Azure DevOps <> Jira Cloud Keep Issue hierarchy

Epic -> Feature -> Story, Task -> Sub-task

In this documentation, I will show you how to keep the Epic link and issue link between issues and work items in Jira Cloud and Azure DevOps

Depending on your instance you can use different issueTypes.

1) From Azure DevOps to Jira Cloud

ADO Outgoing Sync

First, we need to add a few new lines in the Outgoing script.

ADO Outgoing sync

```
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.relations = res.relations
```

We need the parentId to check if an issue has a parent so we can link them together.

With the httpClient we get the relations of the issues if the issue does not have a relation to another issue we don't set the "replica.relations" variable.

Jira Cloud Incoming Sync

In the Jira Incoming sync, we'll start from the top and work to the bottom, you'll just have to Copy & paste these values and change some values so they match your instance.

It will be clear where and when you need to change these values.

The first code we are going to change is in the if(firstSync) block.

1) First sync block

Change to your values

Change the projct name the your project name & Change the typeMap to your issueTypes (you can choose how you want to map your issueTypes).

1. So first you need to change the project name to your Jira Cloud project name.
2. Change the values in the typeMap to the issueTypes you have on your ADO instance (the types before "**:**") and the IssueTypes on your Jira instance (the types after "**:**")
    a. Example: ["Azure devOps":"Jira Cloud"]
3. On line 11 we'll set the issueType when the type is found in the TypeMap it will set issueType to that value if it's not found it's going to look if the issuetype exsists in your project if not it will set it to a default value, a Task in this case.
4. Then on line 14, we check if the issueType is an epic, if it's an epic we will set the Epic name to the given summary

Jira Incoming Sync

```
if(firstSync){
        // Change <Project name> to your project name
           issue.projectKey   = "DEMO"
    // This typeMap hass the values that are coming from ADO if a value is not found set it to a default value
("Task" in this case).
           def typeMap = [
    // "ADO side":"Jira Cloud side"
              "User Story":"Story"
        ]
           issue.typeName     = nodeHelper.getIssueType(typeMap[replica.type?.name], issue.projectKey)?.name ?:
nodeHelper.getIssueType(replica.type?.name, (issue.projectKey ?: issue.project.key))?.name ?: "Task"
           issue.summary      = replica.summary

           if (replica.typeName == "Epic") {
              issue.customFields."Epic Name".value = replica.summary
           }
           store(issue)
}
```

2) Link issues

Here we will determine which issue needs to be linked to which issue, If the issue does not have a parentId it does not need to be linked.

Change to your values

The issueTypes shown below can be different than your issueTypes (Feature) can be something else on your instance.

1. We are going to check if the issueType is a Feature (this can be different in other instances, change "Feature" to the issueType that suits you) and that the parentId is not empty
2. The Feature here is linked under the Epic so when it's a feature we will link it to the Epic if the parentId is not empty.
3. When the next issue (Story) has a parent (Feature) they have a relation link and then the 2 issues will be linked together

Jira Incoming Sync

```
// This will check if the issueType is a Feature and if there is a parentId now it will link the epic with the
feature.
// And it will only link the issues as a child issue under the feature
if (issue.typeName == "Feature" && replica.parentId) {
    def localParent = syncHelper.getLocalIssueKeyFromRemoteId(replica.parentId.toLong())
    if (localParent) {
        issue.customFields."Epic Link".value = localParent.urn
    }
}else {

   replica.relations.each {
      relation ->
      // We check on the Related attribute from ADO and link it wiht Relates in Jira
      if (relation.attributes.name == "Parent"){
            def a = syncHelper.getLocalIssueKeyFromRemoteId(relation.url.tokenize('/')[7])//?.urn
            if (issue.issueLinks[0]?.otherIssueId != a?.id){
                def res = httpClient.put("/rest/api/2/issue/${issue.key}", """
                {
                   "update":{
                      "issuelinks":[
                         {
                            "add":{
                               "type":{
                                  "name":"Relates"
                               },
                               "outwardIssue":{
                                  "key":"${a.urn}"
                               }
                            }
                         }
                      ]
                   }
                }
                """)
            }
        }
   }
}
```

3) Status mapping (additional)

If you also want to map your status for multiple issueTypes you can use this function.

> **Change to your values**
>
> Change the values to the values you get from ADO (first values before the "**:**") and the values you have in your Jira issueTypes (last values after the "**:**")

Jira Incoming Sync

```
def setStatus(){
// First we determine which Issue Type has which statuses Epic, Feature, Story, etc...
   def statusMappingEpic = [
        // "ADO values":"Jira Values"
        "Open":"Open",
        "Doing":"In Progress",
        "Closed":"Done"
    ]
    def statusMappingFeature = [
        // "ADO values":"Jira Values"
        "To Do":"Open",
        "Doing":"In Progress",
        "Closed":"Done"
    ]
     def statusMappingStory = [
        // "ADO values":"Jira Values"
        "To Do":"Open",
        "Doing":"In Progress",
        "Closed":"Done"
    ]
    def remoteStatusName = replica.status.name // Status name from the ADO side
    // We wil check which issueType this issue has and them map the right Statuses to it, the default value is
set to the default value you want.
    if (issue.type.name == "Epic"){ return statusMappingEpic[remoteStatusName] ?: "Open"}
    if (issue.type.name == "Feature"){ return statusMappingFeature[remoteStatusName] ?: "Open"}
    if (issue.type.name == "User Story"){ return statusMappingStory[remoteStatusName] ?: "Open"}
    // We return the right value and set the right Status in your issue
}
// We do this after the first sync other wise it can cause troubles.
if (!firstSync){
   workItem.setStatus(setStatus())
}
```

4) System & Custom Fields.

Now we have done the parent-child link we only need to add the System or custom fields that you also want to set in your Jira issue.

Jira Incoming Sync

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

// Custom Fields
//issue.customFields."CF Name".value = replica.customFields."CF Name".value
```

2) From Jira Cloud to Azure DevOps

Jira Cloud Outgoing Script

First, we need to add a few values in the Jira Outgoing sync.

By default "replica.parentId = issue.parentId" is already in the outgoing script but check this to be sure.

Jira Outgoing sync

```
// Add these to the outging script
replica.linkedIssues   = issue.issueLinks
replica.parentId       = issue.parentId
```

We need the linked issues and the parentId to see in ADO which issues are linked with each other.

Azure DevOps Incoming Sync

Here we also start from the top to the bottom, you can also copy & paste these values and change them where needed to match your instance values.

It will be clear where and when you need to change these values.

1) First sync block

Change to your values

Change the projct name the your project name & Change the typeMap to your issueTypes (you can choose how you want to map your issueTypes).

1. So first you need to change the project name to your Jira Cloud project name.
2. Change the values in the typeMap to the issueTypes you have on your Jira instance (the types before "**:**") and the IssueTypes on your ADO instance (the types after "**:**")
    a. Example: ["Jira Cloud":"Azure devOps"]
3. On line 10 we'll set the issueType when the type is found in the TypeMap it will set issueType to that value if it's not found it's going to look if the issuetype exsists in your project if not it will set it to a default value, a Task in this case.

ADO Incoming Sync

```
if(firstSync){
   workItem.projectKey  =  "<project name>"
   def typeMap = [
      // "Jira Cloud":"ADO"
      "Epic" : "Epic",
      "Feature": "Feature",
      "Story" : "User Story"
   ]

   workItem.typeName = nodeHelper.getIssueType(typeMap[replica.type?.name], issue.projectKey)?.name ?:
nodeHelper.getIssueType(replica.type?.name, (issue.projectKey ?: issue.project.key))?.name ?: "Task"
   workItem.summary      = replica.summary
   store(issue)
}
```

2) We will set the parentId if the remote issue has a parentId

ADO Incoming sync

