














A connection between two instances specifies the synchronization behavior. Each connection includes information about two instances and describes how they are related to each other.

- The connection is pending, waiting for acceptance from the other side.
- A connection has been established and synchronization messages start to go out. You can now synchronize issues.
- Synchronization is paused, but changes are queued for a later update. Once you activate the connection, all changes are applied.

 azure_to_jira	0	 Deactivated	 
 jira_to_github	0	 Active	 
 jira-a_to_jira-b	0	 Pending	  

To know more about how Connections in Exalate work, please refer to our [Building a Connection](#) guide.

An **Instance** is a task management system that contains information you want to synchronize. Examples of instances are Jira Cloud, Salesforce, Zendesk, etc.

You can integrate with multiple instances using different connections. You can also synchronize between local projects within the same instance.

Note: Local synchronization is not supported in all task management systems.

Whenever you set up a connection between two different task management systems, one of them is a **local instance**, and the other one is a **remote instance**.

For example, if you set up integration between Jira Cloud and Zendesk, you can initiate a connection from either side.

If you choose to set up a connection on the Jira Cloud side, Jira Cloud becomes your local or source instance, and Zendesk the remote or destination instance.

To sync information between two task management systems, you need to set up a connection. When doing this, Exalate generates an invitation code. This works as a shared secret that helps authenticate both source and destination instances. Invitation codes store encrypted connection information, such as:

- Shared secret
- Connection type
- Connection name
- Connection initiator information
- Exalate app version
- Task management system and version
- Task management system URL
- Exalate Node URL
- Task management system UID - Unique instance identifier

An invitation code is used to set up a connection with the destination instance if an initiator does not have access to both sides of the connection. The code only applies to the instance you are inviting to synchronize.

Note: An invitation code is required only in Script mode configurations.

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

Rules Triggers Statistics Info

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 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 */

```

[More Integrations](#) [Documentation](#) [EULA](#) [Support](#) [Report a bug](#)

Powered by Exalate v. 5.6.0 (Core v. 5.6.0)

To learn more about how to create Sync Rules, please refer to the [Sync Rules guide](#).

The Incoming Sync Event is a record of the changes in the remote entity. Every Outgoing Sync event on the sending side results in an Incoming sync event on the Destination side.

The outgoing sync event is a record of the changes in the local entity. When the outgoing sync processor sends out information, it gets recorded as an Outgoing sync event.

A replica is a copy of the information that is getting transferred to the destination side. Exalate uses replicas to extract specific data and then send it over. You can use the replica in the Outgoing rules to specify which data should be sent. On the destination side, the replica object is used to represent the remote issue. It contains only the fields provided through the data filter on the source side.

You can view the replica details in the **Entity Sync status** panel. If the entity is synced, you can view the local replica by clicking the **Show local replica** button and the remote replica by clicking the **Show remote replica** button.

The replica looks something like this:

```
🏠 >> >> Local replica
{
  "version": {
    "major": 1,
    "minor": 14,
    "patch": 0
  },
  "hubIssue": {
    "components": [],
    "attachments": [],
    "voters": [],
    "customFields": {},
    "eventTriggerContext": {},
    "description": "Desc",
    "project": {
      "idStr": "10022",
      "key": "UD",
      "name": "UI Design",
      "versions": [],
      "components": []
    },
    "watchers": [],
    "fixVersions": [],
    "key": "UD-381",
    "summary": "SLA incident",
    "comments": [
      {
```

In this way, you can see what information is being passed over from the local instance. The remote replica has a similar structure.

Note: The hubIssue represents the information that is passed and how you can access it. Whereas, the replica is more than the hubIssue section. The replica is the entire payload to be transferred between platforms.

Note: For more information on this topic please read [Sync Rules](#).

ON THIS PAGE

