# Jira Cloud Jira OnPrem: Syncing User Mentions across the systems

When you are synchronizing two different systems, there is always a challenge of both systems having its own nuances that the integration needs to deal with. Considering this particular example of Jira OnPrem and Jira Cloud synchronization, we would be focusing on how user mentions can be correctly synced to the remote side.

To understand the challenge involved here, we need to understand how both systems handle mentions internally i.e.

- Jira Cloud uses the format [~accountid:account-id-string] in order to represent mentions. You can read more on the subject on the Atlassian community.
- Jira Server uses the format [~username] to represent mentions internally. Some more information on the topic here.

## Jira Cloud to Jira OnPrem

Let us now take a look at how we handle this situation when going from Jira Cloud to Jira OnPrem.

### Jira Cloud Outgoing Script:

```
replica.comments = issue.comments.collect {
    comment ->
    def matcher  = comment.body =~ /\[~accountid:([\w:-]+)\]/
    def newCommentBody = comment.body
    matcher.each {
        target = nodeHelper.getUser(it[1])?.email ?: "Stranger"
        newCommentBody = newCommentBody.replace(it[0],target)
    }
    comment.body = newCommentBody
    comment
}
```

In the above snippet, we use a regular expression to find comments with user mentions in them. For each such comment, we replace the actual mention with the email address of the user being mentioned here. This is achieved by employing the nodeHelper.getUser() method. Now the comment added to the replica does not contain the actual mention, but rather the email address of the user mentioned.

### Jira OnPrem Incoming Script:

```
for(comment in replica.addedComments){
    def newCommentBody=comment.body
    def matcher  = comment.body =~ /[a-zA-Z0-9+._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+/
     matcher.each {
        x->
        if (nodeHelper.getUserByEmail(x)){
            def target = "[~" + nodeHelper.getUserByEmail(x).username + "]"
            newCommentBody = newCommentBody.replace(x,target)
        }
        else
                newCommentBody = newCommentBody.replace(x,"[External user with email ${x} mentioned]")
      }
    comment.body=  newCommentBody
}
issue.comments     = commentHelper.mergeComments(issue, replica)
```

Once the replica arrives on the OnPrem side, we again deploy a regular expression to find the presence of an email address within the comment body. If found, we try to find the user account corresponding to this email address using the nodeHelper.getUserByEmail() method and then replace the email address with the username of the user (taking care of the OnPrem format for the mention). If the user is not found on the OnPrem side, we simply replace the email address with a custom message informing the end-user that the user mentioned here (on the source side) does not have an account on this instance.

## Jira OnPrem to Jira Cloud

When we consider the other direction i.e. Jira OnPrem to Jira Cloud, the logic essentially remains the same but the format of what we are searching for changes.

## Jira OnPrem Outgoing Script:

```
replica.comments = issue.comments.collect {
    comment ->
    if (comment.roleLevel != "Developers"){
        def matcher  = comment.body =~ /\[~(\w+)\]/
        def newCommentBody = comment.body
        matcher.each {
            target = nodeHelper.getUserByUsername(it[1])?.email ?: "Stranger"
            newCommentBody = newCommentBody.replace(it[0],target)
        }
        comment.body = newCommentBody
        comment
    }
}
```

In the above snippet, we use a regular expression to find comments with user mentions in them. For each such comment, we replace the actual mention with the email address of the user being mentioned here. This is achieved by employing the nodeHelper.getUserByUsername() method. Now the comment added to the replica does not contain the actual mention, but rather the email address of the user mentioned.

## Jira Cloud Incoming Script:

```
for(comment in replica.addedComments){
    def newCommentBody=comment.body
    def matcher  = comment.body =~ /[a-zA-Z0-9+._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+/
     matcher.each {
        x->
        if (nodeHelper.getUserByEmail(x)){
            def target = "[~accountid:" + nodeHelper.getUserByEmail(x).key + "]"
            newCommentBody = newCommentBody.replace(x,target)
        }
        else
                newCommentBody = newCommentBody.replace(x,"[External user with email ${x} mentioned]")
        }
    comment.body=  newCommentBody
}
issue.comments     = commentHelper.mergeComments(issue, replica)
```

Once the replica arrives on the Cloud side, we again deploy a regular expression to find the presence of an email address within the comment body. If found, we try to find the user account corresponding to this email address using the nodeHelper.getUserByEmail() method and then replace the email address with the username of the user (taking care of the Cloud format for the mention). If the user is not found on the Cloud side, we simply replace the email address with a custom message informing the end-user that the user mentioned here (on the source side) does not have an account on this instance.

Here is a video demonstrating the use case in action: