How to Sync and Maintain Issue links, Relations, and Sub-task Mappings between Jira and Azure DevOps

Table of contents

This article was originally published on the Atlassian Community.

[exalate_title]

[exalate_title] [exalate_id]
Companies often need to set up advanced or customized integrations between Jira and Azure DevOps. Such integrations are commonly used to streamline communication between disparate applications.

In this article, we discuss how to maintain issue links and their relationship types between Jira and Azure DevOps. We also see a secondary requirement of syncing the parent-child equation of issues and sub-tasks.

We use a synchronization solution called Exalate to implement this use case.

Get guide on email

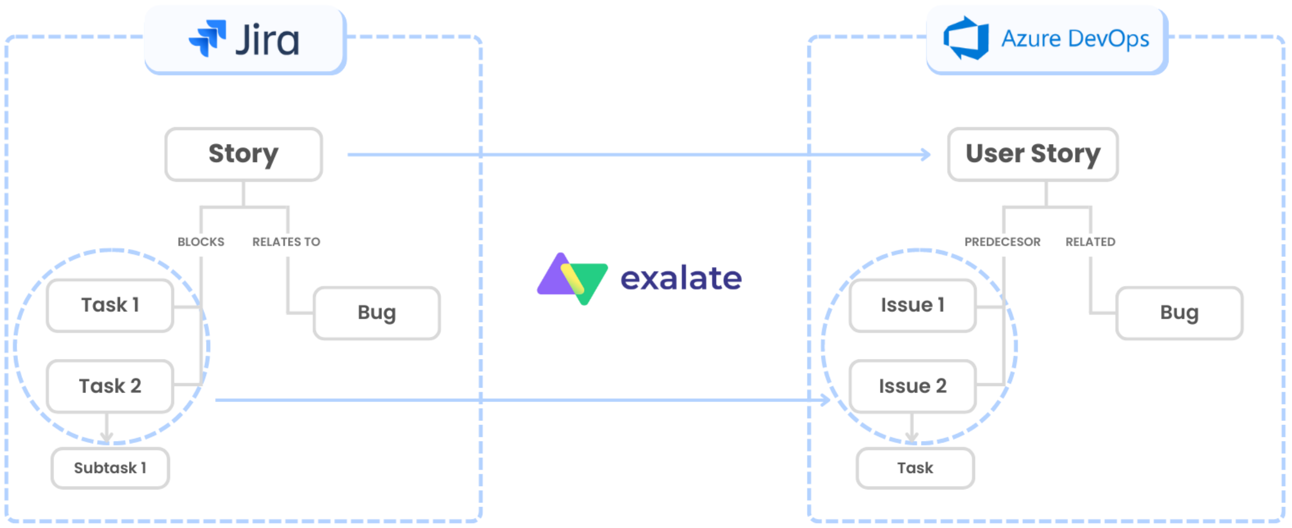
The Use Case

The following are the use case requirements:

- Stories in Jira should arrive as a User Stories (work item) in Azure DevOps.
- Stories have issue links: Tasks and Bugs. These need to be reflected in Azure DevOps under the corresponding User Story as issues and Bugs.
- Tasks have a “blocks” relationship type with the Story. They must be synced over as issues with a “predecessor” relation to the User Story in Azure DevOps.

- Bugs in Jira have a “relates to” relationship with the Story. They should be reflected as bugs work item type with a relation “related” to the User Story in Azure DevOps.
- Sub-tasks of Tasks in Jira are to be mapped to Tasks (as a child relationship) under the corresponding issues in Azure DevOps.

A visual depiction of the use case will make things clearer.



Potential Challenges

The requirements are unique and advanced. So there are a couple of challenges we need to address first:

- Syncing a Story in Jira to User Story in Azure DevOps
  Tasks in Jira fly as issues in Azure DevOps. Bugs are to be kept as Bugs. Maintaining these mappings is important.
- Mapping of the parent-child relationship between the sub-tasks and Tasks in Jira to Tasks and issues in Azure DevOps
- Customizing the relationship types fully
  You can define and map them as you want. For instance, the “blocks” relationship type is mapped to “predecessor” in Azure DevOps. Or the “relates to” relationship type is mapped as “related” in Azure DevOps. These are just examples, the actual mappings will depend on your synchronization requirements.

After taking note of these challenges, it’s time to implement the use case.

But first, we need to understand why we chose Exalate.

3rd-Party Synchronization Solution: Exalate

Exalate is a 3rd-party synchronization solution that supports integrations for various work management systems like Jira, ServiceNow, Azure DevOps, Salesforce, Zendesk, GitHub, etc.

