Post Exalate Processor

Last Modified on 11/19/2022 3:33 am EST

Disclaimer

- This is an experimental feature and there is no guarantee of its stability, as it didn't run through a complete validation cycle
- The feature can evolve and even be retracted in case it doesn't meet our quality criteria
- We love to get feedback on experimental features which can be sent to support@exalate.com

The post-exalate processor executes a change on the original issue once the synchronization has been finished. It's an advanced option of the Sync Rules configuration.

property name	type	description
issueKey	IssueKey	A key of an issue that was synchronized
remoteIssueKey	IssueKey	A Key of an issue, created on the other side
jiraIssue	MutableIssue	An issue that was synchronized
isParentIssue	Boolean	True if the issue was initiating the sync
workflowUtil	WorkflowUtil	Helper method for JIRA issue transitions

property name	type	description
jiraProxyUser	ApplicationUser	The proxy user configured in General Settings

You can use the Post exalate processor to automate some actions, which should be done once the issue has been synchronized. For example:

- Sync sub-tasks after parent issue sync
- Sync epic stories, after epic sync
- Update a custom field with the remote issue key
- Transition Exalated issue
- Delete an issue if the twin has been deleted

Sync Sub-tasks after Parent Issue Sync

```
import com.exalate.api.replication.out.IEventSchedulerService
import com.atlassian.jira.component.ComponentAccessor
import com.exalate.basic.domain.BasicIssueKey
import com.exalate.api.domain.trigger.ExalateJiraEventType

if(isParentIssue && eventType == ExalateJiraEventType.EXALATED){
  final def eventScheduler = ComponentAccessor.getOSGiComponentInstanceOfType(IEventSchedulerService.class)
  def stm = ComponentAccessor.getSubTaskManager()
  def subtasks = stm.getSubTaskObjects(jiraIssue)
  subtasks.each{
    def subtaskKey = new BasicIssueKey(it.id, it.key)
    eventScheduler.schedulePairEvent(subtaskKey, connection)
  }
}
```

Sync epic Stories, after epic Sync

```
import com.exalate.api.replication.out.IEventSchedulerService
import com.atlassian.jira.component.ComponentAccessor
import com.exalate.basic.domain.BasicIssueKey
import com.exalate.api.domain.trigger.ExalateJiraEventType
def getStories = { BasicIssueKey epicExIssueKey ->
       def esClass = ComponentAccessor.getPluginAccessor().getClassLoader().findClass("com.atlassian.greenhopper.a
pi.issuelink.ManagedIssueLinkTypesService")
       def es = ComponentAccessor.getOSGiComponentInstanceOfType(esClass)
       if (es == null) {
          throw relationLevelErrorInternal("Unable to load the epic link manager, even though the Jira Agile plugin is fou
nd. The script is wrong and should be adapted")
       }
       def iltResult = es.getEpicLinkIssueLinkType()
       if (!iltResult.isValid()) {
          def errorCollection = iltResult.errorCollection
          throw relationLevelErrorInternal(
                "Unable to get the story-epic issue link: " +
                      "\n\tErrors: " + errorCollection?.errors +
                      "\n\tReasons: " + errorCollection?.reasons
          )
       }
       def epicLinkType = iltResult.get()
       if (epicLinkType == null) {
          throw relationLevelErrorInternal(
   "Epic Link Type is null. Something is terribly wrong with either your Jira Agile installation or your script."
   )
       }
       def stories = ComponentAccessor.getIssueLinkManager().getOutwardLinks(epicExIssueKey.id)
             .findAll { epicLinkType.id.equals(it.linkTypeId) }
             .collect { it.destinationObject }
       return stories.collect { ["id" : it.id, "key": it.key]}
    }
if(isParentIssue && eventType == ExalateJiraEventType.EXALATED){
  final def eventScheduler = ComponentAccessor.getOSGiComponentInstanceOfType(IEventSchedulerService.class)
  def stories = getStories(issueKey)
  stories.each{
    eventScheduler.schedulePairEvent(new BasicIssueKey(it.id, it.key), connection)
  }
}
```

Update a Custom Field with the Remote Issue Key

```
import com.atlassian.jira.component.ComponentAccessor
import  com.atlassian.jira.event.type.EventDispatchOption
import com.exalate.api.domain.trigger.ExalateJiraEventType
if(isParentIssue && eventType == ExalateJiraEventType.EXALATED){
  def cf = ComponentAccessor.getCustomFieldManager().getCustomFieldObjectByName("Remote Issue Key")
  jiraIssue.setCustomFieldValue(cf, remoteIssueKey.getURN())
  ComponentAccessor.getIssueManager().updateIssue(jiraProxyUser, jiraIssue, EventDispatchOption.ISSUE_UPDATED,
false)
}
```

Transition Exalated Issue

```
import com.exalate.api.domain.trigger.ExalateJiraEventType

if(eventType == ExalateJiraEventType.EXALATED)
 workflowUtil.doTransition(jiraProxyUser, jiraIssue, "Start Progress")
```

Delete an Issue if the Twin has been Deleted

```
import com.atlassian.jira.component.ComponentAccessor
import com.exalate.api.domain.trigger.ExalateJiraEventType


def issueService = ComponentAccessor.getIssueService()


if(isParentIssue && eventType == ExalateJiraEventType.DELETED){
  def validation = issueService.validateDelete(jiraProxyUser, jiraIssue.id)

  log.info("validation of the delete is ${validation.isValid()} and error collection is ${validation.getErrorCollection()}")
  if (validation.isValid()) {
    log.info("Going to delete issue with key ${jiraIssue.key}")
    issueService.delete(jiraProxyUser, validation)
  }
}
```

ON THIS PAGE