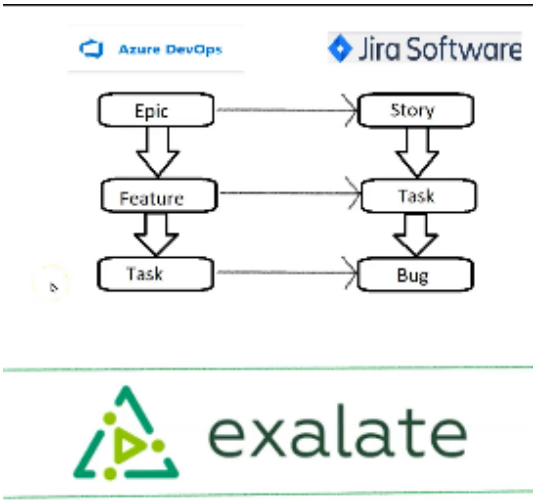
Azure Devops -> Jira On-Prem: Multi-level Issue Hierarchy

The main requirement for this use case is to maintain the multi-level issue hierarchy while Exalating issues from Azure DevOps to Jira Server/DC.

The relationship on Azure DevOps is Epic Feature Task, and this should be mirrored as Story Task Bug on Jira Server. The following depicts what we are trying to achieve here:



In the Outgoing script of Azure DevOps, we need to ensure that the relationships between the different workItems is sent across to Jira. The following code snippet achieves this:

ADO Outgoing

```
replica.parentId = workItem.parentId

def res = httpClient.get("/_apis/wit/workitems/${workItem.key}?\$expand=relations&api-version=6.0",false)
if (res.relations != null){
    replica."relation" = res.relations[0].attributes.name
    replica."relationid" = (res.relations[0].url).tokenize('/')[7]
    }
```

The rest of the magic happens in the Jira Incoming script. The full code is as follows:

