Design Document for Publication Workflows

This document looks at some possible designs to allow remotely stored datasets. It's written as part of implementing issue #3561. The idea is to have a framework that's more flexible than strictly required by #3561, allowing Dataverse to support multiple remote systems and various scenarios.

Goal

Allow extensible workflow for publishing datasets. The workflow has to support external systems, including waiting for them to successfully finish a task. The system also needs to support recovery, in case of external system failure. Additional design goal is to allow workflows from other scenarios in the application.

Model Description

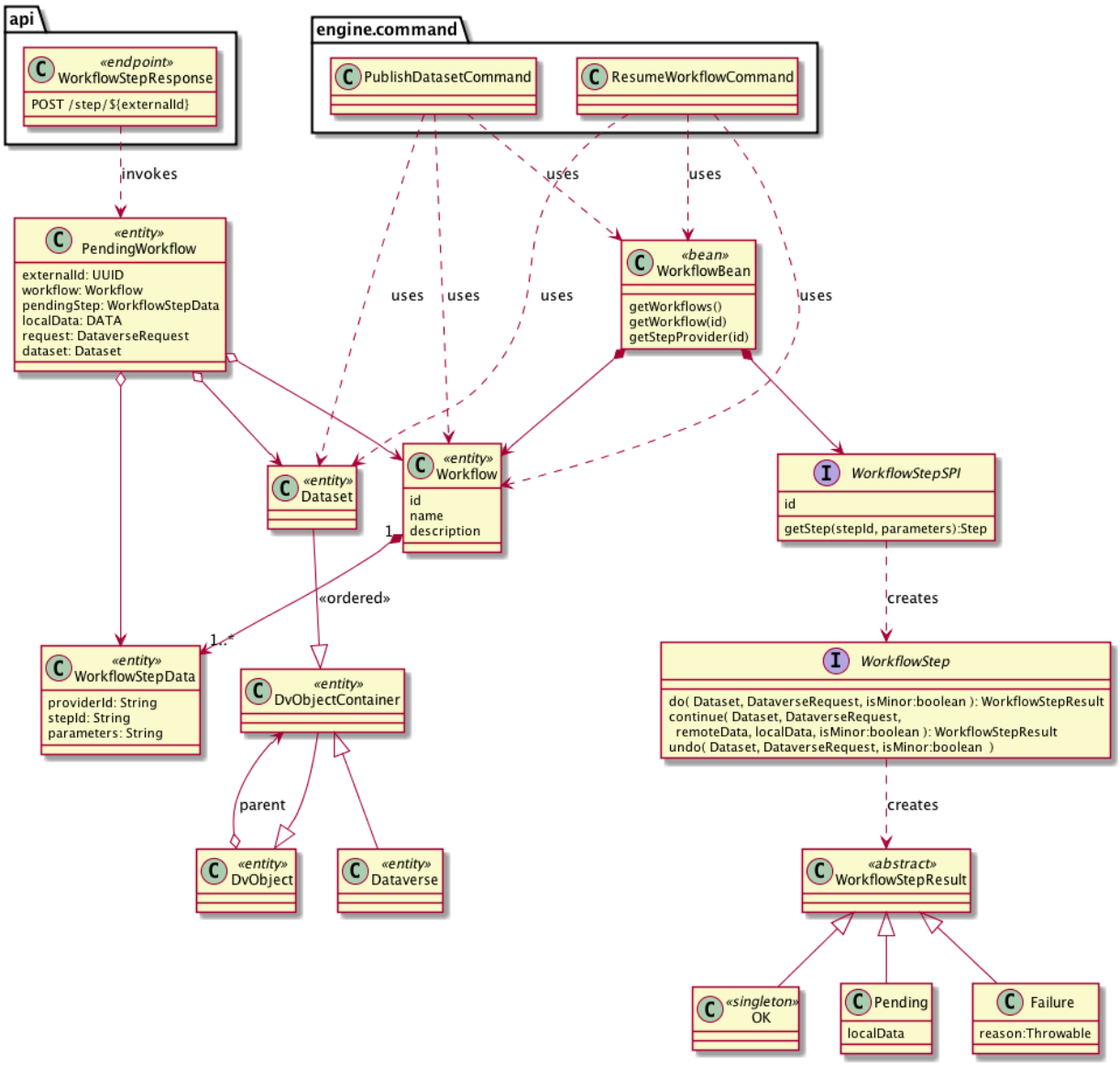
Each Dataset is associated with a workflow for publishing a new version. Exactly how this association is made is left to decision point 1. A `Workflow` is just an ordered list of `WorkflowStepData`, a record-like object that can be used to instantiate a `WorkflowStep` object. `WorkflowStep` objects perform the actual workflow steps.

