

# Sync Rules

Last Modified on 03/11/2024 7:33 am EDT

Exalate uses synchronization rules (Sync Rules) to handle outgoing and incoming messages. You can find Sync Rules as a separate tab when you select the connection to edit.

**Note:** For script connections, Exalate is using the Monaco editor with Groovy support.

**Note:** Starting with Exalate v. 8.5.1 we are running Groovy v. 4.

Outgoing sync

Outgoing sync rules define what Information can be sent to the destination Instance. Check [the documentation](#) for more details.

```

1 replica.key = issue.key
2 replica.assignee = issue.assignee
3 replica.reporter = issue.reporter
4 replica.summary = issue.summary
5 replica.description = issue.description
6 replica.type = issue.type
7 replica.labels = issue.labels
8 replica.attachments = issue.attachments
9 replica.comments = issue.comments
10 replica.status = issue.status
11
12 replica.customFields."Customer" = issue.customFields."Customer"
13 replica.customFields."Supported Browsers" = issue.customFields."Supported Browsers"
14
15 //Send a Custom Field value
16 //replica.customFields."CF Name" = issue.customFields."CF Name"
    
```

Incoming sync

Incoming sync rules define how received data can be interpreted on the source side. Check [the documentation](#) for more details.

```

1 issue.labels = replica.labels
2 issue.summary = replica.summary
3 issue.description = replica.description ? "No description"
4 issue.attachments = attachmentHelper.mergeAttachments(issue, replica)
5 issue.comments += replica.addedComments
6
7 //Receive a Custom Field value
8 //issue.customFields."CF Name".value = replica.customFields."CF Name".value
9 /*
10 Status Synchronization
11
12 Sync status according to the mapping [remote issue status: local issue status]
13 If statuses are the same on both sides don't include them in the mapping
14 def statusMapping = ["Open":"New", "To Do":"Open"]
15 def remoteStatusName = replica.status.name
16 issue.setStatus(statusMapping[remoteStatusName] ? remoteStatusName)
17 */
    
```

## Editor Features

- **Light Mode:** Use to toggle to switch between Dark and Light mode
- **Copy:** Copy Incoming/Outgoing code blocks to the clipboard.
- **Expand:** Expands the code editor window for a better view if needed.
- **Mini Code Map:** Allows you to quickly navigate through the code, useful when dealing with longer code blocks.

Dark Mode

### Incoming sync

Incoming sync rules define how received data can be interpreted on the source side. Check [the documentation](#) for more details.

```
33 private static <I> scala.Option<I> none(Class<I> evidence) { scala.Option$.MODULE$.<I> empty() }
34
35 private static <L, R> scala.Tuple2<L, R> pair(L l, R r) { scala.Tuple2$.MODULE$.<L, R> apply(l, r) }
36
37 // SERVICES AND EXALATE API
38 private static play.api.inject.Injector getInjector() {
39     InjectorGetter.getInjector()
40 }
41
42 private static def getGeneralSettings() {
43     def gsp = InjectorGetter.getInjector().instanceOf(com.exalate.api.persistence.issue.tracker.IGenera
44     //def gsp = InjectorGetter.getInjector().instanceOf(com.exalate.api.persistence.issue.tracker.IGene
45     def gsOpt = await(gsp.get())
46     def gs = orNull(gsOpt)
47     as
```



Dark Mode on:

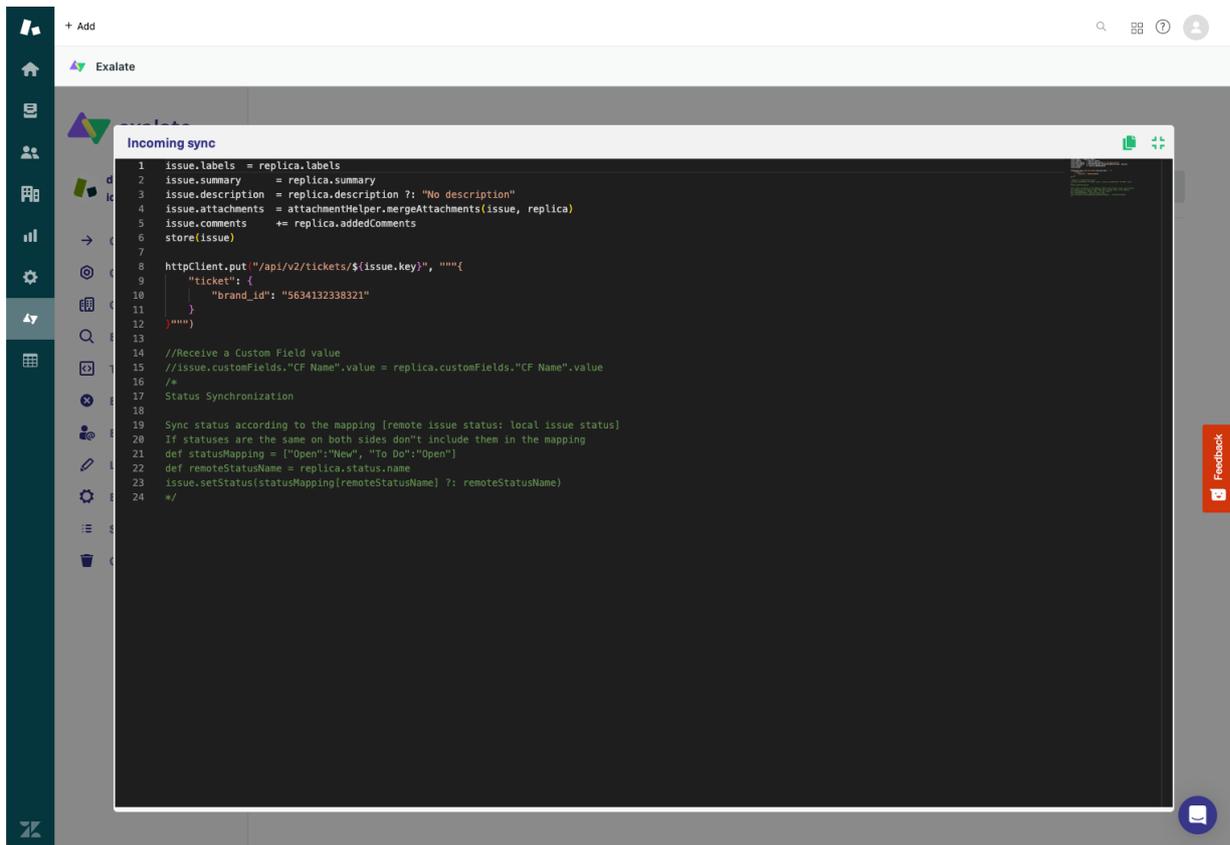
Dark Mode

### Outgoing sync

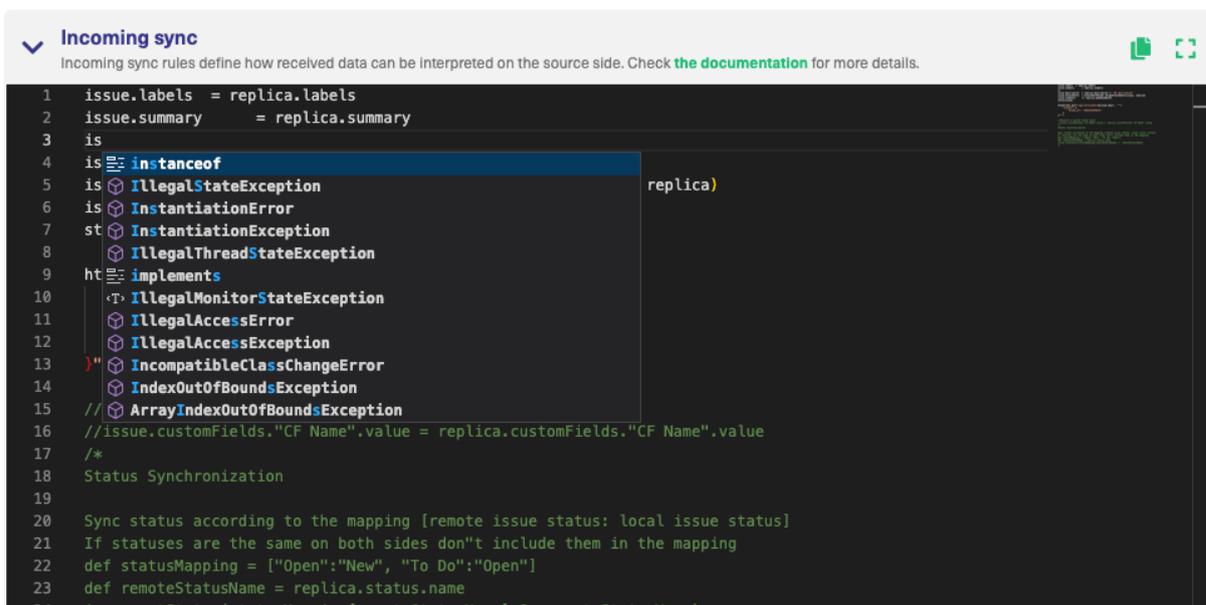
Outgoing sync rules define what information can be sent to the destination instance. Check [the documentation](#) for more details.

```
1 replica.key = issue.key
2 replica.assignee = issue.assignee
3 replica.reporter = issue.reporter
4 replica.summary = issue.summary
5 replica.description = issue.description
6 replica.type = issue.type
7 replica.labels = issue.labels
8 replica.attachments = issue.attachments
9 replica.comments = issue.comments
10 replica.status = issue.status
11
12 //Send a Custom Field value
13 //replica.customFields."CF Name" = issue.customFields."CF Name"
```

Expanded mode on (Dark mode):



In the Sync processors, the Monaco editor library allows for auto-complete, syntax highlight and syntax errors highlight when writing sync rules.



## How to create a Sync Rule

Sync rules are groovy-based scripts that can be used to implement [filtering](#), mapping, and [transformation](#). These are essential operations in any synchronization.

**Groovy** is a dynamic language for the Java platform. Check out the following links to get more details about Groovy and how to develop in this language:

- <http://www.groovy-lang.org/>
- <http://www.groovy-lang.org/learn.html>
- <http://www.groovy-lang.org/documentation.html>

Groovy learning courses that we can recommend:

- <https://www.pluralsight.com/courses/groovy-getting-started>
- <https://www.pluralsight.com/courses/groovy-fundamentals>

## Sync rule types

There are 2 different types of Sync Rules, each with its own purpose.

- Outgoing sync
- Incoming sync

**Note:** For more information please see [the synchronization process explanation](#).

## Outgoing sync

This rule defines what information is sent to the destination side.

Exalate runs the outgoing sync processor when you start the synchronization or update the local issue which is under sync.