Why Exalate

We chose Exalate to implement this advanced integration use case for the following reasons:

- Intuitive Scripting Engine
  It has Groovy-based scripts that help set up complex logical mappings between the entities to be synced.
- Advanced automatic sync triggers
  It supports triggers that can be fine-grained to advanced levels and enable automatic synchronization of data. These are native to the systems under sync.
  For instance, you can create triggers in JQL if you use Jira and in WIQL if you use Azure DevOps.
- Independent control of information flow
  Since the use case we want to implement needs complex mappings in both Jira and Azure DevOps, we need independent control of the information to be sent and received. This ensures that you don't mess with each other's synchronization.
- Bulk synchronization of entities

Sometimes there is a need to sync entities in bulk if a particular condition is satisfied.

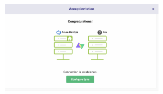
Sync Issue Links, Relations, and Sub-task Mappings between Jira and Azure DevOps Using Exalate

Start by installing Exalate on the integrating systems, Jira and Azure DevOps in our case.

Then you can establish a connection between the 2 instances in the Script Mode.

You can also find a step-by-step guide on how to do that.

The Script mode has Groovy-based scripts in the form of "Incoming sync" and "Outgoing sync".



When a connection is established, click on the “Configure Sync” button or edit the connection to access the “Rules” tab. This is where the syncs we just talked about reside.

These syncs are present in both Jira and Azure DevOps instances. That means:

- Outgoing sync defines what information is sent from the source to the destination.
- Incoming sync defines how information must be received from the source.

The Scripts

Let’s look at the actual scripts required to implement this use case.

Outgoing Sync: Jira

Essentially, the only information that must move out of Jira are the issue links and the parent ID for the sub-tasks.

The following code does it:

```
replica.linkedIssues = issue.issueLinks
 replica.parentId       = issue.parentId
```

The main mapping logic is on the Azure DevOps side.

Incoming Sync: Azure DevOps

You must add a parent-child relationship corresponding to the task-sub-task relation in Jira.

Take help from the following code to do that:

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

Next, we must cover the relationship type requirement, i.e mapping the links and creating them in Azure DevOps.

To do that, we create a mapping and then call an end-point to get the relevant links created in Azure DevOps.

Refer to the following code:

```
def linkTypeMapping = [
    "blocks": "System.LinkTypes.Dependency-Reverse",
    "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$)
         .withBody(play.api.libs.json.Json.parse(createIterationBodyStr), play.api.libs.ws.JsonBodyWritables$
         .withMethod("PATCH")
         .execute())

    }
}
```

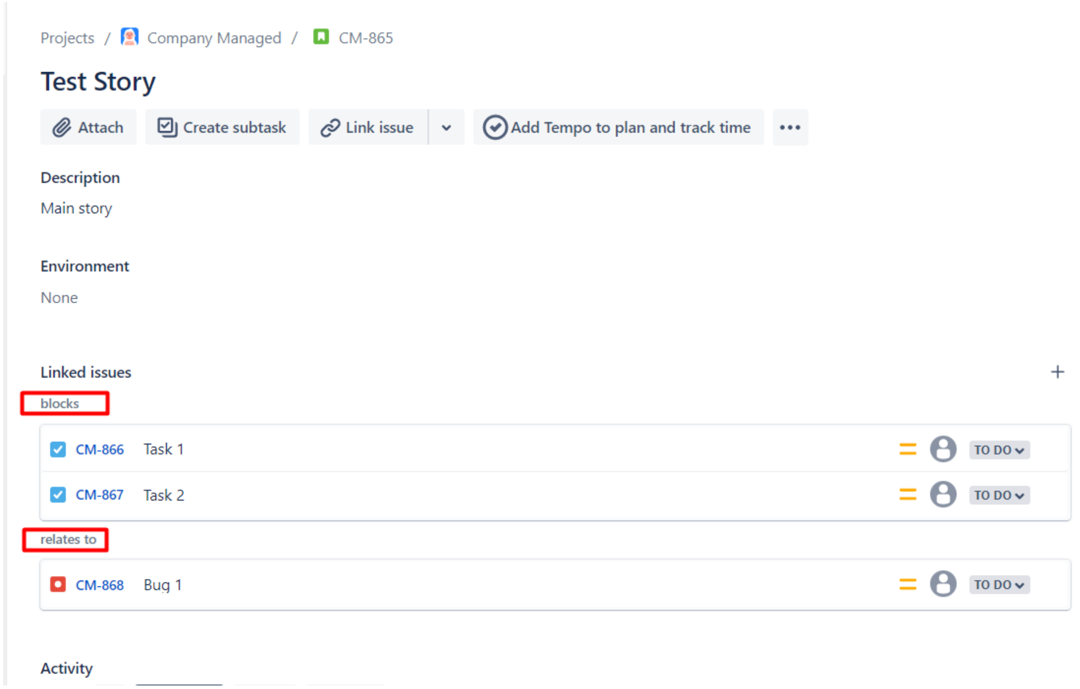
Once you have set all of this up, it’s time to see the output.

Output

You can set up automatic sync triggers to start the sync whenever a Story is created in Jira.