```
workItem.parentId = null
if (replica.parentId) {
   def localParent = syncHelper.getLocalIssueKeyFromRemoteId(replica.parentId.toLong())
   if (localParent) {
      workItem.parentId = localParent.id
   }
}
```

Change to your values

On line 1 change <ADO Project> to your actual project.

ADO Incoming Sync

```
def res =httpClient.get("/<ADO Project>/_apis/wit/workItems/${workItem.id}/revisions",true)
def await = { f -> scala.concurrent.Await$.MODULE$.result(f, scala.concurrent.duration.Duration.apply(1, java.
util.concurrent.TimeUnit.MINUTES)) }
def creds = await(httpClient.azureClient.getCredentials())
def token = creds.accessToken()
def baseUrl = creds.issueTrackerUrl()
def project = workItem.projectKey
def localUrl = baseUrl + '/_apis/wit/workItems/' + workItem.id
int x =0
res.value.relations.each{
   revision ->
      def createIterationBody1 = [
            [
               op: "test",
               path: "/rev",
               value: (int) res.value.size()
            ],
            [
               op:"remove",
               path:"/relations/${++x}"
            ]
        ]

}

def linkTypeMapping = [
   "relates to": "System.LinkTypes.Related"
]
def linkedIssues = replica.linkedIssues
if (linkedIssues) {
   replica.linkedIssues.each{
      def localParent = syncHelper.getLocalIssueKeyFromRemoteId(it.otherIssueId.toLong())
      if (!localParent?.id) { return; }
      localUrl = baseUrl + '/_apis/wit/workItems/' + localParent.id
      def createIterationBody = [
            [
               op: "test",
               path: "/rev",
               value: (int) res.value.size()
            ],
            [
               op:"add",
               path:"/relations/-",
               value: [
                  rel:linkTypeMapping[it.linkName],
                  url:localUrl,
                  attributes: [
                     comment:""
                  ]
               ]
            ]
       ]

   def createIterationBodyStr = groovy.json.JsonOutput.toJson(createIterationBody)
      converter = scala.collection.JavaConverters;
      arrForScala = [new scala.Tuple2("Content-Type","application/json-patch+json")]
      scalaSeq = converter.asScalaIteratorConverter(arrForScala.iterator()).asScala().toSeq();
      createIterationBodyStr = groovy.json.JsonOutput.toJson(createIterationBody)
      def result = await(httpClient.azureClient.ws
            .url(baseUrl+"/${project}/_apis/wit/workitems/${workItem.id}?api-version=6.0")
            .addHttpHeaders(scalaSeq)
            .withAuth(token, token, play.api.libs.ws.WSAuthScheme$BASIC$.MODULE$)
            .withBody(play.api.libs.json.Json.parse(createIterationBodyStr), play.api.libs.ws.
JsonBodyWritables$.MODULE$.writeableOf_JsValue)
            .withMethod("PATCH")
            .execute())

   }
}
```

3) Status mapping (additional)

If you also want to map your status for multiple issueTypes you can use this function.

> **Change to your values**
>
> Change the values to the values you get from Jira (first values before the "**:**") and the values you have in your ADO issueTypes (last values after the "**:**")

You can add other Issuetype Statues.

ADO incoming

