing deny button",  
51   "deniedBy": "Steve Dodson",  
52   "deniedByEmail": "steve.dodson@adminbyrequest.com",  
53   "requestTime": "2023-07-06T22:38:39",  
54   "auditlogLink": "https://www.adminbyrequest.com/AuditLog2Page-Admin-  
55   ions&ID=128895023ShowFilter=false"  
56 }  
}
```

Thunder client

The screenshot shows the Thunder Client interface. The top bar displays the request method (GET) and the URL (https://dclapi.adminbyrequest.com/requests). A red box highlights the 'Send' button. Below the URL bar, the 'Headers' tab is selected, showing the following headers:

Header	Value
Accept	*/*
User-Agent	Thunder Client (https://www.thunderclient.com)
apikey	[REDACTED]
header	value

The response tab shows the following JSON response:

```

1  [
2  {
3    "scanResults": [],
4    "id": 158182257,
5    "traceNo": "128895023",
6    "settingsName": "Global",
7    "type": "Admin Session",
8    "typeCode": 1,
9    "status": "Denied",
10   "statusCode": 3,
11   "application": {
12     "file": null,
13     "name": null,
14     "vendor": null,
15     "version": null,
16     "sha256": null,
17     "scanResult": null,
18     "scanResultCode": 0,
19     "threat": null,
20     "virustotalLink": null
21   },
22   "user": {
23     "account": "STDUSR",
24     "fullName": "Mac Standard",
25     "email": "[REDACTED]",
26     "phone": "555 12345678"
27   },
28   "computer": {
29     "name": "STEVE'S MACBOOK PRO",
30     "platform": "Mac",
31     "platformCode": 1,
32     "make": "Apple Inc.",
33     "model": "MacBookPro1 4,3"
34   },
35   "reason": "Install vmware tools",
36   "approvedBy": null,
37   "approvedByEmail": null,
38   "deniedReason": "Testing deny button",
39   "deniedBy": "Steve Dodson",
40   "deniedByEmail": "[REDACTED]",
41   "requestTime": "2023-07-06T22:38:39",
42   "auditlogLink": "https://www.adminbyrequest.com/AuditLog?Page=AdminSessions&ID=128895023&ShowFilter=false"
43   },

```

There are multiple ABR requests returned across all states (pending, approved, denied, etc.) - only the first request is shown. Scroll the response you get back to see all requests.

Once you have this information, we're ready to get started.

There are three main tasks required to configure Jira integration. The next three sections describe each task in detail.

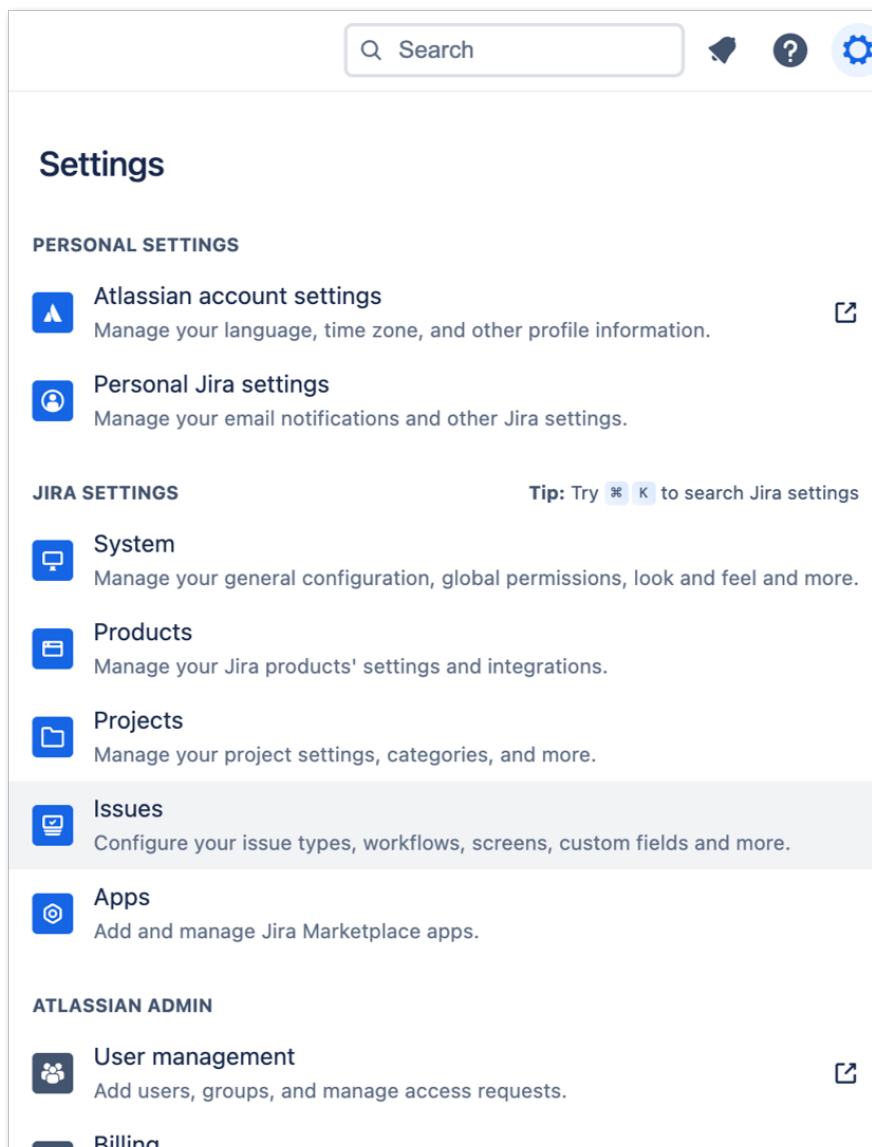
1. Creating custom fields

The integration requires you to set up a few custom fields. These fields hold information about:

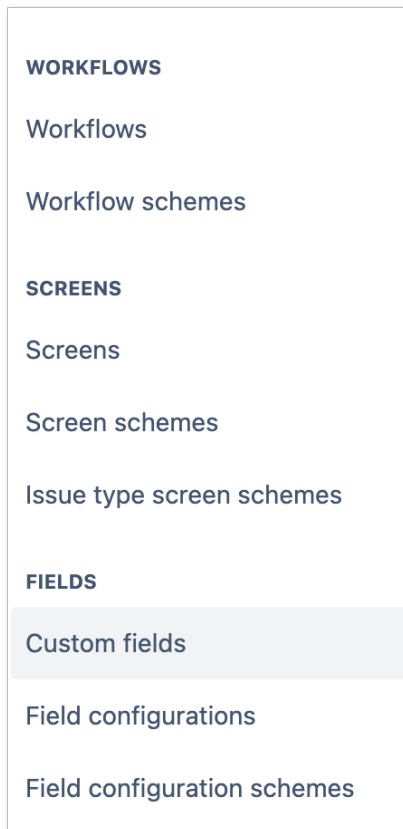
- The ID of the request from Admin By Request (can be used for further automation).
- The name of the approver (if the request is approved from the platform or via another integration).
- The reason supplied if a request is denied.

Create custom fields

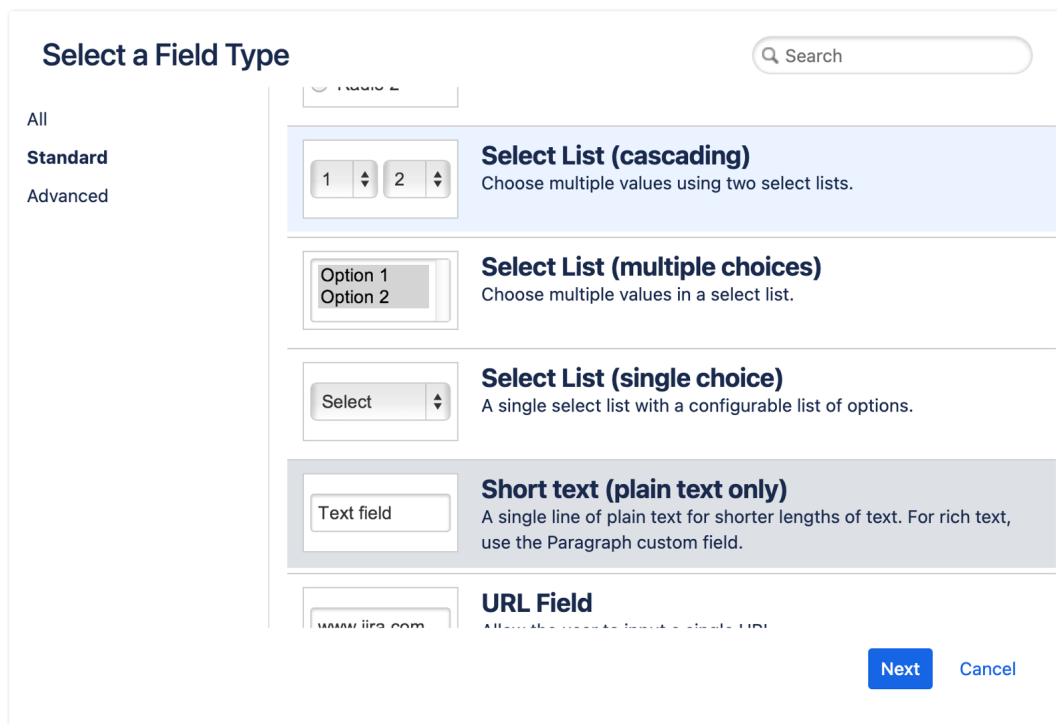
1. Log in to Jira Service Management and click the cog in the upper right corner to navigate to **Settings > Issues**:



- In the left menu, select **Custom fields**:



- In the *Custom fields* screen, click button **Create custom field** and select Field Type **Short text (plain text only)**:



- Name the field (e.g. **ABR Request ID**) and, optionally, provide a description.

5. Repeat these steps for field **ABR Handled by**.
6. Repeat the steps one more time for field **ABR Reason**, but select the multiline option **Paragraph (supports rich text)** as the Field Type.

NOTE:

You don't have to select any specific views for the fields - these will be assigned when creating the *Request Type* in the next section.