You can assign issue fields to a replica on the outgoing sync. For more information on this, see [issue fields available for synchronization](#).

	Variable	Explanation
<b>Input</b>	issue	local issue data you need to synchronize
<b>Output</b>	replica	copy of the issue data which is sent to the destination instance

### Simple examples of Outgoing sync rules

```
1 replica.summary = issue.summary // send summary
2 replica.description = issue.description // send description
3 replica.comments = issue.comments // send comments
4 replica.attachments = issue.attachments // send attachments
```

### Condition example in the Outgoing Sync rules

Don't send anything when priority is trivial

```

1 // If the issue priority is "Trivial" don't send any data. In other cases send the summary, description, comments and
  attachments
2
3 if (issue.priority.name == "Trivial") {
4   return
5 }
6 replica.summary    = issue.summary
7 replica.description = issue.description
8 replica.comments   = issue.comments
9 replica.attachments = issue.attachments

```

## Incoming sync

When you receive data from the other side you need to apply this data on your instance. You can define how to handle the received information on your instance with the help of the incoming sync rules.

Exalate runs the incoming sync every time there's new data received from the remote side.

When you receive the synchronization data from the remote side for the first time, Exalate creates the issue locally in your instance.

Starting from this moment the issue is considered ***under synchronization (under sync)***. From that moment every issue update triggers the update of the synced issue on the other side.

	Variable	Explanation
<b>Input</b>	replica	Information received from the source instance
<b>Output</b>	issue	Issue object which is used to create/update the local issue
	previous	Previous information received from the source instance

**Important:** When you leave the incoming sync empty, nothing is synchronized. Please see the [Unidirectional synchronization](#) for more details.