```
def setStatus(){
// First we determine which Issue Type has which statuses Epic, Feature, Story, etc...
   def statusMappingEpic = [
        // "Jira values":"ADO Values"
        "Open":"Open",
        "In Progress":"Doing",
        "Done":"Closed"
    ]
    def statusMappingFeature = [
        // "Jira values":"ADO Values"
        "Open":"To Do",
        "In Progress":"Doing",
        "Done":"Closed"
    ]
     def statusMappingStory = [
        // "Jira values":"ADO Values"
        "Open":"To Do",
        "In Progress":"Doing",
        "Done":"Closed"
    ]
    def remoteStatusName = replica.status.name // Status name from the Jira side
    // We wil check which issueType this issue has and them map the right Statuses to it, the default value is
set to the default value you want.
    if (issue.type.name == "Epic"){ return statusMappingEpic[remoteStatusName] ?: "Open"}
    if (issue.type.name == "Feature"){ return statusMappingFeature[remoteStatusName] ?: "To Do"}
    if (issue.type.name == "User Story"){ return statusMappingStory[remoteStatusName] ?: "To Do"}
    // We return the right value and set the right Status in your issue
}
// We do this after the first sync other wise it can cause troubles.
if (!firstSync){
   workItem.setStatus(setStatus())
}
```

4) System & Custom Fields.

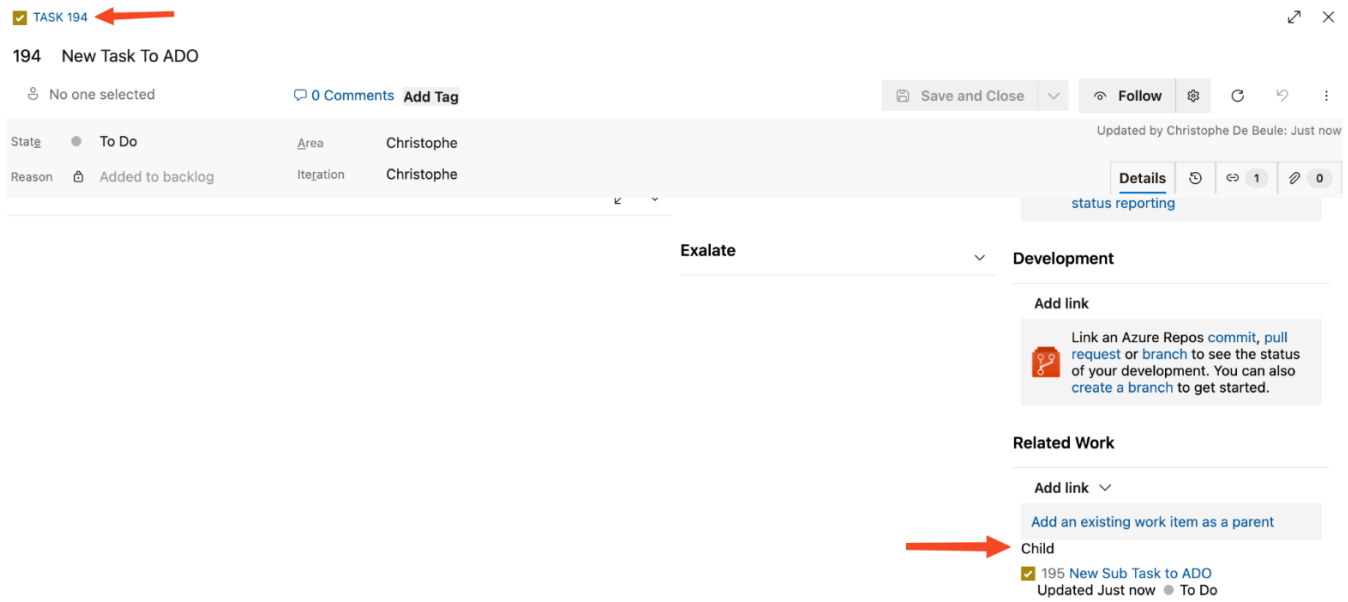
Now we have done the parent-child link we only need to add the System or custom fields that you also want to set in your Jira issue.

ADO Incoming Sync

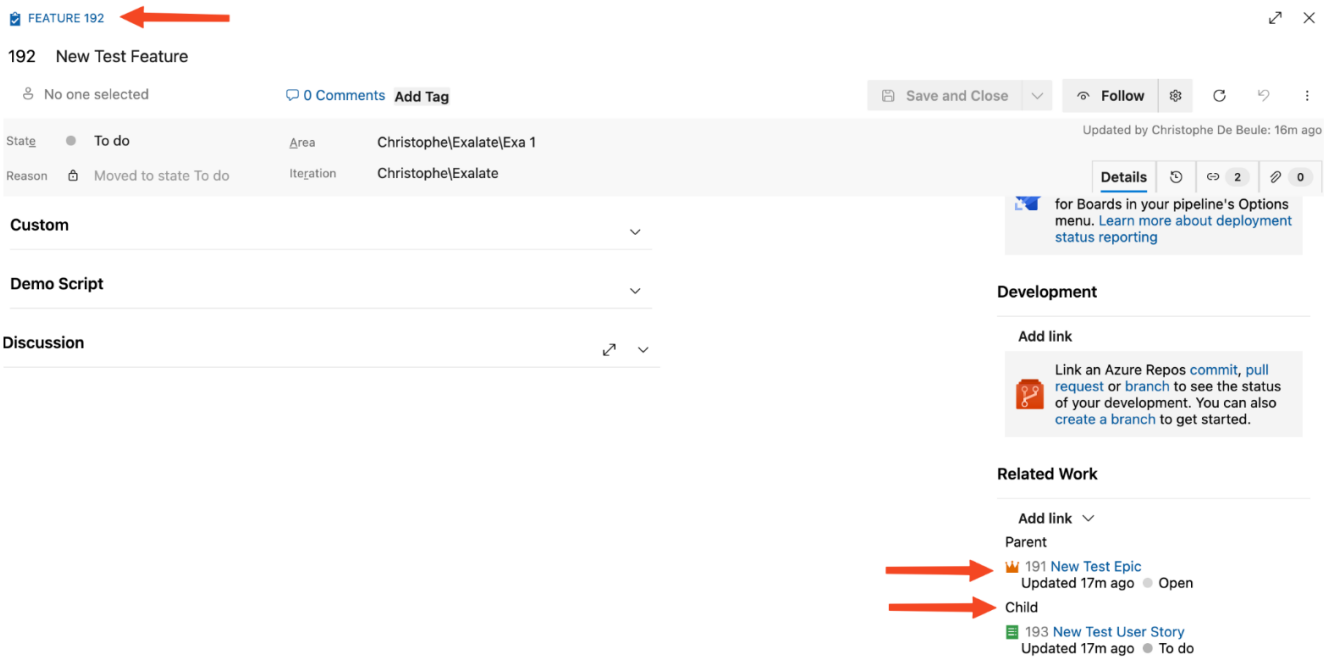
```
workItem.summary      = replica.summary
workItem.description  = replica.description
workItem.attachments  = attachmentHelper.mergeAttachments(workItem, replica)
workItem.comments     = commentHelper.mergeComments(workItem, replica)
workItem.labels       = replica.labels
workItem.priority     = replica.priority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

