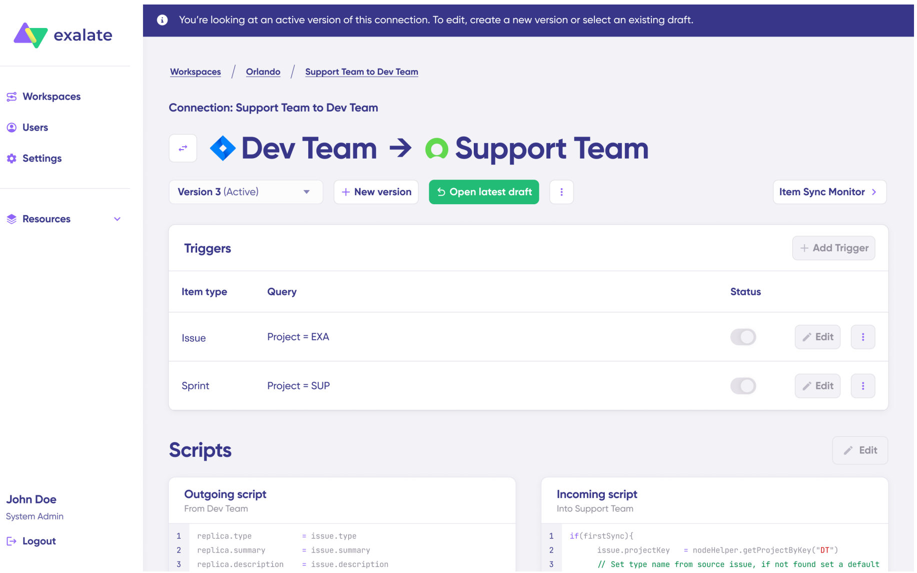
Using Aida AI for Script Configuration

Last Modified on 12/18/2025 4:15 am EST

Aida is Exalate's AI-powered integration assistant embedded into the Exalate script editor to help you generate and modify synchronization scripts through natural language prompts. Instead of writing Groovy code manually, you can describe what you want to achieve, and Aida will draft the code for you.

Accessing Aida

1. Navigate to your connection
2. Click **Add new version** (or open the latest draft version)
3. Click **Edit** to enter the script editor



You're looking at an active version of this connection. To edit, create a new version or select an existing draft.

Connection: Support Team to Dev Team

Triggers

Item type	Query	Status
Issue	Project = EXA	
Sprint	Project = SUP	

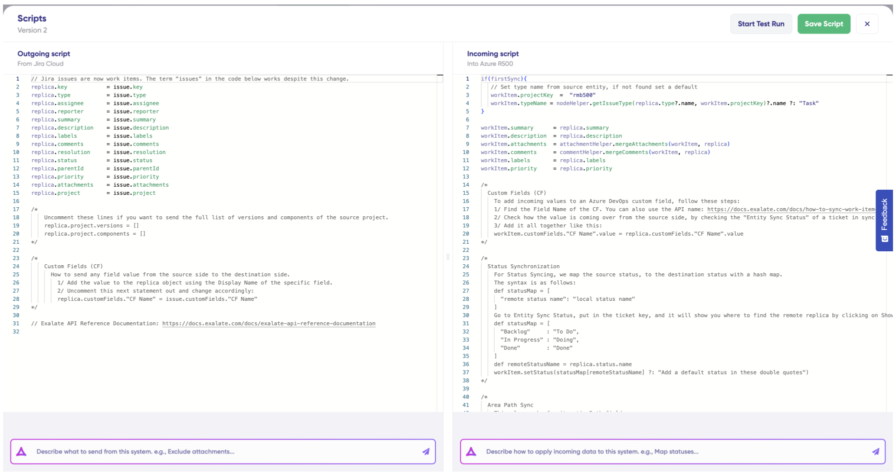
Outgoing script
From Dev Team

```
replica.type            = issue.type
replica.summary         = issue.summary
replica.description     = issue.description
```

Incoming script
Into Support Team

```
if(firstSync){
      issue.projectKey   = nodeHelper.getProjectByKey("DT")
      // Set type name from source issue, if not found set a default
```

You'll see Aida chat interfaces at the bottom of both the **Outgoing script** and **Incoming script** sections. Each Aida chat works independently, allowing you to configure both sides of your connection simultaneously.