### Simple Incoming sync example

```

1 if(firstSync){
2   // If it's the first sync for an issue and local copy of the issue does not exist yet
3   // Set project key from source issue, if not found set a default
4   issue.projectKey = nodeHelper.getProject(replica.project?.key)?.key ?: "TEST"
5   // Set type name from source issue, if not found set a default
6   issue.typeName   = nodeHelper.getIssueType(replica.type?.name, issue.projectKey)?.name ?: "Task"
7   issue.summary    = replica.summary
8   issue.description = replica.description
9   issue.comments   = commentHelper.mergeComments(issue, replica)
10  issue.attachments = attachmentHelper.mergeAttachments(issue, replica)

```

**Note:** Incoming requests are distinguished based on the information stored in a replica. So if

you want to have different sync rules for the first synchronization and then others for synced issue updates, you should use [conditional statements](#).

```
1 if (firstSync) {
2   return // don't create any issues, only sync changes to the issue which are already under sy3 nc}
4 issue.summary = replica.summary
5 issue.description = replica.description
6 issue.labels = replica.labels
7 issue.comments = commentHelper.mergeComments(issue, replica)
8 issue.comments = attachmentHelper.mergeAttachments(issue, replica)
```

### More advanced configuration

You can set your own values for the local issue based on the received data from the other side.

For example, if the synced issue status changes to Done on the remote side → set the local issue status to Resolved.

To configure advanced conditions for your synchronization use [script helper methods](#).

Check the example below:

```
1 // Create a request in the support project "SD", but if it's a critical issue in the customer's WEB project, assign it to Bob Price
2
3 issue.project = nodeHelper.getProject("SD")
4
5 if (replica.priority.name = "Critical" && replica.project.key = "WEB") {
6   issue.assignee = nodeHelper.getUser("bprice") // assign to Bob
7   issue.priority = nodeHelper.getPriority("Blocker")
8 } else {
9   issue.priority = nodeHelper.getPriority("Major")
10 }
11
12 issue.summary = replica.summary
13 issue.projectKey = "SD" issue.typeName = "Request"
```

### Default configuration

By default, Exalate configures some basic scripts in the [Sync Rules](#) for your convenience. Below is the default configuration of the Outgoing and Incoming sync.

It helps to synchronize basic issue data: summary, description, comments, resolution, status, attachments and project.

### Default Outgoing sync

```
1 replica.key      = issue.key
2 replica.type     = issue.type
3 replica.assignee = issue.assignee
4 replica.reporter = issue.reporter
5 replica.summary  = issue.summary
6 replica.description = issue.description
7 replica.labels   = issue.labels
8 replica.comments = issue.comments
9 replica.resolution = issue.resolution
10 replica.status  = issue.status
11 replica.parentId = issue.parentId
12 replica.priority = issue.priority
13 replica.attachments = issue.attachments
14 replica.project = issue.project
```

## Default Incoming sync

```
1 if(firstSync){
2   // If it's the first sync for an issue (local issue does not exist yet)
3   // Set project key from source issue, if not found set a default
4   issue.projectKey = nodeHelper.getProject(replica.project?.key)?.key ?: "TEST"
5   // Set type name from source issue, if not found set a default
6   issue.typeName   = nodeHelper.getIssueType(replica.type?.name, issue.projectKey)?.name ?: "Task"}
7 issue.summary     = replica.summary
8 issue.description = replica.description
9 issue.comments    = commentHelper.mergeComments(issue, replica)
10 issue.attachments = attachmentHelper.mergeAttachments(issue, replica)
```

### Product

#### ON THIS PAGE

[About Us](#)   
[Release History](#)   
[Outgoing sync](#)  
[Glossary](#) 

[Incoming sync](#) 

[Security](#) 

[Pricing and Licensing](#) 

#### Resources

[Academy](#) 

[Blog](#) 

[YouTube Channel](#) 

[Ebooks](#) 

#### Still need help?

[Join our Community](#) 

[Visit our Service Desk](#) 

[Find a Partner](#) 