7. Finally, find the ids of the fields just created. To do this, make sure at least one issue exists in your project. Identify the issue number and enter the following URL in a browser:

```
https://<your-jira-website>.atlassian.net/rest/api/2/issue/<your-jira-issue-number>?expand=names
```

Copy and paste the resulting JSON into an editor that can format it nicely (e.g. Notepad++) and find the labels of the fields created.

The following example has issue number **ABRJIRA-40** (from the "key" field) and shows both field id (**customfield_10064**) and field name for **ABR Request ID**:

```

4      "self": "https://abr-integration-test.atlassian.net/rest/api/2/issue/10176",
5      "key": "ABRJIRA-40",
6      "names": {
7          "statuscategorychangedate": "Status Category Changed",
8          "fixVersions": "Fix versions",
9          "resolution": "Resolution",
10         "lastViewed": "Last Viewed",
11         "customfield_10061": "Category",
12         "customfield_10062": "Atlas project key",
13         "customfield_10063": "Atlas project status",
14         "customfield_10064": "ABR Request ID",
15         "customfield_10066": "ABR Handled by",
16         "priority": "Priority",
17         "customfield_10067": "ABR Reason",
18         "customfield_10068": "Time to done",
19         "labels": "Labels",
20         "aggregatetimeoriginalestimate": "Σ Original Estimate",
21         "timeestimate": "Remaining Estimate",
22         "versions": "Affects versions",
23         "issuelinks": "Linked Issues",
24         "assignee": "Assignee",
25         "status": "Status",
26         "components": "Components",

```

The field id for ABR Request ID is needed later in task ["Example 2 - Approving or Denying ABR Requests in Jira" on page 20](#).

NOTE:

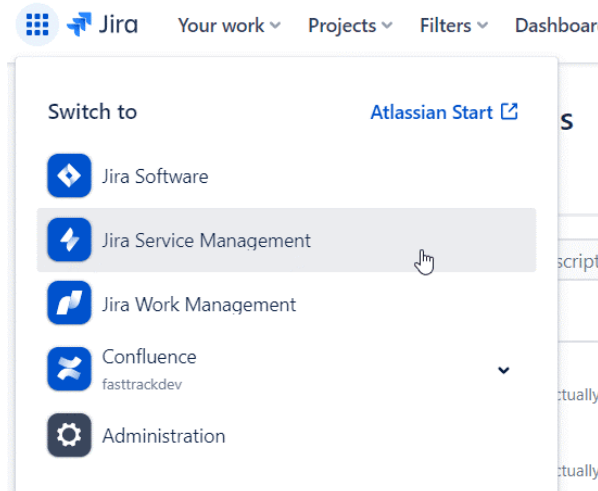
This is an example - your field id(s) will almost certainly be different.

2. Creating a custom Request Type

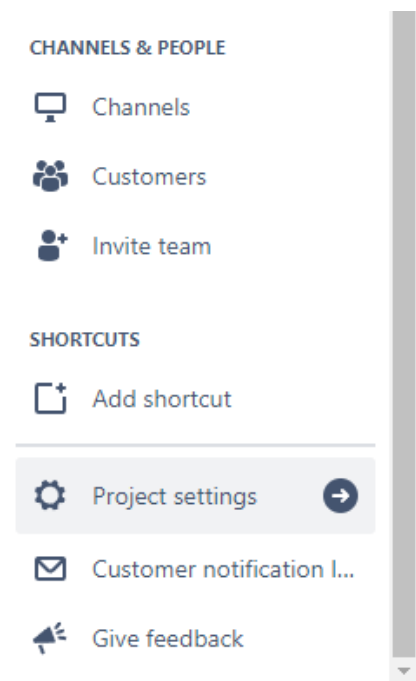
A custom Request Type enables Jira to identify any requests coming from the Admin By Request servers.