Outgoing script
From Jira Cloud

```
// Jira issues are now work items. The term “issues” in the code below works despite this change.
replica.key            = issue.key
replica.type           = issue.type
replica.assignee       = issue.assignee
replica.reporter       = issue.reporter
replica.summary        = issue.summary
replica.description    = issue.description
replica.labels         = issue.labels
replica.comments       = issue.comments
replica.resolution     = issue.resolution
replica.status         = issue.status
replica.parentId       = issue.parentId
replica.priority       = issue.priority
replica.attachments    = issue.attachments
replica.project        = issue.project

/*
    Uncomment these lines if you want to send the full list of versions and components of the source project.
    replica.project.versions = []
    replica.project.components = []
*/

/*
    Custom Fields (CF)
      How to send any field value from the source side to the destination side.
        1/ Add the value to the replica object using the Display Name of the specific field.
        2/ Uncomment this next statement out and change accordingly:
        replica.customFields."CF Name" = issue.customFields."CF Name"
*/

// Exalate API Reference Documentation: https://docs.exalate.com/docs/exalate-api-reference-documentation

```

Describe what to send from this system. e.g., Exclude attachments...

Incoming script
Into Azure R500

```
if(firstSync){
   // Set type name from source entity, if not found set a default
   workItem.projectKey  =  "rmb500"
   workItem.typeName = nodeHelper.getIssueType(replica.type?.name, workItem.projectKey)?.name ?: "Task"
}

workItem.summary      = replica.summary
workItem.description  = replica.description
workItem.attachments  = attachmentHelper.mergeAttachments(workItem, replica)
workItem.comments     = commentHelper.mergeComments(workItem, replica)
workItem.labels       = replica.labels
workItem.priority     = replica.priority

/*
  Custom Fields (CF)
    To add incoming values to an Azure DevOps custom field, follow these steps:
    1/ Find the Field Name of the CF. You can also use the API name: https://docs.exalate.com/docs/how-to-sync-work-item
    2/ Check how the value is coming over from the source side, by checking the "Entity Sync Status" of a ticket in sync
    3/ Add it all together like this:
    workItem.customFields."CF Name".value = replica.customFields."CF Name".value
*/

/*
  Status Synchronization
    For Status Syncing, we map the source status, to the destination status with a hash map.
    The syntax is as follows:
    def statusMap = [
      "remote status name": "local status name"
    ]
    Go to Entity Sync Status, put in the ticket key, and it will show you where to find the remote replica by clicking on Show
    def statusMap = [
      "Backlog"     : "To Do",
      "In Progress" : "Doing",
      "Done"        : "Done"
    ]
    def remoteStatusName = replica.status.name
    workItem.setStatus(statusMap[remoteStatusName] ?: "Add a default status in these double quotes")
*/

/*
  Area Path Sync
```

Describe how to apply incoming data to this system. e.g., Map statuses...

How Aida Works

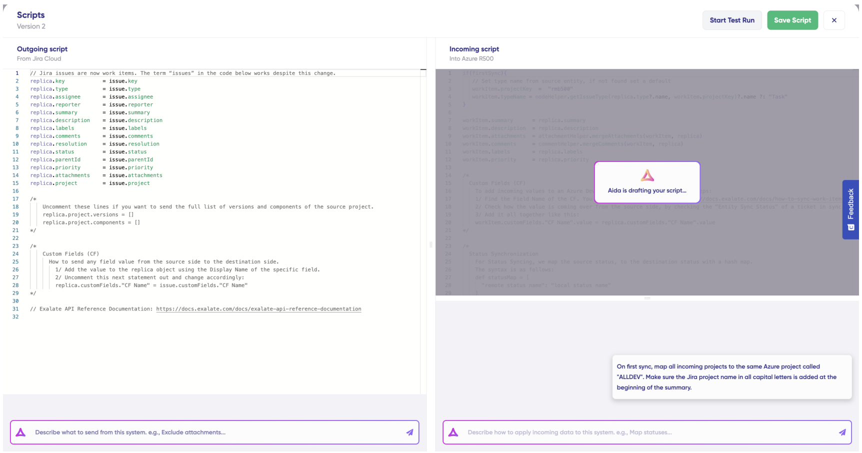
Aida helps you in two ways:

- **For Outgoing scripts**: Describe what data should leave your system. For example, "Exclude attachments" or "Only sync high-priority issues."
- **For Incoming scripts**: Describe how incoming data should be applied to your system. For example, "Map statuses" or "Set a default assignee if the user can't be found."