Jira Incoming

```
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.security.JiraAuthenticationContext
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.issue.link.IssueLinkType
import org.slf4j.Logger

class LogIn {

        static logIn(u) {
            def authCtx = com.atlassian.jira.component.ComponentAccessor.getJiraAuthenticationContext()
            try {
                //Jira 7
                authCtx.setLoggedInUser(u)
            } catch (Exception ignore) {
                // Jira 6
                //noinspection GroovyAssignabilityCheck
                authCtx.setLoggedInUser(u.getDirectoryUser())
            }
       }
```

```
        static <R> R tryLogInFinallyLogOut(Closure<R> fn) {
            def authCtx = com.atlassian.jira.component.ComponentAccessor.getJiraAuthenticationContext()
            def proxyAppUser = getProxyUser()
            def loggedInUser = authCtx.getLoggedInUser()
            try {
                logIn(proxyAppUser)
                fn()
            } finally {
                logIn(loggedInUser)
            }
        }

        static getProxyUser() {
            def nserv = com.atlassian.jira.component.ComponentAccessor.getOSGiComponentInstanceOfType(com.
exalate.api.node.INodeService.class)
            nserv.proxyUser
        }

    }
class CreateIssue {

        static def log = org.slf4j.LoggerFactory.getLogger("com.exalate.scripts.Epic")

        private static def doCreate = {
            com.exalate.basic.domain.hubobject.v1.BasicHubIssue replica,
            com.exalate.basic.domain.hubobject.v1.BasicHubIssue issue,
            com.exalate.api.domain.request.ISyncRequest syncRequest,
            com.exalate.node.hubobject.v1_3.NodeHelper nodeHelper,
            com.exalate.basic.domain.hubobject.v1.BasicHubIssue issueBeforeScript,
            com.exalate.api.domain.INonPersistentReplica remoteReplica,
            List<com.exalate.api.domain.twintrace.INonPersistentTrace> traces,
            List<com.exalate.api.domain.IBlobMetadata> blobMetadataList,
            Logger log ->
                def firstSync = com.exalate.processor.jira.JiraCreateIssueProcessor.createProcessorContext.get()
== true
                def issueLevelError = { String msg ->
                    new com.exalate.api.exception.IssueTrackerException(msg)
                }
                def issueLevelError2 = { String msg, Throwable c ->
                    new com.exalate.api.exception.IssueTrackerException(msg, c)
                }
                def toExIssueKey = { com.atlassian.jira.issue.MutableIssue i ->
                    new com.exalate.basic.domain.BasicIssueKey(i.id, i.key)
                }
                final def authCtxInternal = com.atlassian.jira.component.ComponentAccessor.
getJiraAuthenticationContext()
                final def imInternal = com.atlassian.jira.component.ComponentAccessor.issueManager
                final def umInternal = com.atlassian.jira.component.ComponentAccessor.userManager
                final def nservInternal = com.atlassian.jira.component.ComponentAccessor.
getOSGiComponentInstanceOfType(com.exalate.api.node.INodeService.class)

                final def hohfInternal2 = com.atlassian.jira.component.ComponentAccessor.
getOSGiComponentInstanceOfType(com.exalate.api.hubobject.IHubObjectHelperFactory.class)
                //noinspection GroovyAssignabilityCheck
                final def hohInternal2 = hohfInternal2.get(remoteReplica.payload.version)

                if (issue.id != null) {
                    def existingIssue = imInternal.getIssueObject(issue.id as Long)
                    if (existingIssue != null) {
                        return [existingIssue, toExIssueKey(existingIssue)]
                    }
                }

                def proxyAppUserInternal = nservInternal.getProxyUser()

                def loggedInUser = authCtxInternal.getLoggedInUser()

                log.debug("Logged user is " + loggedInUser)

                def reporterAppUser = null
```

```
                if (issue.reporter != null) {
                    reporterAppUser = umInternal.getUserByKey(issue.reporter?.key)
                }
                reporterAppUser = reporterAppUser ?: proxyAppUserInternal

                issue.project = issue.project ?: ({ nodeHelper.getProject(issue.projectKey) })()
                issue.type = issue.type ?: ({ nodeHelper.getIssueType(issue.typeName) })()

                def jIssueInternal = null
                try {
                    LogIn.logIn(reporterAppUser)

                    if (issue.id != null) {
                        def existingIssue = imInternal.getIssueObject(issue.id as Long)
                        if (existingIssue != null) {
                            issue.id = existingIssue.id
                            issue.key = existingIssue.key
                            return [existingIssue, toExIssueKey(existingIssue)]
                        }
                    }
                    def cir
                    try{
                        cir = hohInternal2.createNodeIssueWith(issue, hohInternal2.createHubIssueTemplate(),
null, [:], blobMetadataList, syncRequest)
                    } catch (MissingMethodException e){
                        cir = hohInternal2.createNodeIssueWith(issue, hohInternal2.createHubIssueTemplate(),
null, [:], blobMetadataList, syncRequest.getConnection())
                    }

                    def createdIssueKey = cir.getIssueKey();

                    jIssueInternal = imInternal.getIssueObject(createdIssueKey.id)