Create a custom Request Type

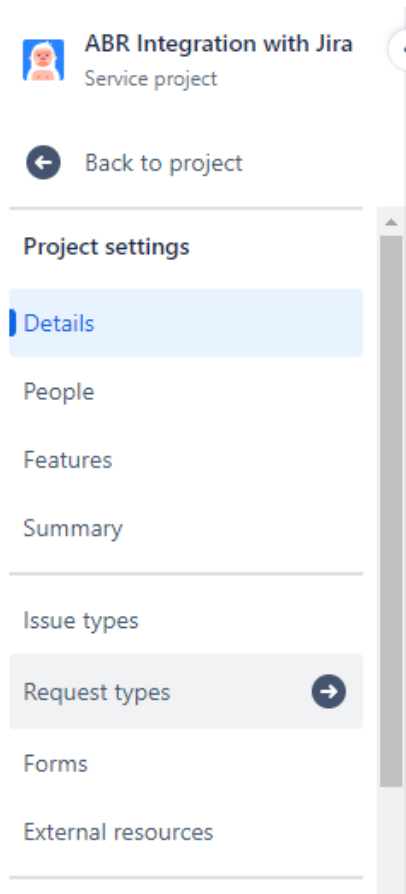
1. Login to Jira and, from the Switch to... menu, select Jira Service Management:



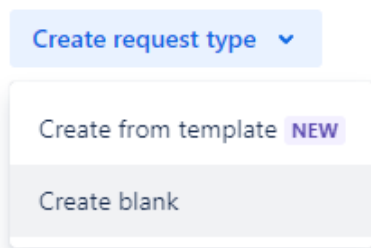
2. From within Jira Service Management, either create a new project or select the project you want to integrate with. Open the project and select **Project settings** towards the bottom of the left menu:



3. From the Project settings menu, select **Request types**:



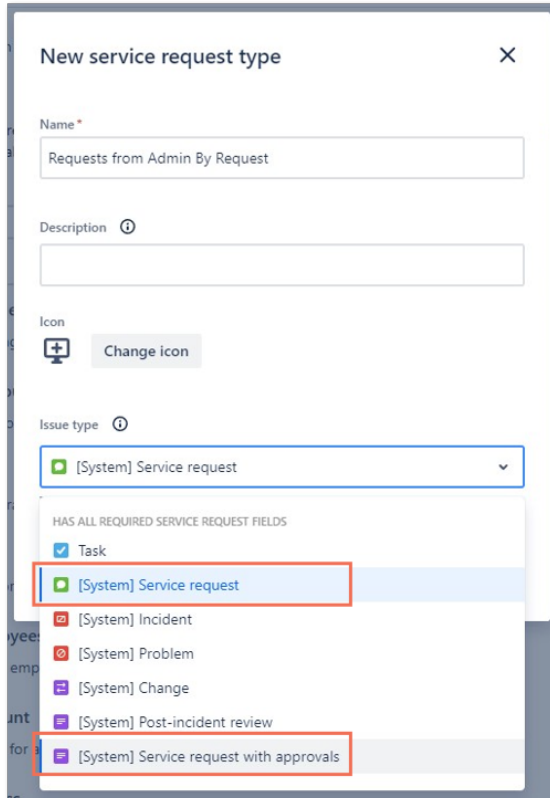
4. On the *Service requests* screen, use the **Create request type** button (top right) to create a request type (click the button and select **Create blank**):



5. Give the request type a name and select the Issue type as either:
- **[System] Service request**
This sends requests to Jira and updates them as they get approved or denied, OR
 - **[System] Service request with approvals**
Works in the same way as the ordinary service request type, but allows designated approvers to approve or deny requests from within Jira.
With a bit of automation, this also enables Jira to call the Admin By RequestAPI to approve or deny requests directly from within Jira.

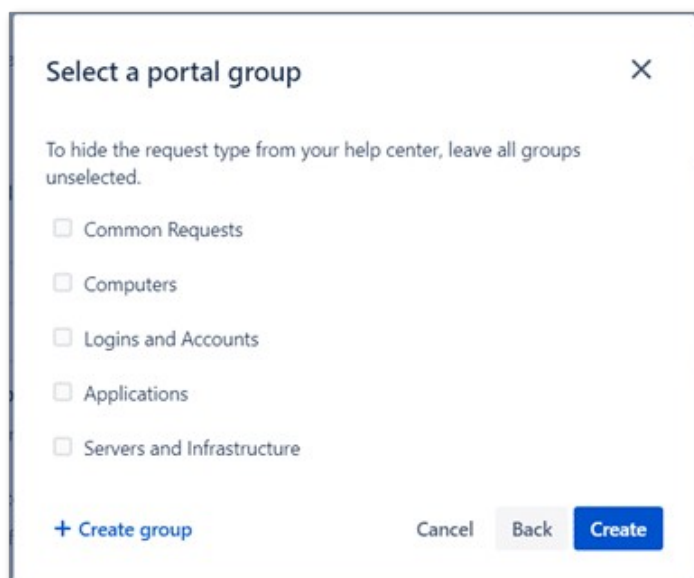
NOTE:

The automation example ("Example 2 - Approving or Denying ABR Requests in Jira" on page 20) requires that **[System] Service request with approvals** is the option chosen at this point:



The screenshot shows the 'New service request type' dialog box. The 'Name' field is filled with 'Requests from Admin By Request'. The 'Issue type' dropdown is open, showing a list of options. The option '[System] Service request with approvals' is highlighted with a red box. Other options include '[System] Service request', '[System] Incident', '[System] Problem', '[System] Change', and '[System] Post-incident review'. The 'Task' option is also checked.

6. Select the portal group to assign to this new request type (or leave all options blank):



The screenshot shows the 'Select a portal group' dialog box. The dialog box is empty, showing a list of portal groups: Common Requests, Computers, Logins and Accounts, Applications, and Servers and Infrastructure. The 'Create group' button is highlighted.

7. Finally, drag and drop the three newly created custom fields, as well as the **Description** field into the view of the new request type:

Request form Issue view Workflow statuses

Requests from Admin By Request

Fields added to the request form are filled out by customers when they raise a request from the portal. [Learn more about the portal](#), or [how to customize fields](#).