Using Aida

Submitting a Prompt

1. In the Aida input field, type your request in plain language
2. Click the send button (or press Enter)
3. Aida will begin drafting your script


Scripts
Version 2
Start Test Run
Save Script
×
Outgoing script
From Jira Cloud
Incoming script
Into Azure R500
Aida is drafting your script...
Feedback
Describe what to send from this system. e.g., Exclude attachments...
Describe how to apply incoming data to this system. e.g., Map statuses...

```
// Jira issues are now work items. The term “issues” in the code below works despite this change.
replica.key            = issue.key
replica.type           = issue.type
replica.assignee       = issue.assignee
replica.reporter       = issue.reporter
replica.summary        = issue.summary
replica.description    = issue.description
replica.labels         = issue.labels
replica.comments       = issue.comments
replica.resolution     = issue.resolution
replica.status         = issue.status
replica.parentId       = issue.parentId
replica.priority       = issue.priority
replica.attachments    = issue.attachments
replica.project        = issue.project

/*
    Uncomment these lines if you want to send the full list of versions and components of the source project.
    replica.project.versions = []
    replica.project.components = []
*/

/*
    Custom Fields (CF)
      How to send any field value from the source side to the destination side.
        1/ Add the value to the replica object using the Display Name of the specific field.
        2/ Uncomment this next statement out and change accordingly:
        replica.customFields."CF Name" = issue.customFields."CF Name"
*/

// Exalate API Reference Documentation: https://docs.exalate.com/docs/exalate-api-reference-documentation

```

```
if(firstSync){
   // Set type name from source entity, if not found set a default
   workItem.projectKey  =  "rmb500"
   workItem.typeName = nodeHelper.getIssueType(replica.type?.name, workItem.projectKey)?.name ?: "Task"
}

workItem.summary      = replica.summary
workItem.description  = replica.description
workItem.attachments  = attachmentHelper.mergeAttachments(workItem, replica)
workItem.comments     = commentHelper.mergeComments(workItem, replica)
workItem.labels       = replica.labels
workItem.priority     = replica.priority

/*
  Custom Fields (CF)
    To add incoming values to an Azure Dev                              eps:
    1/ Find the Field Name of the CF. You                              //docs.exalate.com/docs/how-to-sync-work-item
    2/ Check how the value is coming over from the source side, by checking the "Entity Sync Status" of a ticket in sync
    3/ Add it all together like this:
    workItem.customFields."CF Name".value = replica.customFields."CF Name".value
*/

/*
  Status Synchronization
    For Status Syncing, we map the source status, to the destination status with a hash map.
    The syntax is as follows:
    def statusMap = [
      "remote status name": "local status name"
    ]
```

On first sync, map all incoming projects to the same Azure project called "ALLDEV". Make sure the Jira project name in all capital letters is added at the beginning of the summary.

Example prompts:

- "Exclude attachments from sync."
- "Map priority field to remote system."
- "On first sync, map all incoming projects to the same Azure project called 'ALLDEV'"
- "Set a Reporter/Assignee from the source side, if the user can't be found, set a default use.r"

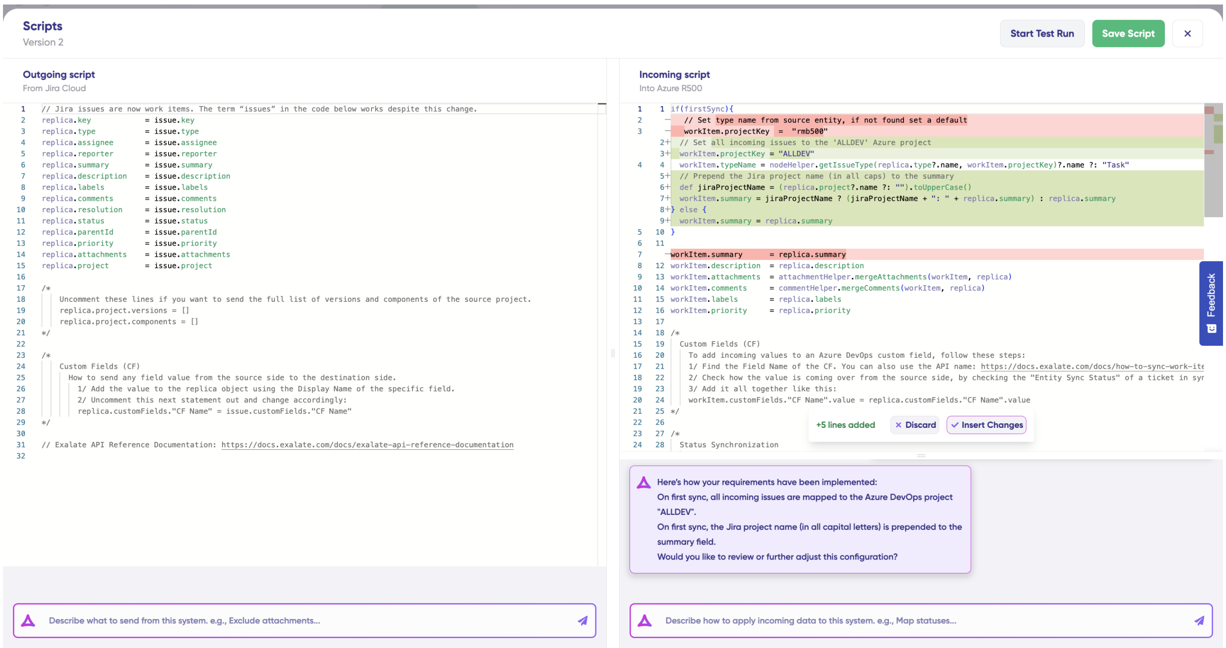
Reviewing Aida's Changes

Once Aida finishes, you'll see:

- **Green highlighting**: New lines that will be added
- **Red highlighting**: Lines that will be removed
- A floating panel with **Insert** and **Discard** buttons

- Aida's explanation of the changes in the chat

The script editor becomes read-only during review, so you must choose to either insert or discard the changes before making manual edits.



```
// Jira issues are now work items. The term “issues” in the code below works despite this change.
replica.key           = issue.key
replica.type          = issue.type
replica.assignee      = issue.assignee
replica.reporter      = issue.reporter
replica.summary       = issue.summary
replica.description   = issue.description
replica.labels        = issue.labels
replica.comments      = issue.comments
replica.resolution    = issue.resolution
replica.status        = issue.status
replica.parentId      = issue.parentId
replica.priority      = issue.priority
replica.attachments   = issue.attachments
replica.project       = issue.project

/*
    Uncomment these lines if you want to send the full list of versions and components of the source project.
    replica.project.versions = []
    replica.project.components = []
*/

/*
    Custom Fields (CF)
      How to send any field value from the source side to the destination side.
        1/ Add the value to the replica object using the Display Name of the specific field.
        2/ Uncomment this next statement out and change accordingly:
        replica.customFields."CF Name" = issue.customFields."CF Name"
*/

// Exalate API Reference Documentation: https://docs.exalate.com/docs/exalate-api-reference-documentation
```

```
if(firstSync){
-   // Set type name from source entity, if not found set a default
-   workItem.projectKey  =  "rmb500"
+  // Set all incoming issues to the 'ALLDEV' Azure project
+  workItem.projectKey = "ALLDEV"
   workItem.typeName = nodeHelper.getIssueType(replica.type?.name, workItem.projectKey)?.name ?: "Task"
+  // Prepend the Jira project name (in all caps) to the summary
+  def jiraProjectName = (replica.project?.name ?: "").toUpperCase()
+  workItem.summary = jiraProjectName ? (jiraProjectName + ": " + replica.summary) : replica.summary
+} else {
+  workItem.summary = replica.summary
}

-workItem.summary      = replica.summary
workItem.description  = replica.description
workItem.attachments  = attachmentHelper.mergeAttachments(workItem, replica)
workItem.comments     = commentHelper.mergeComments(workItem, replica)
workItem.labels       = replica.labels
workItem.priority     = replica.priority

/*
  Custom Fields (CF)
    To add incoming values to an Azure DevOps custom field, follow these steps:
    1/ Find the Field Name of the CF. You can also use the API name: https://docs.exalate.com/docs/how-to-sync-work-ite
    2/ Check how the value is coming over from the source side, by checking the "Entity Sync Status" of a ticket in syr
    3/ Add it all together like this:
    workItem.customFields."CF Name".value = replica.customFields."CF Name".value
*/

/*
  Status Synchronization
```

Here's how your requirements have been implemented:
On first sync, all incoming issues are mapped to the Azure DevOps project "ALLDEV".
On first sync, the Jira project name (in all capital letters) is prepended to the summary field.
Would you like to review or further adjust this configuration?

Describe what to send from this system. e.g., Exclude attachments...

Describe how to apply incoming data to this system. e.g., Map statuses...

Working with Scripts on Both Sides (Incoming and Outgoing)

The Outgoing and Incoming Aida chats work independently, so you can:

- Submit a prompt in the Outgoing script while Aida is generating code for the Incoming script
- Maintain separate conversation contexts for each script direction

Adjusting Your Workspace

You can resize the script panels and Aida chat windows to focus on what matters most:

- Drag the handle at the top of the Aida chat to increase or decrease its height
- Drag the divider between Outgoing and Incoming scripts to adjust their widths

Important Notes

- Aida is a helpful assistant, but you should always review the generated code before applying it
- Test your configuration using **Start Test Run** before publishing changes to production
- Aida works best when you provide clear, detailed descriptions of your sync requirements
- You can always manually edit the scripts after applying Aida's suggestions

ON THIS PAGE

Accessing Aida

Using Aida

Adjusting Your Workspace

Important Notes

Product

About Us

Release History

Glossary

API Reference

Security

Pricing and Licensing

Resources

Subscribe for a weekly Exalate hack

Academy

Blog

YouTube Channel

Ebooks

Still need help?

Join our Community

Visit our Service Desk

Find a Partner