# Configuration FAQs

Last Modified on 04/22/2024 10:25 am EDT

## Is it possible to pause synchronization for maintenance purposes?

Yes, you can pause the synchronization. Synchronization events are queued while a Connection is deactivated. You can later resume the synchronization from the moment it was stopped.

## Can the configuration evolve independently?

Thanks to the distributed architecture, it is possible to adapt the common hub issue (or replica) to the local context. For instance, if your local workflow evolves, you will be able to change the Incoming Sync Rules to take into account the new configuration.

## How does the engine ensure that attachments are sent over correctly?

When an instance requests another instance to send over the content of an attachment, the other instance will respond with a hash key calculated from the content of the attachment. The same hash key is calculated on the receiving end to ensure that the received content matches the original attachment. The download will be retried once to avoid transmission errors. Failing this second attempt an error is raised and the administrator is notified.

## Can you upgrade your own environment without affecting any remote configurations?

Exalate is based on a distributed architecture where each application administrator configures what information can be sent and how incoming messages must be handled. All messages exchanged between the instances are based on a commonly defined 'hub issue' or replica which carries all the information one application administrator wants to communicate to the other side. The processors will use this information to apply the changes to the local issues.

## Is it possible to transition an issue whenever a certain comment is given?

Yes. With the help of groovy-based script rules, you can parse the newly added comments and trigger a resolve transition

```
# Resolve the issue when a comment contains the word 'Resolve'
#
def resolveComments = replica.addedComments.findAll { comment -> comment.body.contains("Resolve")}
if(resolveComments.size() > 0) {
  workflowHelper.transition(issue,"Resolve")
}
```

## How difficult is it to configure a complex scenario?

A scenario we encounter regularly is the Service Desk use case where tickets raised by a customer need to be raised internally in different projects - depending on the set of properties. One use case was particularly interesting. The target project depended on 3 different properties resulting in 200+ potential permutations. This mapping is stored in a single database table.

The way we solved it, involves finding the right projectkey in the database table and using it to raise the issue in the right project. Check the following example showing how this behavior can be configured using Exalate.

## Is it possible to synchronize contextual data of an issue (such as project, version, and user information)?

Yes. A hub issue contains all contextual data allowing you to implement complex business logic. Check out the detailed information a hub issue message transports such as:

- Project
- Version
- User

## How long does it take to process all synchronization events when 10000 issues in 100 different projects have been updated?

The synchronization engine is processing sync events sequentially. Processing 10000 issues in 100 different projects might take a couple of hours. We have been synchronizing 10k+ issues as part of our performance tests and we are looking at increasing the processing speed to meet the performance challenges.

## Can the configuration of the local platform change without impacting the configuration of the other platforms?

Yes. Each integration point can evolve as long as the hub issue model and Exalate API are respected.

## How long does it take to synchronize an issue, when 10 other issues under sync have been updated at the same time?

This happens immediately. Exalate is based on a fully event-driven engine. Whenever a user changes some information and that information needs to be sent, a sync event is generated containing a snapshot of the modified issue - ready to be sent to the remote instance. When 10 issues are changed, or 1 issue is changed10 times, at the same moment - 10 events are generated.

The replication layer in the Exalate architecture will transmit each sync event to the remote instance, triggering a sync request, which is processed at its own pace.

Handling a full synchronization event takes a couple of seconds (give/take the size of the information to be transmitted)

## How is the troubleshooting process supported?

Whenever an error occurs, a detailed overview is raised. It leads to the problem and we call it the stack trace. In case this is not sufficient, you can create a support.zip file and send it to support@exalate.com for further processing.

## Is Exalate compatible with any reverse-proxy solutions?

Since Exalate is an app for different deployments of Jira (Server, Cloud, Data Center) and other platforms, we do not attempt to ensure that our application is compatible with all third-party reverse-proxy solutions that might be set up on the Jira host.
However, we've tested it with nginx, and we can state that Exalate is compatible with nginx as a reverse-proxy server.

## How are the system administrators notified in case an error is raised?

The group of Exalate administrators will be notified in 2 different ways

- by email with the details of the error notification.
- In JIRA - by using 'In-JIRA' notifications - popping up a warning about the blocking error.

Also, errors are generated at 4 different levels.

- Issue or entity
    - As an example - when the proxy user is not allowed to modify the local issue due to issue security
- Relation
    - Typical example - when there is an error in the processor scripts, or when there is a permission problem to apply changes to an entity
- Instance
- A connection problem
- Node
- Bugs in the synchronization layer.

## What happens if I change the project key which is under sync?

Whenever a user changes something in an issue, Jira will notify Exalate about the change by generating an update event. If the issue is under an update gets synchronized. However, changing the key does not trigger such an event.

To synchronize a key change, the issue under sync needs to be touched by modifying something or by adding a comment in order to synchronize that information change.

Additional considerations when changing the project key

- Review the **Incoming sync** to make sure that the old project key is not hard coded.
- Review the triggers and update the JQL if this is relevant.

## Is it possible to sync multiple projects to one using one Connection?

Yes, it's possible to sync multiple projects with one Connection.

Basically, it's all about the configuration of the Sync Rules. Once you establish a connection between instances you can specify which data you would like to send to the other side and how to interpret incoming data.

The Outgoing sync rules of the sending side interact with the Incoming sync rules of the receiving side. For more details, please check the synchronization processors.

## The short name has a space, how to reference it in the script rules (Visual Mode)?

Assume that you created a connection with a shortname that has a space like 'Public Cloud'

```
def pcIssue = on["""Public Cloud"""]

pcIssue.customFields."Request Type".value = "Restart Server"
```

## What is the difference between issue.type and issue.type.name?

One of the most important details when configuring synchronization between multiple Jira Instances is the issue type field.

Jira **issue.type** is an **object** with a set of properties such as name, id, etc.

When syncing the issue type you can send the whole object or **issue.type.name** string **property** (issue.type object property).

Below you can see examples of the scripts that help to sync issue types in different ways.

Sync Rules include issue type sync by default. You can adapt the script to your needs.

**Source Side**

**Outgoing sync**

- Send the **issue.type** object(This script is included in the default sync rules).

```
replica.type = issue.type
```

- Send **issue.typeName** string property

```
replica.typeName = issue.type.name
```

**Destination Side**

**Incoming sync**

- Set an *issue.typeName* to *Task* for all incoming issues

```
issue.typeName = replica.type?.name ?: "Task"
```

- Set an *issue.type* for incoming issues as received from the source side

```
issue.typeName = replica.typeName
```

## How to retrieve all the bindings presented to a processor?

Add the following code at the beginning of the processor

```
def bindings = context.getBindings(100)
def result = ""
bindings.each { result += "<b>$it.key</b><br>$it.value?.properties<br><hr><p>" }
debug.error("Bindings = ${result}")
```

This will generate an error for you to see the results.

## Is it possible to migrate from Visual Mode to Script Mode?

No, it's not possible to migrate from Visual to Script Mode.

Only Connections in the Basic Mode can be upgraded to Script or Visual.