Executing a workflow step has three possible outcomes:

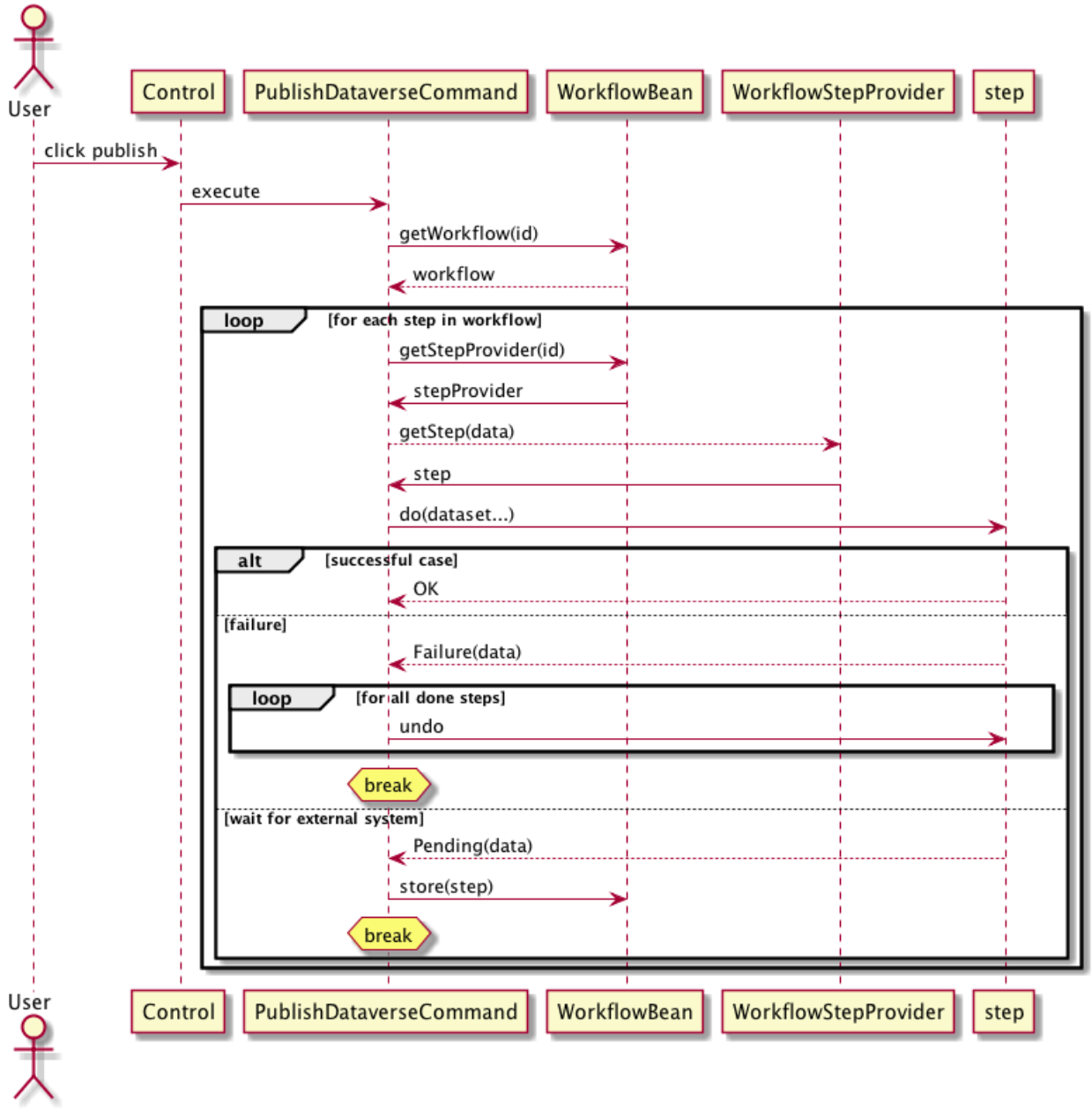
- OK - success, workflow can move on to next step
- Failure - Step failed (with some failure data provided). Workflow has to be rolled back.
- Pending - Step has to wait for an external system to return some result. This result includes a localData field, that will be passed to the step when it is resumed.

External systems return results by sending a HTTP POST request to a new API endpoint. The body of the result is then passed to the step in order to resume its execution. Resuming a workflow step can have the same three results mentioned above, and they are treated the same way.

After the workflow completes, the dataset status is changed to *published*.

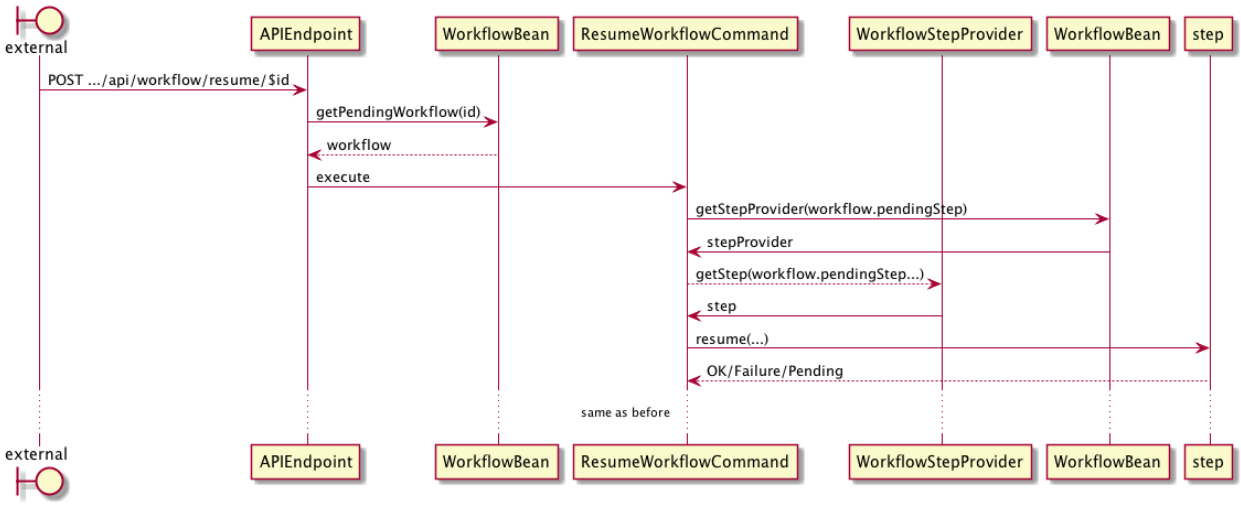

api
«endpoint»
WorkflowStepResponse
POST /step/${externalId}
invokes
engine.command
PublishDatasetCommand
ResumeWorkflowCommand
uses
uses
uses
uses
uses
uses
«entity»
PendingWorkflow
externalId: UUID
workflow: Workflow
pendingStep: WorkflowStepData
localData: DATA
request: DataverseRequest
dataset: Dataset
«bean»
WorkflowBean
getWorkflows()
getWorkflow(id)
getStepProvider(id)
«entity»
Dataset
«entity»
Workflow
id
name
description
WorkflowStepSPI
id
getStep(stepId, parameters):Step
1
«ordered»
1..*
creates
«entity»
WorkflowStepData
providerId: String
stepId: String
parameters: String
«entity»
DvObjectContainer
WorkflowStep
do(Dataset, DataverseRequest, isMinor:boolean): WorkflowStepResult
continue(Dataset, DataverseRequest,
remoteData, localData, isMinor:boolean): WorkflowStepResult
undo(Dataset, DataverseRequest, isMinor:boolean)
creates
parent
«entity»
DvObject
«entity»
Dataverse
«abstract»
WorkflowStepResult
«singleton»
OK
Pending
localData
Failure
reason:Throwable

Starting a Publication Workflow



Resuming a Publication Workflow

The external system have received a request with a unique ID, and returned a HTTP POST with that id in the URL, and some (optional) response body. The POST request is received by a new API endpoint, that finds the pending workflow in the database and resumes it.



