Comment Visibility - an Advanced Issue Sync Case

Last Modified on 01/28/2026 11:59 am EST

Introduction

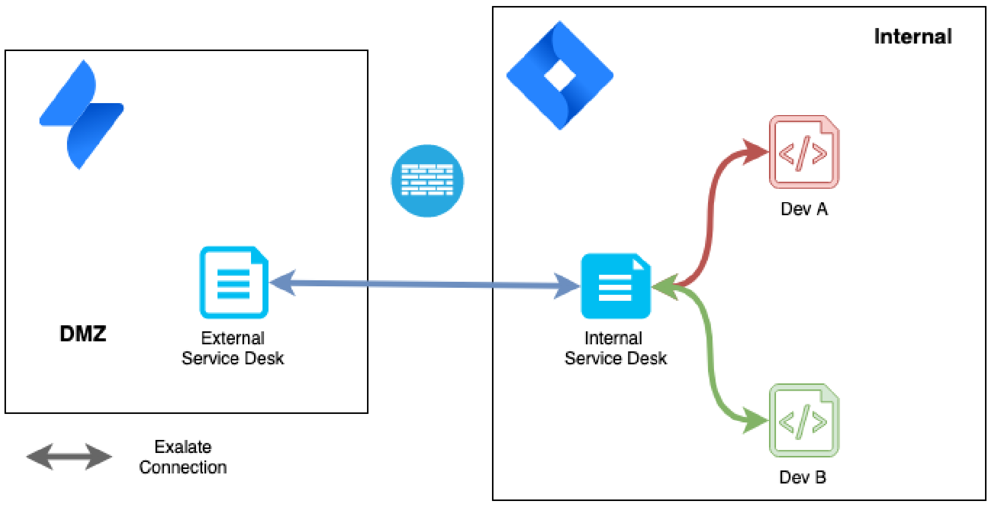
Warning: Despite our best efforts, code can change without notice due to a variety of factors. If you encounter an issue in any of the code shown here and find that a specific block of code is not correct, or is causing errors, please check with the Community to find an updated version.

One of our customers is using Exalate for an advanced use case, where comments need to be handled differently based on the context in which they are created.

Challenge

There are 2 instances and 4 projects at play

- External Instance (label JSD)
  with one service management project
- Internal Instance (Label INT) with 3 projects
  Internal service management and 2 dev projects: dev project A and dev project B



Requirements

The comment-related requirements are:

- Comments on the external service management need to go to the internal service management and ripple through to the dev projects
- Comments on the dev project(s) should only go to the internal service management project

but not to the external service management
- Comments from the dev project on the internal service management must be restricted, while the comments from the internal service management on the dev project must be open
- If a comment is made on dev project A, then it should not ripple through to dev project B or the original ticket (on the external service management)

Understanding the Challenge

There are 3 types of comments:

- Ticket comments made by customers on the external service management
- Service management comments made by support engineers on the internal service management
- Dev comments made by developers on the dev project

This table depicts the required behavior whenever a comment is made on one project, how it needs to be synced to the twin issue
(The column is the source)

Comment Sync Behavior Map (From/To)	**External Service management**	**Internal Service management**	**Dev Project A**	**Dev Project B**
External Service management	Restricted	Open	Open	
Internal Service management	Open	Open	Open	
Dev Project A	Not allowed	Restricted		Not Allowed
Dev Project B	Not Allowed	Restricted	Not Allowed	

Filtering out the Comments which should not be Exchanged

To meet the requirements

- Comments on the external service management need to go to the internal service management and ripple through to the dev projects
- If a comment is made on dev project A, then it should not ripple through to dev project B or the original ticket
- Comments on the dev project should only go to the internal service management project but not to the external service management

The approach we choose was to have 3 different synchronization users. These are functional accounts that define the connection which created the comment

- SyncJSD (which is the proxy user)
- SyncCon1
- SyncCon2

The incoming sync processor of the internal connections has the following logic to impersonate the comment

The **Incoming sync processor** of the connections contains the following statement:

```
// set the author of the twin comment to SyncCon1

issue.comments     = commentHelper.mergeComments(issue, replica, {
       comment ->
                          comment.executor = nodeHelper.getUserByUsername("SyncCon1")
                          comment
  })
```

Note: In the second connection, you would use **SyncCon2.**

And in the **outgoing sync processor,** you exclude the comments which are exchanged over the other connection as follows

```
replica.comments = issue.comments.findAll { comment ->
                      comment.author.username == "SyncCon1" ||
      !comment.author.username.contains("SyncCon")
                  }
```

Note: Change the name to SyncCon2 for the second connection.

Why Would this Work?

Whenever a comment is made on the internal service management project, it is either created by a user, the SyncJSD user (when it comes from the external connection), the SyncCon1 or the SyncCon2 user (when it comes from the dev projects). The statement **!comment.author.username.contains("SyncCon")** filters out all comments that are created in the other dev project, but all other comments pass through.

Whenever a comment is created by the SyncJSD user (ie the customer comment), the comment.author.username is 'SyncJSD', and included in the sync message from the internal service management to both development projects, as it passes the condition.

Ok - the First Hurdle Met - What's Next?

- Comments from the dev project on the internal service management must be restricted, while the comments from the internal service management on the dev project must be open

The way to restrict the comment is explained in 'How to manage comment visibility'. By setting the comment.role level (check the comment object to the appropriate role, comments are restricted. The challenge here is to apply this change only to the internal service management project and not to the dev project.

The additional complexity is that there are multiple internal service management projects (kept this for last). We decided to use the category to indicate the nature of the project. You could also use other ways of finding out what the type of project it is, but this approach allows us to demonstrate how one can work around a gap in the exalate product. Currently, there is no category in the project object. As we don't have the time for the exalate team to update the product, we can apply the following workaround

```
// fetch the project  category using the Jira API
import com.atlassian.jira.component.ComponentAccessor
def projectManager = ComponentAccessor.projectManager
def jProject = projectManager.getProjectByCurrentKey(issue.projectKey)
def jCategory = jProject.projectCategoryObject

// If the category is 'Service management', set the role level to 'Team', else null


issue.comments     = commentHelper.mergeComments(issue, replica, {
    comment ->
    comment.roleLevel = (jCategory.name == "Service management") ? "Team" : null
                        comment.executor = nodeHelper.getUserByUsername("SyncCon1")
                        comment
 })
```

Conclusion

You can create advanced synchronization use cases by combining the flexibility of Exalate with the capabilities of Jira.

ON THIS PAGE