How to Sync Issue User Fields in Jira On-premise

Last Modified on 04/09/2024 6:57 am EDT

This article describes how to sync Jira issue fields, that include users.

Jira Issue Fields that include a User Object

There are several issue fields, which include user information, you can sync all these fields using the Exalate app.

- issue creator
- issue assignee
- issue reporter
- comment author
- custom field of type user

Despite the fact that multiple Jira instances have different users set, you can sync users information with the help of the Exalate nodeHelper methods. To handle user sync you can use following helpers:

Jira on-premise	Jira Cloud
• getUser • getUserByUsername • getUserByEmail • getUserByFullName	• getUser • getUserByEmail ⚠️ (only if the user email is not hidden in the Atlassian account settings) • getUserByFullName ⚠️ (only if the user email is not hidden in the Atlassian account settings)

Check some examples of the user fields sync below.

Source Side

Add the code below to the **Outgoing sync** to send fields like issue creator, issue assignee, issue reporter

```
replica.assignee  = issue.assignee
replica.reporter  = issue.reporter
replica.creator   = issue.creator
```

Destination Side

Depending on whether the destination side is Jira Cloud or Jira on-premise you need to use different nodeHelper methods. Add the code below to the **Incoming sync** to add received user fields data from the source side.

issue creator

Jira on-premise to Jira on-premise case

```
issue.creator = nodeHelper.getUserByUsername(replica.creator?.username)
```

Jira on-premise to Jira Cloud case

```
issue.creator = nodeHelper.getUserByEmail(replica.creator?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
    "admin@admin.com" : "557358:bda57a72g56a9-4219-9c29-7d666481388f",
]
 issue.creator = nodeHelper.getUser(userMapping[replica.creator?.email])
```

Jira Cloud to Jira on-premise case

```
issue.creator = nodeHelper.getUserByEmail(replica.creator?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
    "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]
 issue.creator = nodeHelper.getUserByEmail(userMapping[replica.creator?.key])
```

Jira Cloud to Jira Cloud case

```
issue.creator = nodeHelper.getUser(replica.creator?.key)
```

issue assignee

Jira on-premise to Jira on-premise case

```
issue.assignee = nodeHelper.getUserByUsername(replica.assignee?.username)
```

Jira on-premise to Jira Cloud case

```
issue.assignee = nodeHelper.getUserByEmail(replica.assignee?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
    "admin@admin.com" : "557358:bda57a72g56a9-4219-9c29-7d666481388f",
]
 issue.assignee = nodeHelper.getUser(userMapping[replica.assignee?.email])
```

Jira Cloud to Jira on-premise case

```
issue.assignee = nodeHelper.getUserByEmail(replica.assignee?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]
 issue.assignee = nodeHelper.getUserByEmail(userMapping[replica.assignee?.key])
```

Jira Cloud to Jira Cloud case

```
issue.assignee = nodeHelper.getUser(replica.assignee?.key)
```

issue reporter

Jira on-premise to Jira on-premise case

```
issue.reporter = nodeHelper.getUserByUsername(replica.reporter?.username)
```

Jira on-premise to Jira Cloud case

```
 issue.reporter = nodeHelper.getUserByEmail(replica.reporter?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "admin@admin.com" : "557358:bda57a72g56a9-4219-9c29-7d666481388f",
]
 issue.reporter = nodeHelper.getUser(userMapping[replica.reporter?.email])
```

Jira Cloud to Jira on-premise case

```
 issue.reporter = nodeHelper.getUserByEmail(replica.reporter?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]
 issue.reporter = nodeHelper.getUserByEmail(userMapping[replica.reporter?.key])
```

Jira Cloud to Jira Cloud case

```
issue.reporter = nodeHelper.getUser(replica.reporter?.key)
```

comment author

Jira on-premise to Jira on-premise case

```
issue.comments = commentHelper.mergeComments(
  issue,
  replica,
  {
    it.executor = nodeHelper.getUserByUsername(it.author?.username);it
  }
)
```

Jira on-premise to Jira Cloud case

```
issue.comments = commentHelper.mergeComments(
  issue,
  replica,
  {
    it.executor = nodeHelper.getUserByEmail(it.author?.email);it
  }
)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
      "admin@admin.com" : "557358:bda57a72g56a9-4219-9c29-7d666481388f",
]


issue.comments = commentHelper.mergeComments(
  issue,
  replica,
  {
    it.executor = nodeHelper.getUser(userMapping[it.author?.email]);it
  }
)
```

Jira Cloud to Jira on-premise case

```
issue.comments = commentHelper.mergeComments(
  issue,
  replica,
  {
    it.executor = nodeHelper.getUserByEmail(it.author?.email);it
  }
)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
      "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]


issue.comments = commentHelper.mergeComments(
  issue,
  replica,
  {
    it.executor = nodeHelper.getUserByEmail(userMapping[it.author?.key]);it
  }
)
```

Jira Cloud to Jira Cloud case

```
issue.comments = commentHelper.mergeComments(
  issue,
  replica,
  {
    it.executor = nodeHelper.getUser(it.author?.key);it
  }
)
```

custom field of type user

Jira on-premise to Jira on-premise case

```
issue.customFields."Cool User"?.value =
 nodeHelper.getUserByUsername(replica.customFields."Cool User"?.value?.username)
```

Jira on-premise to Jira Cloud case

```
issue.customFields."Cool User"?.value =
 nodeHelper.getUserByEmail(replica.customFields."Cool User"?.value?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "admin@admin.com" : "557358:bda57a72g56a9-4219-9c29-7d666481388f",
]
issue.customFields."Cool User"?.value =
 nodeHelper.getUser(userMapping[replica.customFields."Cool User"?.value?.email])
```

Jira Cloud to Jira on-premise case

```
issue.customFields."Cool User"?.value =
 nodeHelper.getUserByEmail(replica.customFields."Cool User"?.value?.email)
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]
issue.customFields."Cool User"?.value =
 nodeHelper.getUserByEmail(userMapping[replica.customFields."Cool User"?.value?.key])
```

Jira Cloud to Jira Cloud case

```
issue.customFields."Cool User"?.value = nodeHelper.getUser(replica.customFields."Cool User"?.value?.key)
```

Custom Handling

You can define your own logic for the synchronization of any user field using the helper methods and the power of Groovy scripting. Check the examples below:

- You can get a user on your side using remote user data such as username, email, accountId or set a default user.

  The example below shows how you can try getting a user by remote username or email or set a default user if not found.

  ```
  def localReporter = nodeHelper.getUserByUsername(replica.reporter?.username)
  if(!localReporter){
    localReporter = nodeHelper.getUserByEmail(replica.reporter?.email)
  }
  if(!localReporter){
    localReporter = nodeHelper.getUserByEmail("default user email")
  }
  issue.reporter   = localReporter
  ```

Get a user by mapping remote account Id to local email or set a default user if not found (if the source side is Jira Cloud)

```
final def userMapping = [
     "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]
def localReporter = nodeHelper.getUserByEmail(userMapping[replica.reporter?.key])
if(!localReporter){
  localReporter = nodeHelper.getUserByEmail("default user email")
}
issue.reporter   = localReporter
```

- User object has "active" property in Jira, so you can also check whether the user is active or not and use this criteria in the sync rules.

  The example below shows how to determine whether the **assignee user exists** in the system and his account is **active**, in other cases set the default user **admin**.

  Jira on-premise to Jira on-premise case

```
issue.assignee = nodeHelper.getUserByUsername(replica.assignee?.username)?.active ?
    nodeHelper.getUserByUsername(replica.assignee?.username) :
     nodeHelper.getUserByUsername("admin")
```

Jira on-premise to Jira Cloud case

```
issue.assignee = nodeHelper.getUserByEmail(replica.assignee?.email)?.active ?
    nodeHelper.getUserByEmail(replica.assignee?.email) :
     nodeHelper.getUserByEmail("default user email")
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "admin@admin.com" : "557358:bda57a72g56a9-4219-9c29-7d666481388f",
]
issue.assignee = nodeHelper.getUser(userMapping[replica.assignee?.email])?.active ?
    nodeHelper.getUser(userMapping[replica.assignee?.email]) :
     nodeHelper.getUser("default user account ID")
```

Jira Cloud to Jira on-premise case

```
issue.assignee = nodeHelper.getUserByEmail(replica.assignee?.email)?.active ?
    nodeHelper.getUserByEmail(replica.assignee?.email) :
     nodeHelper.getUserByUsername("admin")
```

or (if a Cloud user's email is hidden for Exalate)

```
final def userMapping = [
     "557358:bda57a72g56a9-4219-9c29-7d666481388f" : "admin@admin.com"
]
issue.assignee = nodeHelper.getUserByEmail(userMapping[replica.assignee?.key])?.active ?
    nodeHelper.getUserByEmail(userMapping[replica.assignee?.key]) :
     nodeHelper.getUserByUsername("admin")
```

Jira Cloud to Jira Cloud case

```
issue.assignee = nodeHelper.getUser(replica.assignee?.key)?.active ?
    nodeHelper.getUser(replica.assignee?.key) :
     nodeHelper.getUser("default user account ID")
```

ON THIS PAGE