Request type description ⓘ

Enter text

☰ Instructions >

Aa Summary REQUIRED ... >

☰ Description ... >

Aa ABR Handled by ... >

☰ ABR Reason ... >

Aa ABR Request ID HIDDEN ... >

Forms
Attach an existing form to this request type, or create a new form using a template. Attach form ▾

NOTE:

- Make sure you are on the **Request form** tab and not *Issue view* or *Workflow statuses*.
- Make sure the **Description** field is added along with the three custom fields. *Summary* and *Instructions* should already be there.
- You can leave the ABR Request ID field as "Use preset value and hide from portal" if you do not wish the ABR Request ID to be visible.

8. Save the changes.
Everything is now ready to create the ABR-Jira integration.

3. Integrating Jira with Admin By Request

Now that the pre-requisites are in place, head over to <https://jira.adminbyrequest.com> to start the setup process.

Setup Jira integration

1. From the *Set up Jira integration* screen, check once again that you've completed all the prerequisite steps and click button **Start setup**:
2. Retrieve your Atlassian API Token as well as your Admin By Request API key and enter these into the setup form alongside your Atlassian username and instance URL:



Set up Jira integration - Information

Please fill in the information below to set up the integration with Jira.

You Atlassian username and API token can be generated from your Atlassian profile page. The Admin By Request API key can be generated from the Admin By Request portal.

Atlassian URL

The URL of you Atlassian setup (e.g. <https://myorganization.atlassian.net>).

Username

The username of the Atlassian user to use for API calls.

Atlassian API token

The API token to use for the API calls.

Admin By Request API key

API key generated from the Admin By Request portal.


[Continue →](#)

NOTE:

Your username needs to be one that is authorized to make API calls. It is generally the email address recorded under your profile at the top right of the Jira Service Management screen:



3. Next, you are prompted to select the Jira service desk to associate with the integration:



Set up Jira integration - Service desk

Select the service desk in Jira to associate with the integration.

Service desk

ABR Integration with Jira

✓ Select service desk

If you have more than one service desk project in Jira, a drop-down arrow will appear at the right of the *Service desk* field, allowing you to select the one you want.

4. The final screen is where the custom fields you created are associated with the three required by the ABR integration. *Request type* is already selected for you - choose the other three to match your custom fields:

Request type

ABR Request for Access

New requests will be created as this request type in Jira.

Handled by custom field

ABR Handled by

The custom field to show who has handled the request from Admin By Request.

Reasons custom field

ABR Reason

The custom field to show the denied reason for the request.

Request ID custom field

ABR Request ID

The custom field to use for the request ID (used for further automation).

✓ Finish installation

5. Click **Finish installation** to complete setting up the Jira integration.

New requests from Admin By Request will now be sent to your Jira instance as service requests.

You can check the status of your Jira integration via the portal (**Settings > Integrations > Jira**):



The screenshot shows the 'Jira Integrations' page. The main heading is 'Jira Integrations'. Below it, there is a section titled 'Integrations' with a diamond icon. This section contains a table with the following data:

Name	Functional	
abr-integration-test.atlassian.net	<input checked="" type="checkbox"/>	Delete

Below the table is a red button labeled 'New Jira Integration'. To the right of the table is a section titled 'About Jira Integrations' with a gear icon. It contains the following text:

The Jira integration will allow your team to interact with Admin By Request directly via Jira - giving access to features like:

- Receive incoming requests directly in Jira.
- Approve and deny requests directly from Jira.

Refer to [this page](#) for documentation.

Updating Jira issues

When a user makes a request via ABR for either an "Admin Session" or to execute something "Run As Admin", a Jira issue is created in your selected service desk instance.

NOTE:

It can take several minutes after the ABR request is submitted for the issue to be created in Jira.

The Jira *Summary* indicates the type of request: **Admin session** or **Run as admin**. This field is updated once the request is either APPROVED or DENIED.

For example, the following list of issues shows one "Run as admin request" waiting for approval, two "Admin session" requests approved and one admin session denied:

T	Key	Summary
	ABRJIRA-41	Run as admin request - Notepad++ : a free (GNU) source code editor
	ABRJIRA-40	APPROVED - Admin session request
	ABRJIRA-39	DENIED - Admin session request
	ABRJIRA-38	APPROVED - Admin session request

What next?

The preceding sections cover the extent of the Admin By Request Jira integration. If you want to do more with Jira issues once they have been created, you can use Jira automation to populate issues with more information or to carry out actions such as email notifications.

IMPORTANT:

The examples provided in the next section are for convenience only - we do not support any Jira automation rules and they must be used entirely at the customer's own risk.

If you do decide to work with Jira automation, note the following:

- If you selected **Service request** as the Request Type (see ["2. Creating a custom Request Type" on page 8](#)), you can use simple automation to interrogate the Summary field to carry out further actions (illustrated in ["Example 1 - Populating issue fields after issue creation" on page 17](#)).
- If you selected **Service request with Approval** as the Request Type, you can use more advanced automation, as illustrated in ["Example 2 - Approving or Denying ABR Requests in Jira" on page 20](#).