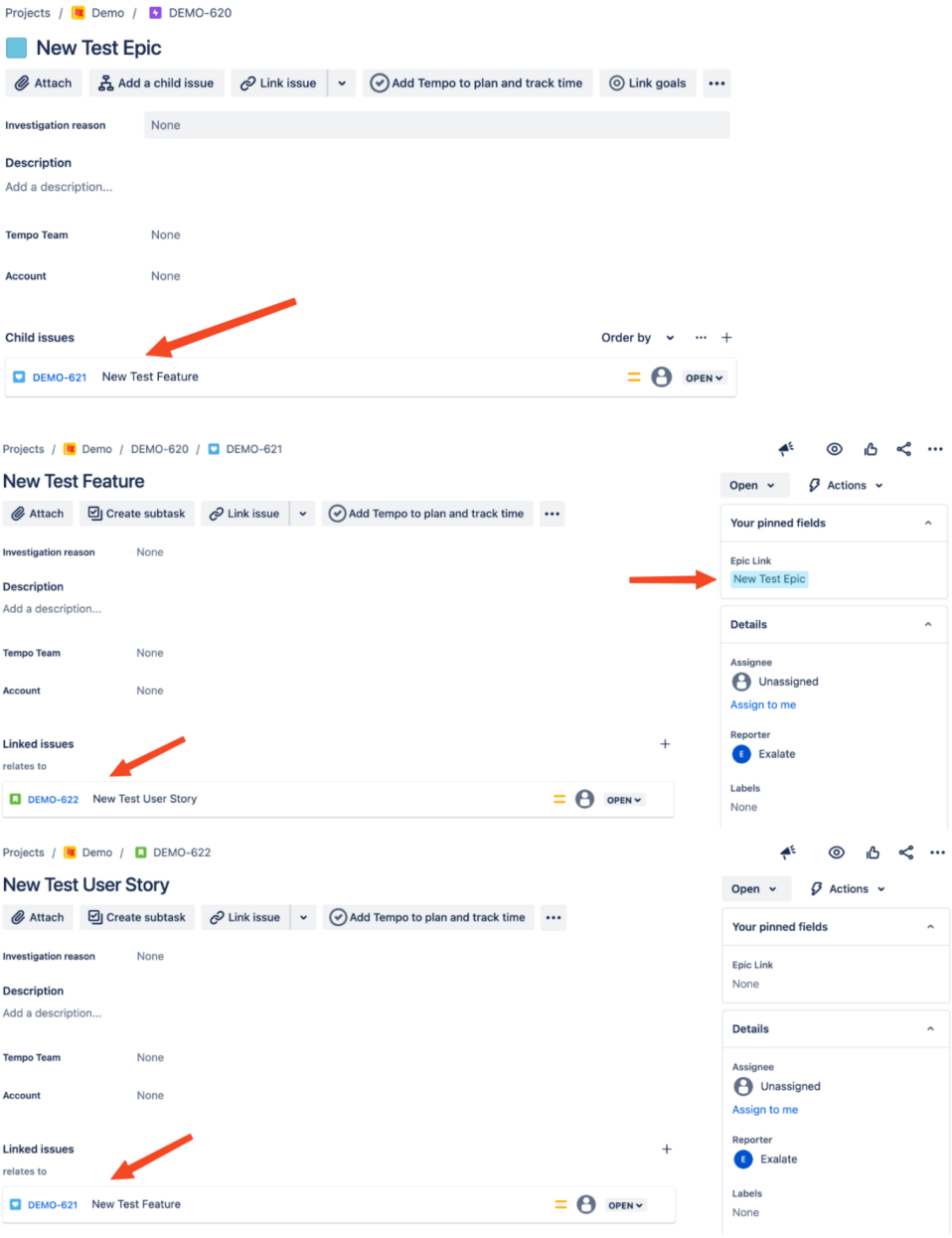
And you're done This is the implementation to keep the issue link hierarchy.