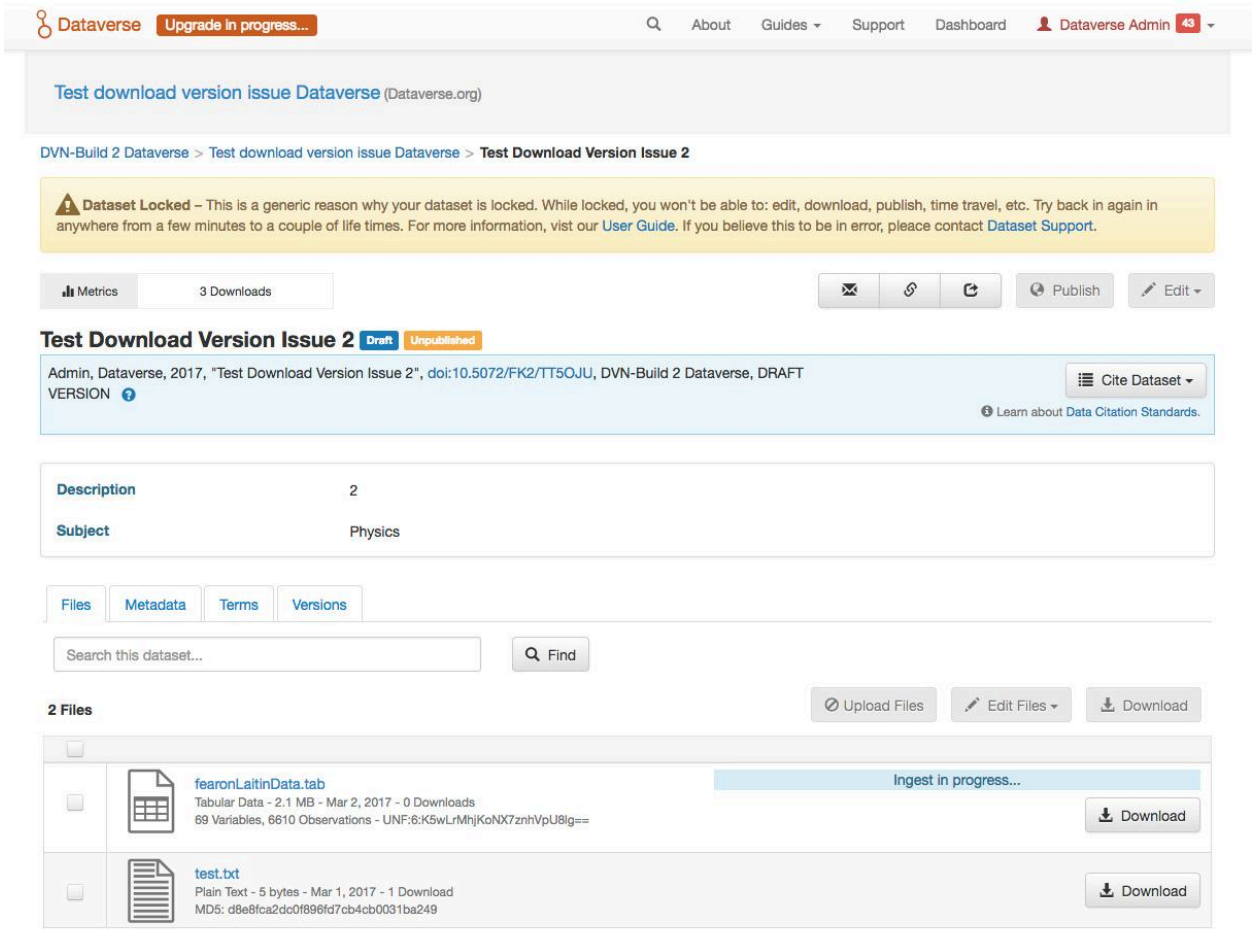
Adding Support for a New Set of Tools

This is done by implementing the WorkflowStepSPI, and registering the provider instance with the WorkflowBean at startup (similar to identity providers etc.)

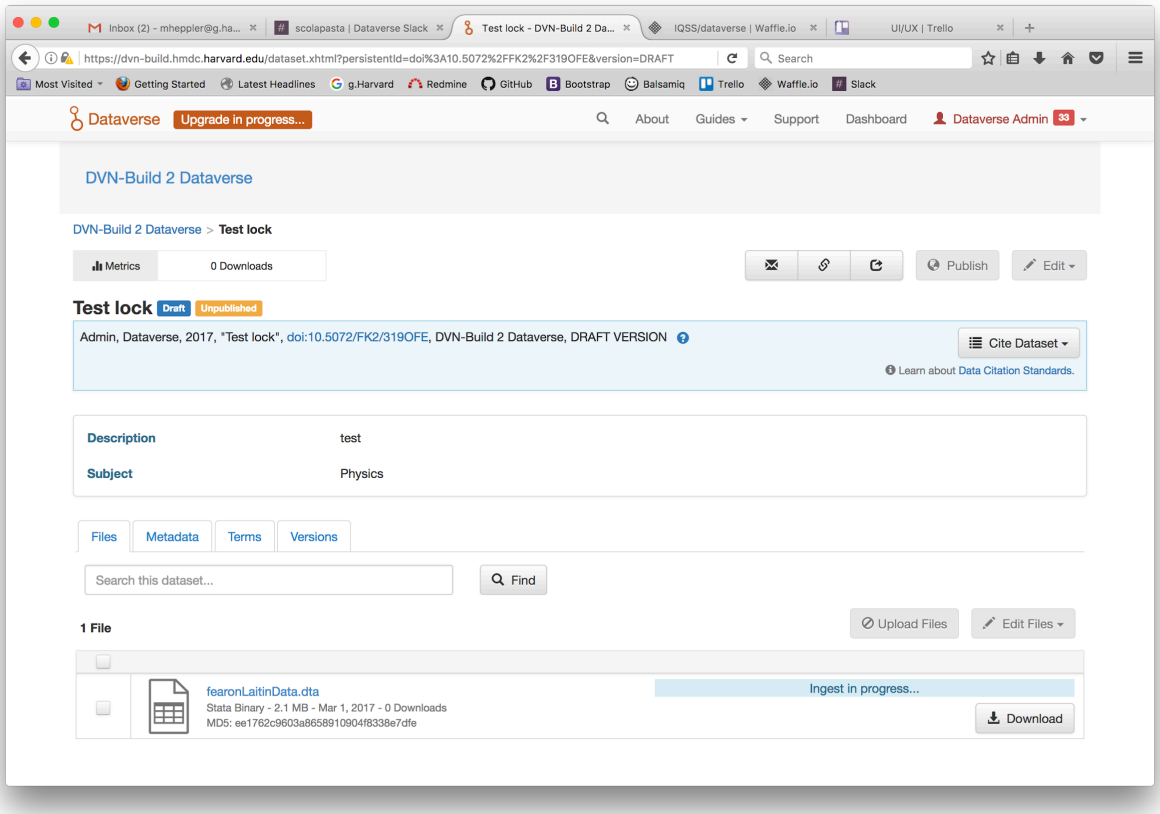
Changes to Dataverse

- Add a “LockReason” enum
    - (none) - dataset is not locked
    - Ingest (currently a boolean field)
    - InReview (currently a boolean field)
    - InPublicationProcess
- Add a "Publication Error" status to Dataset. Also needs a place to see what the error was.
- Add a way to manage workflows (API only) [DONE]
- Add an API way of inspecting current workflow status
- Need UI team:
    - Add a "current workflow status" to the UI (?)
    - Add a workflow error screen to the UI, allowing the user to find out why a certain workflow failed.

Proposed UI for a locked dataset upon generic reason for locking:



Current UI for a locked dataset upon tabular file ingest:



Decision Points

1. Associating datasets and publication workflows:
    a. **installation-wide**
    b. directly
    c. By dataverse
    d. By dataverse, plus inheritance from parent dataverses ← Nice to have
    e. By dataverse, plus inheritance from parent dataverses and local overrides
        i. in practice, this adds the workflow reference field to theDvObjectContainer class, so as easy as “c”.
2. Current design is only for pre-publish workflows. We can add post-publish workflows as well. Should we add workflows for other stages of the dataset lifecycle?
    a. Answer - post-harvest is on the horizon. Others may follow. Keep this as general as possible.
3. Data type for localData:
    a. String
    b. Blob
    c. **Map<String,String>**

    d. JSON I tend to go with the map, as it offers a nice middle-ground between non-stuctured (string/blob) and fully structured bug hard to traverse (json).
4. Data type for remoteData:
    a. String
    b. Blob
    c. JSON

I tend to go with String. String is more generic, and allows JSON where needed. But also keeps the simple cases simple.

5. API Endpoint security - preventing attacks from the API endpoint. I'm not sure this would be a high-value target, plus the attacker has to know the id of the step being resumed to be successful. Still, some precautions should probably be taken. Options:
    a. None - attacker has to know the step UUID, so we got some basic level anyway
    b. Whitelist IP addresses, manage it using the SETTINGS api. [DONE]
    c. System-wide token
    d. Step id/remote token pair: Have a token for each registered remote system. Each pending step knows which system it waits for, and so when the reply comes (with the system's token) we can check that the token matches the expected one.