Creating Jira automation rules

Using the Jira Audit log

The Jira Audit log is a very useful tool when working with Jira automation, especially as an aid in troubleshooting.

Each rule has its own Audit Log:

The screenshot shows the Jira Audit Log for the rule "ABR-Outgoing-Approval-Accepted-or-Declined". The rule is enabled. The audit log shows two entries for the date 1/23/24:

Date	Rule	Scope	Status	Duration	Operations
01/23/24, 11:13:12 pm (26539661535)	ABR-Outgoing-Approval-Accepted-or-Declined	ABR Integration with Jira	SUCCESS	6.40s	Show more
01/23/24, 10:54:40 pm (26538753223)	ABR-Outgoing-Approval-Accepted-or-Declined	ABR Integration with Jira	SOME ERRORS	5.64s	Show less

The "Action details" section for the "SOME ERRORS" entry shows the following actions:

- If block:** The following issues did not match the condition: ABRJIRA-39
- Send web request:** Successfully published web request
- Comment on issue:** Comment added to issue
ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}} Reason: {{issue.fields.customfield_10065}}
- Transition issue:** Destination status could not be resolved. If using a smart-value ensure this resolves to a numeric status ID or untranslated name for issues (with current status): ABRJIRA-39 (Resolved - 5)
- If/else block:**

The "Associated items" section shows: Approval completed ABRJIRA-39

Linking back to the ABR Portal

A future release of the Jira integration will almost certainly include a link back to the ABR Portal "Requests" page, from where requests can be approved or denied. Until this is available, *Example 1* below provides a workaround that customers can implement using a Jira rule (Action 1 - set fields, Description).

Example 1 - Populating issue fields after issue creation

Issue created from ABR ENABLED ✓ Audit log Rule details Update Return to rules

+ When: Issue created
Rule is run when an issue is created.

⚙️ Request Type equals
ABR Request for Access

✎ Then: Edit issue fields
Description, Reporter, Approvers

🔗 And: Transition the issue to
WAITING FOR APPROVAL

✉️ And: Send email
myemailaddress@mycompany.com
Issue {{issue.key}} just created and waiting for approval

+ Add component

Rule details

Name*

Description

Scope

Owner*

The owner will receive emails when the rule fails.

Actor*

Actions defined in this rule will be performed by the user selected as the actor. [Learn more about rule actors in automation.](#)

Notify on error

Who can edit this rule?*

All admins

Check to allow other rule actions to trigger this rule. Only enable this if you need this rule to execute in response to another rule.

1. Trigger:

+ When: Issue created
Rule is run when an issue is created.

⚙️ Request Type equals
ABR Request for Access

+ Issue created 🗑️

Rule is run when an issue is created. This trigger needs no configuration.

2. IF Request Type is **ABR Request for Access**:

The screenshot shows a rule configuration interface. On the left, a vertical flow of components is shown: 'When: Issue created', 'Request Type equals ABR Request for Access' (highlighted with a blue border), 'Then: Edit issue fields Description, Reporter, Approvers', and 'And: Transition the issue to WAITING FOR APPROVAL'. On the right, the 'Issue fields condition' configuration panel is open, showing 'Request Type' as the field and 'equals' as the condition. The value field is set to 'ABR Request for Access'. 'Back' and 'Next' buttons are visible at the bottom.

3. Action 1 - set fields:

The screenshot shows a rule configuration interface. On the left, a vertical flow of components is shown: 'When: Issue created', 'Request Type equals ABR Request for Access', 'Then: Edit issue fields Description, Reporter, Approvers' (highlighted with a blue border), 'And: Transition the issue to WAITING FOR APPROVAL', and 'And: Send email myemailaddress@mycompany.com Issue {{issue.key}} just created and waiting for approval'. At the bottom left is an 'Add component' button. On the right, the 'Edit issue' configuration panel is open. It shows a 'Choose fields to set...' dropdown, and fields for 'Description' (with a text area containing a URL and instructions), 'Reporter' (set to {{issue.user}}, and 'Approvers' (set to 'Jo Approver'). A 'More options' link is also present. 'Back' and 'Next' buttons are visible at the bottom.

4. Action 2 - Set Status to **Waiting for Approval** (possibly optional, depending on workflow):

The screenshot shows a workflow editor with five conditions in a vertical sequence:

- When: Issue created** (Rule is run when an issue is created.)
- Request Type equals** (ABR Request for Access)
- Then: Edit issue fields** (Description, Reporter, Approvers)
- And: Transition the issue to** (WAITING FOR APPROVAL) - This step is highlighted with a blue border.
- And: Send email** (myemailaddress@mycompany.com, Issue {{issue.key}} just created and waiting for approval)

The detailed view of the **Transition issue** action is shown on the right:

- Transition issue** (with copy and delete icons)
- Description: Transitions an issue from one status to another, through a workflow. [Learn more about Transition issue action](#)
- Choose the status to transition the issue to:
- Destination status** dropdown menu: **WAITING FOR APPROVAL**
- Ensure a transition from the issue's source status to your selected destination status exists; [more info](#).
- + add regex to distinguish between multiple transitions to the same status
- Choose fields to set...** dropdown menu
- > More options
- Buttons: **Back** and **Next**

5. Action 3 - Send email:

The screenshot shows a workflow editor with five conditions in a vertical sequence:

- When: Issue created** (Rule is run when an issue is created.)
- Request Type equals** (ABR Request for Access)
- Then: Edit issue fields** (Description, Reporter, Approvers)
- And: Transition the issue to** (WAITING FOR APPROVAL)
- And: Send email** (myemailaddress@mycompany.com, Issue {{issue.key}} just created and waiting for approval) - This step is highlighted with a blue border.

At the bottom of the editor is a button: **+ Add component**

The detailed view of the **Send email** action is shown on the right:

- Send email** (with copy and delete icons)
- [Learn more about Send email action](#)
- To *** dropdown menu: myemailaddress@mycompany.com x
- Cc Bcc**
- Subject *** text field: Issue {{issue.key}} just created and waiting for approval
- Content *** text area: Issue: {{issue.description}}
Reason: {{issue.reason}}
Requestor: {{issue.reporter}}
- > More options
- Buttons: **Back** and **Next**

Example 2 - Approving or Denying ABR Requests in Jira

ABR-Outgoing-Approval-Accepted-or-Declined ENABLED ✓ Audit log Rule details Update Return to rules

When: Approval completed
Rule is run when an approval is accepted or declined

If: matches
{{approval.decision}} equals Approved

Then: Send web request
PUT
https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}

And: Add comment to issue
ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}}

And: Transition the issue to
RESOLVED

Rule details

Name *
ABR-Outgoing-Approval-Accepted-or-Declined

Description
When a request approval is accepted or declined ...

Scope
Single project

Projects *
ABR Integration with Jira (ABRJIRA)

Owner *
Steve Dodson

The owner will receive emails when the rule fails.

Actor *
Automation for Jira

Actions defined in this rule will be performed by the user selected as the actor. [Learn more about rule actors in automation.](#)

Notify on error
E-mail rule owner once when rule starts faili...

Who can edit this rule? *
All admins

Check to allow other rule actions to trigger this

1. Trigger:

When: Approval completed
Rule is run when an approval is accepted or declined

Approval completed
For Jira Service Management only. Rule is run when an approval is accepted or declined. This trigger needs no configuration.

Back Next

2. IF Approved:

When: Approval completed
Rule is run when an approval is accepted or declined

IF

If: matches
{{approval.decision}} equals Approved

Then: Send web request
PUT
https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}

And: Add comment to issue
ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}}

If block

The if block executes the actions within that block when the all specified conditions matches. Otherwise, the following else blocks will be evaluated. [Learn more about If / else block condition](#)

Run actions if...

All conditions match

At least one condition matches

Conditions

> If: matches {{approval.decision}} equals Approved

AND

+ Add conditions...

Back Next

3. Action 1 - Send web request (PUT). Important - make sure you read up on API calls [here \(API Overview\)](#) and [here \(Requests API\)](#):

When: Approval completed
Rule is run when an approval is accepted or declined

IF

If: matches
{{approval.decision}} equals Approved

Then: Send web request
PUT
https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}

And: Add comment to issue
ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}}

And: Transition the issue to
RESOLVED

+ Add component

ELSE IF

Send web request

This action will send a HTTP request to the url specified. [Learn more](#)

Web request URL *
https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}

Request parameters must be url encoded, smart values should use: {{value.urlEncode}}.

HTTP method *
PUT

Web request body *
Empty

Delay execution of subsequent rule actions until we've received a response for this web request

Headers (optional)

Key	Value	Hidden
apikey	c32fcbfd-3144-43	<input type="checkbox"/>
Content-length	0	<input type="checkbox"/>

+ Add another header

> Validate your web request configuration

Back Next

> How do I access web request response values in subsequent rule actions?

4. Action 2 - Add Comment to Jira issue (Important - make sure you identify the correct customfield ids - yours might be different from the example):

The screenshot shows a Jira automation rule configuration. On the left, a flowchart shows the rule's logic: 'When: Approval completed' (Rule is run when an approval is accepted or declined) leads to an 'IF' condition 'If: matches' ({{approval.decision}} equals Approved). This is followed by 'Then: Send web request' (PUT https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}) and finally 'And: Add comment to issue' (ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}}). The 'Add comment to issue' step is highlighted with a blue border.

On the right, the configuration for the 'Comment on issue' action is shown. It includes a title 'Comment on issue', a link to 'Learn more about Comment on issue action', and a prompt 'Please enter the comment to add:'. The 'Comment' field contains the text: 'ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}}'. There is a checked checkbox for 'Prevent duplicates by only adding this comment once to a particular issue.' and a 'Comment Visibility' dropdown menu. 'Back' and 'Next' buttons are at the bottom.