Here is an image of how a Task and a sub-task are synced over from Jira in ADO.



Here is an image of how the Epic hierarchy is synced over




Projects / Demo / DEMO-620
New Test Epic
Attach
Add a child issue
Link issue
Add Tempo to plan and track time
Link goals
Investigation reason
None
Description
Add a description...
Tempo Team
None
Account
None
Child issues
Order by
DEMO-621
New Test Feature
OPEN
Projects / Demo / DEMO-620 / DEMO-621
New Test Feature
Attach
Create subtask
Link issue
Add Tempo to plan and track time
Investigation reason
None
Description
Add a description...
Tempo Team
None
Account
None
Linked issues
relates to
DEMO-622
New Test User Story
OPEN
Open
Actions
Your pinned fields
Epic Link
New Test Epic
Details
Assignee
Unassigned
Assign to me
Reporter
Exalate
Labels
None
Projects / Demo / DEMO-622
New Test User Story
Attach
Create subtask
Link issue
Add Tempo to plan and track time
Investigation reason
None
Description
Add a description...
Tempo Team
None
Account
None
Linked issues
relates to
DEMO-621
New Test Feature
OPEN
Open
Actions
Your pinned fields
Epic Link
None
Details
Assignee
Unassigned
Assign to me
Reporter
Exalate
Labels
None