                    if (issue.id != null) {
                        def oldIssueKey = jIssueInternal.key
                        def oldIssueId = jIssueInternal.id
                        try {
                            jIssueInternal.key = issue.key
                            jIssueInternal.store()
                        } catch (Exception e) {
                            log.error("""Failed to sync issue key: ${e.message}. Please contact Exalate
Support. Deleting issue `$oldIssueKey` ($oldIssueId)""".toString(), e)
                            imInternal.deleteIssue(proxyAppUserInternal, jIssueInternal as com.atlassian.jira.
issue.Issue, com.atlassian.jira.event.type.EventDispatchOption.ISSUE_DELETED, false)
                        }
                    }

                    issue.id = jIssueInternal.id
                    issue.key = jIssueInternal.key

                    return [jIssueInternal, toExIssueKey(jIssueInternal)]
                } catch (com.exalate.api.exception.IssueTrackerException ite) {
                    if (firstSync && jIssueInternal != null) {
                        imInternal.deleteIssue(proxyAppUserInternal, jIssueInternal as com.atlassian.jira.issue.
Issue, com.atlassian.jira.event.type.EventDispatchOption.ISSUE_DELETED, false)
                    }
                    throw ite
                } catch (Exception e) {
                    if (firstSync && jIssueInternal != null) {
                        imInternal.deleteIssue(proxyAppUserInternal, jIssueInternal as com.atlassian.jira.issue.
Issue, com.atlassian.jira.event.type.EventDispatchOption.ISSUE_DELETED, false)
                    }
                    throw issueLevelError2("""Failed to create issue: ${
                        e.message
                    }. Please review the script or contact Exalate Support""".toString(), e)
                } finally {
                   LogIn.logIn(loggedInUser)
                }
        }

        /**
```

```
         * @param whenIssueCreatedFn - a callback closure executed after the issue has been created
         * */
        static com.exalate.basic.domain.BasicIssueKey create(
                com.exalate.basic.domain.hubobject.v1.BasicHubIssue replica,
                com.exalate.basic.domain.hubobject.v1.BasicHubIssue issue,
                com.exalate.api.domain.request.ISyncRequest syncRequest,
                com.exalate.node.hubobject.v1_3.NodeHelper nodeHelper,
                com.exalate.basic.domain.hubobject.v1.BasicHubIssue issueBeforeScript,
                com.exalate.api.domain.INonPersistentReplica remoteReplica,
                List<com.exalate.api.domain.twintrace.INonPersistentTrace> traces,
                List<com.exalate.api.domain.IBlobMetadata> blobMetadataList,
                Closure<?> whenIssueCreatedFn) {

            def firstSync = com.exalate.processor.jira.JiraCreateIssueProcessor.createProcessorContext.get() ==
true
            def (_jIssue, _exIssueKey) = doCreate(replica, issue, syncRequest, nodeHelper, issueBeforeScript,
remoteReplica, traces, blobMetadataList, log)
            com.atlassian.jira.issue.MutableIssue jIssue = _jIssue as com.atlassian.jira.issue.MutableIssue
            com.exalate.basic.domain.BasicIssueKey exIssueKey = _exIssueKey as com.exalate.basic.domain.
BasicIssueKey
            try {

                whenIssueCreatedFn()
                UpdateIssue.update(replica, issue, syncRequest, nodeHelper, issueBeforeScript, traces,
blobMetadataList, jIssue, exIssueKey)
            } catch (Exception e3) {
                final def imInternal = com.atlassian.jira.component.ComponentAccessor.issueManager
                final def nservInternal2 = com.atlassian.jira.component.ComponentAccessor.
getOSGiComponentInstanceOfType(com.exalate.api.node.INodeService.class)
                def proxyAppUserInternal = nservInternal2.getProxyUser()
                if (firstSync && _jIssue != null) {
                    imInternal.deleteIssue(proxyAppUserInternal, _jIssue as com.atlassian.jira.issue.Issue, com.
atlassian.jira.event.type.EventDispatchOption.ISSUE_DELETED, false)
                }
                throw e3
            }
            return exIssueKey
        }
    }
class UpdateIssue {

        private static def log = org.slf4j.LoggerFactory.getLogger("com.exalate.scripts.Epic")

        private static def doUpdate = { com.exalate.basic.domain.hubobject.v1.BasicHubIssue replica,
                                        com.exalate.basic.domain.hubobject.v1.BasicHubIssue issue,
                                        com.exalate.api.domain.request.ISyncRequest syncRequest,
                                        com.exalate.node.hubobject.v1_3.NodeHelper nodeHelper,
                                        com.exalate.basic.domain.hubobject.v1.BasicHubIssue issueBeforeScript,
                                        List<com.exalate.api.domain.twintrace.INonPersistentTrace> traces,
                                        List<com.exalate.api.domain.IBlobMetadata> blobMetadataList,
                                        com.atlassian.jira.issue.MutableIssue jIssue,
                                        com.exalate.basic.domain.BasicIssueKey exIssueKey ->
            try {
                final def hohfInternal2 = com.atlassian.jira.component.ComponentAccessor.
getOSGiComponentInstanceOfType(com.exalate.api.hubobject.IHubObjectHelperFactory.class)
                //noinspection GroovyAssignabilityCheck
                final def hohInternal2 = hohfInternal2.get("1.2.0")
                final def nservInternal2 = com.atlassian.jira.component.ComponentAccessor.