In Jira

1. Create a story in Jira.
2. Create the issue links
    1. Create 2 tasks: Task 1 and Task 2 with the required relationship types.
    2. Create a bug with the required relationship type.



Projects / Company Managed / CM-865

Test Story

Description

Main story

Environment

None

Linked issues

blocks

CM-866	Task 1	TO DO
CM-867	Task 2	TO DO

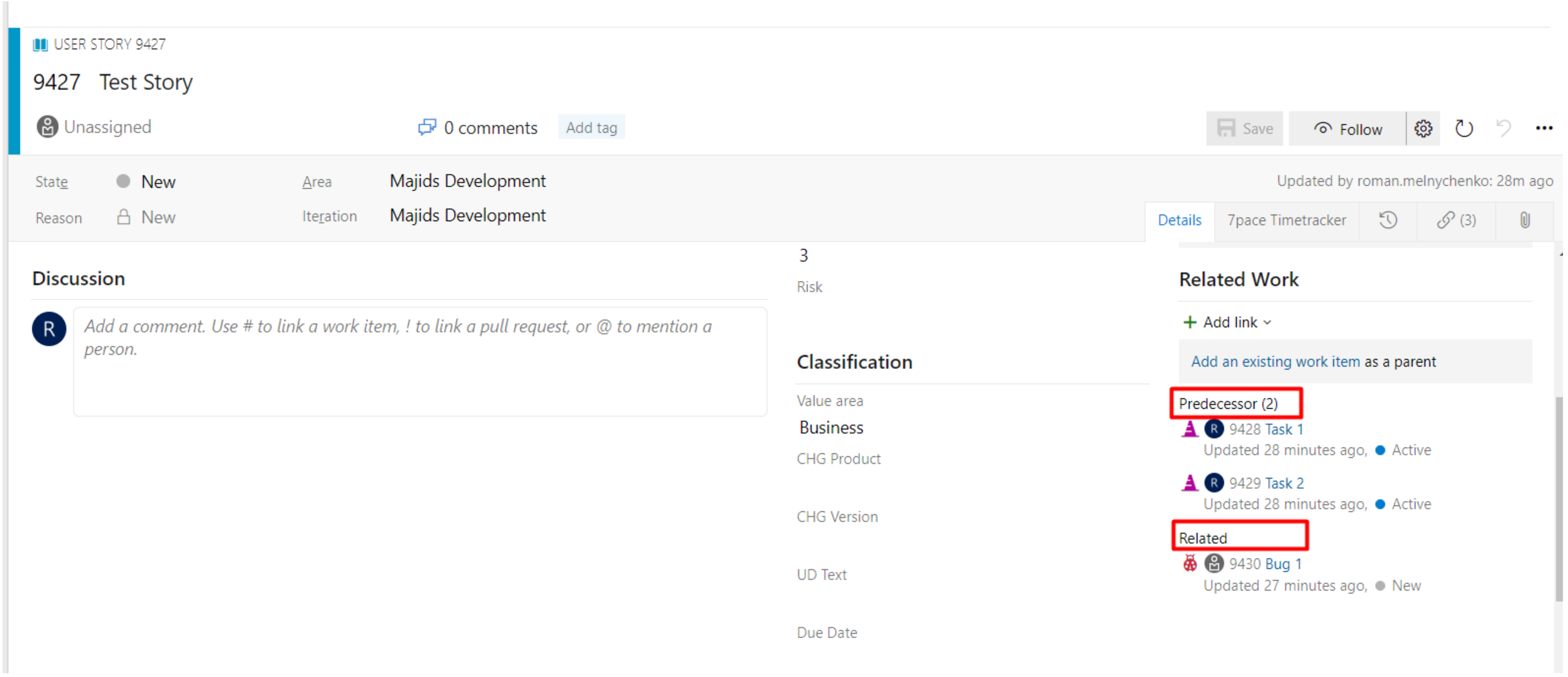
relates to

CM-868	Bug 1	TO DO

Activity

In Azure DevOps

Work items and the relations are synced and the parent-child hierarchy is maintained.



USER STORY 9427

9427 Test Story

Unassigned 0 comments Add tag

State New Area Majids Development

Reason New Iteration Majids Development

Updated by roman.melnychenko: 28m ago

Discussion

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

3

Risk

Classification

Value area: Business

CHG Product

CHG Version

UD Text

Due Date

Related Work

Add an existing work item as a parent

Predecessor (2)

9428 Task 1 — Updated 28 minutes ago, Active

9429 Task 2 — Updated 28 minutes ago, Active

Related

9430 Bug 1 — Updated 27 minutes ago, New

***Note**: For additional information about Exalate, watch the Exalate [Academy videos](#).*

Conclusion

Integrating Jira and Azure DevOps can bring in a lot of benefits. It can reduce manual efforts, increase the visibility of important information, and bring everyone on the same page. You can use 3rd-party tools like Exalate to implement similar integrations.

Book a demo with us to see how it works for your specific scenario.