5. Action 3 - Set Status to **Resolved**:

The screenshot shows a Jira automation rule configuration. On the left, a flowchart shows the rule's logic: 'When: Approval completed' (Rule is run when an approval is accepted or declined) leads to an 'IF' condition 'If: matches' ({{approval.decision}} equals Approved). This is followed by 'Then: Send web request' (PUT https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}) and finally 'And: Add comment to issue' (ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}}). The 'Add comment to issue' step is highlighted with a blue border.

On the right, the configuration for the 'Transition issue' action is shown. It includes a title 'Transition issue', a link to 'Learn more about Transition issue action', and a prompt 'Choose the status to transition the issue to:'. The 'Destination status' dropdown menu is set to 'RESOLVED'. Below this, there is a note: 'Ensure a transition from the issue's source status to your selected destination status exists; more info.' and a link '+ add regex to distinguish between multiple transitions to the same status'. There is a 'Choose fields to set...' dropdown menu and a 'More options' link. 'Back' and 'Next' buttons are at the bottom.

6. IF Declined:

The screenshot displays the workflow editor for an 'ELSE IF' block. The main workspace on the left contains four components:

- Else-if: matches** (highlighted with a blue border): `{{approval.decision}} equals Declined`
- Then: Send web request**: DELETE `https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}`
- And: Add comment to issue**: ABR Request: `{{issue.fields.customfield_10064}}` Value of Approval: `{{approval.decision}}` Reason: `{{issue.fields.customfield_10065}}`
- And: Transition the issue to RESOLVED**

At the bottom left is an '+ Add component' button.

The right-hand panel provides details for the selected 'Else block':

- Else block** (with copy and delete icons)
- Description:** The else block executes the actions within the block if the preceding if/else-if blocks failed to match their conditions. The else block can contain its own conditions, making it an else-if, or it can be left blank if it is the last else block. [Learn more about If / else block condition](#)
- Run actions if..**:
 - All conditions match
 - At least one condition matches
- Conditions:**
 - `{{approval.decision}} equals Declined` (with copy and delete icons)
 - AND
 - + Add conditions...
- Navigation:** Back and Next buttons.

7. Action 1 - Send web request (DELETE). Important - make sure you read up on API calls [here](#) (API Overview) and [here](#) (Requests API):

ELSE IF

Else-if: matches
 {{approval.decision}} equals Declined

Then: Send web request
 DELETE
 https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}

And: Add comment to issue
 ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}} Reason: {{issue.fields.customfield_10065}}

And: Transition the issue to
RESOLVED

+ Add component

+ Add else

+ Add component

Send web request

This action will send a HTTP request to the url specified. [Learn more](#)

Web request URL *

Request parameters must be url encoded, smart values should use: {{value.urlEncode}}.

HTTP method *

DELETE

Web request body *

Empty

Delay execution of subsequent rule actions until we've received a response for this web request

Headers (optional)

Key	Value	Hidden
apikey	c32fcbfd-3144-43e	<input type="checkbox"/>

+ Add another header

› Validate your web request configuration

Back
Next

› How do I access web request response values in subsequent rule actions?

8. Action 2 - Add Comment to Jira issue (Important - make sure you identify the correct customfield ids as described in "1. Creating custom fields" on page 5 - even if your field names are the same as the example, your customfield ids might be different):

The screenshot shows the configuration for Action 2: "Add comment to issue". On the left, the rule flow is shown under an "ELSE IF" condition. The steps are:

- Else-if: matches** (Condition): `{{approval.decision}} equals Declined`
- Then: Send web request** (Action): `DELETE https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}`
- And: Add comment to issue** (Action): `ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}} Reason: {{issue.fields.customfield_10065}}`
- And: Transition the issue to** (Action): `RESOLVED`

On the right, the "Comment on issue" configuration panel is shown. It includes a "Comment *" text area with the following content:

```
ABR Request: {{issue.fields.customfield_10064}}
Value of Approval: {{approval.decision}}
Reason: {{issue.fields.customfield_10065}}
```

There is a checked checkbox for "Prevent duplicates by only adding this comment once to a particular issue." and a "Comment Visibility" dropdown menu. "Back" and "Next" buttons are at the bottom.

9. Action 3 - Set Status to **Resolved**:

The screenshot shows the configuration for Action 3: "Transition issue". On the left, the rule flow is shown under an "ELSE IF" condition. The steps are:

- Else-if: matches** (Condition): `{{approval.decision}} equals Declined`
- Then: Send web request** (Action): `DELETE https://dc1api.adminbyrequest.com/requests/{{issue.fields.customfield_10064}}`
- And: Add comment to issue** (Action): `ABR Request: {{issue.fields.customfield_10064}} Value of Approval: {{approval.decision}} Reason: {{issue.fields.customfield_10065}}`
- And: Transition the issue to** (Action): `RESOLVED`

On the right, the "Transition issue" configuration panel is shown. It includes a "Destination status" dropdown menu with "RESOLVED" selected. Below the dropdown, there is a note: "Ensure a transition from the issue's source status to your selected destination status exists; more info." and a link: "+ add regex to distinguish between multiple transitions to the same status". There is also a "Choose fields to set..." dropdown menu and a "More options" link. "Back" and "Next" buttons are at the bottom.