getOSGiComponentInstanceOfType(com.exalate.api.node.INodeService.class)

                def proxyAppUserInternal2 = nservInternal2.getProxyUser()

                log.info("performing the update for the issue `" + jIssue.key + "` for remote issue `" +
replica.key + "`")
                //finally create all
                def fakeTraces2 = com.exalate.util.TraceUtils.indexFakeTraces(traces)
                def preparedIssue2 = hohInternal2.prepareLocalHubIssueForApplication(issueBeforeScript, issue,
fakeTraces2)
                //@Nonnull IIssueKey issueKey, @Nonnull IHubIssueReplica hubIssueAfterScripts, @Nullable String
proxyUser, @Nonnull IHubIssueReplica hubIssueBeforeScripts, @Nonnull Map<TraceType, List<ITrace>> traces,
@Nonnull List<IBlobMetadata> blobMetadataList, IRelation relation
```

```
                def resultTraces2
                try{
                    resultTraces2 = hohInternal2.updateNodeIssueWith(exIssueKey, preparedIssue2,
proxyAppUserInternal2.key, issueBeforeScript, fakeTraces2, blobMetadataList, syncRequest)
                } catch (MissingMethodException e){
                    resultTraces2 = hohInternal2.updateNodeIssueWith(exIssueKey, preparedIssue2,
proxyAppUserInternal2.key, issueBeforeScript, fakeTraces2, blobMetadataList, syncRequest.getConnection())
                }
                traces.clear()
                traces.addAll(resultTraces2 ?: [])
                new Result(issue, traces)
            } catch (com.exalate.api.exception.IssueTrackerException ite2) {
                throw ite2
            } catch (Exception e2) {
                throw new com.exalate.api.exception.IssueTrackerException(e2.message, e2)
            }
        }

        static Result update(com.exalate.basic.domain.hubobject.v1.BasicHubIssue replica,
                             com.exalate.basic.domain.hubobject.v1.BasicHubIssue issue,
                             com.exalate.api.domain.request.ISyncRequest syncRequest,
                             com.exalate.node.hubobject.v1_3.NodeHelper nodeHelper,
                             com.exalate.basic.domain.hubobject.v1.BasicHubIssue issueBeforeScript,
                             List<com.exalate.api.domain.twintrace.INonPersistentTrace> traces,
                             List<com.exalate.api.domain.IBlobMetadata> blobMetadataList,
                             com.atlassian.jira.issue.MutableIssue jIssue,
                             com.exalate.basic.domain.BasicIssueKey exIssueKey) {
            doUpdate(replica, issue, syncRequest, nodeHelper, issueBeforeScript, traces, blobMetadataList,
jIssue, exIssueKey)
        }

        static class Result {
            com.exalate.basic.domain.hubobject.v1.BasicHubIssue issue
            List<com.exalate.api.domain.twintrace.INonPersistentTrace> traces

            Result(com.exalate.basic.domain.hubobject.v1.BasicHubIssue issue, java.util.List<com.exalate.api.
domain.twintrace.INonPersistentTrace> traces) {
                this.issue = issue
                this.traces = traces
            }
        }
    }

int createIssueLink(){
 if (replica.parentId || replica."relation"){
    def parentLinkExists = false
    if (replica.parentId)
        flag = true
    def localParentKey = nodeHelper.getLocalIssueKeyFromRemoteId(replica.parentId ?: replica?."relationid" as
Long, "issue")?.key
    if (localParentKey==null) return 1
    final String sourceIssueKey = localParentKey
    final String destinationIssueKey = issue.key

    def linkTypeMap = [
        "Parent" : "Relates",
        "Duplicate" : "Duplicate"
        ]
    String issueLinkName

    if (!parentLinkExists)
        issueLinkName = linkTypeMap[replica."relation"]
    else
        issueLinkName = "Blocks"

    final Long sequence = 1L

    def loggedInUser = ComponentAccessor.jiraAuthenticationContext.loggedInUser
    def issueLinkTypeManager = ComponentAccessor.getComponent(IssueLinkTypeManager)
    def issueManager = ComponentAccessor.issueManager
```

```
    def sourceIssue = issueManager.getIssueByCurrentKey(sourceIssueKey)
    def destinationIssue = issueManager.getIssueByCurrentKey(destinationIssueKey)

    def availableIssueLinkTypes = issueLinkTypeManager.issueLinkTypes

    int i,f=999
    for (i=0; i<availableIssueLinkTypes.size() ; i++){
        if(issueLinkName.equals(availableIssueLinkTypes[i].name)){
            f = i
            break
        }
    }
    ComponentAccessor.issueLinkManager.createIssueLink(sourceIssue.id, destinationIssue.id,
availableIssueLinkTypes[f].id, sequence, loggedInUser)
  }
  return 1
}

if(firstSync){
   issue.projectKey   = "PM00"

    def issueMap = [
        "Epic" :"Story",
        "Feature" :"Task",
        "Task" :"Bug"
        ]
    issue.typeName  = issueMap[replica.type?.name]

    CreateIssue.create(
                    replica,
                    issue,
                    syncRequest,
                    nodeHelper,
                    issueBeforeScript,
                    remoteReplica,
                    traces,
                    blobMetadataList) {
                        if (replica.parentId || replica."relation")
                            createIssueLink()
                    }

}

issue.summary      = replica.summary
issue.description  = replica.description
issue.comments     = commentHelper.mergeComments(issue, replica)
issue.attachments  = attachmentHelper.mergeAttachments(issue, replica)
issue.labels       = replica.labels
```

Please review the following video to see the use case in action:


ADO Jira Server ... during sync.mp4