# Stanford Digital Library Workflow

```
Outline
- Intro, background, needs
- Stanford approach
   - overview
   - concepts
   - features
- WorkDo structure
   - definitions
   - data in object
   - the workflow datastream, options, DB backend
- How WorkDo works
   - Human and program tasks
   - Queues
      - queue by query
      - reification of queues
   - Robots
      - Independent, reusable
      - Parallel, multiple
      - Restartable
      - Prioritization
      - Exception processing
   - item-by-item processing
   - future dated processing
   - Web applications
- Services
- Multiple workflows
- Lifecycle milestones
- Admin interface
   - current quick queue status one
   - plans for better one, including facets
- Pros and Cons
- Future development
```

## WorkDo: A Lightweight Workflow Solution in a Fedora-based Digital Library Ecosystem

"Lightweight workflow" is both an oxymoron and a continual aspiration of the many stakeholders in the repository community. Orchestrating multi-step processes that may include human interaction is integral to any digital library environment. Consider these use cases:

- Digitization workflows
- Repository Submission and Ingestion workflows
- Preservation workflows
- Library Accessioning workflows

Processing each of these workflows is a multi-step process involving both human and program-based activities. And while there is much overlap in the steps and methods across each, there is a substantial variability across all four use cases (i.e., ingest shares some steps with accessioning or preservation processes, but has many unique components). Each "family" of workflows might take many different manifestations as well, depending on the type of resource, collection or context of the objects being worked with (i.e., digitization workflow for a book will vary from that of an image or other media).

There are two basic approaches to workflow. Each can be coded independently as needed to individual tasks, or one can implement a stand-alone workflow engine with the flexibility, and overhead, of a large-scale, general purpose tool. The former can be a quick and effective solution, especially for a smaller number of workflows which are relatively stable. The latter would seem necessary for volume and scale, but unless you have a robust workflow solution already comfortably at play in your environment, investigating workflow solutions can change the focus from what work needs to get done to an often-larger project *about* workflow and associated software and tools.

Stanford sought a middle path -- one with a higher degree of reuse and flexibility than independently coding steps for every process and variation on a process, but one that avoided the overhead and complexity of running a general purpose workflow system. A Digital Library architecture centered around a central Fedora-based object registry provided a context and framework to make this possible.

The Stanford Libraries' approach to workflow in its digital library infrastructure has been a minimalist one. Leery of workflow efforts that take on a life of their own and that require their own separate resources to sustain, we have worked on a lightweight approach to supporting work that satisfies basic lifecycle requirements for the deposit, accessioning, archiving and delivery of digital resources. This method does not involve external rule or state engines, messaging, or separate process orchestration software, yet it supports the cascade of tasks necessary to manage

these digital objects. With a large queue of digital objects ready to manage, we focused on the tasks that needed to happen, not the processes that would make them happen. With this emphasis on the work to *do*, our scheme was dubbed **Work*Do*** rather than **Work*flow***, a name that has stuck despite efforts to find a more elegant alternative.

## The Stanford Approach: WorkDo [still don't like the name]

A Digital Object Registry (DOR) is at the heart of our digital library management architecture. It supports managing an object from the moment it is identified as a library resource through the transfer of content to our preservation and discovery/delivery environments. It is built on Fedora, an open-source digital object repository system that provides the basic infrastructure for keeping together content and metadata for complex digital objects.

Around DOR is an eco-system of services to support object deposit, conversion, metadata enrichment, derivatives generation, packaging, tracking, etc. A key strategy in managing our digital objects is that they be "assembled" in DOR -- incoming resources are registered, then worked over time as processes prepare them for our *Access* and *Preservation* environments. Later activity against resources, e.g., metadata enrichment or format migration, are similarly managed through DOR-centered processes, allowing the state of any resource to be fully tracked throughout its life.

Whether scanning, describing or transforming an object, the work required to "ready" a resource for preservation and delivery is thought of as a set of conditions that must be addressed -- *"get descriptive metadata", "validate images"*, etc. Each condition implies a task that must be performed -- a unit of work to that might be performed by humans or programs. More specifically, an instance of WorkDo workflow is defined by metadata in the object that declares the processes needed to complete an end-to-end task. Services provided in the WorkDo infrastructure allow processes to interpret and update the workflow metadata.

Thus WorkDo is not a singular system, but a combination of DOR metadata, REST Services and how programs and applications use them. The workflow data in the object *supports* rather than *controls* the processes that do work, and the interplay of independent parts and shared data nets a scalable and robust set of cooperative processes.

Features of WorkDo:

1. supports both automated (program managed) and manual (human interaction) steps
2. identifies and tracks object lifecycle milestones
3. supports future-dated actions
4. allows prioritization of processing via queue management or task injection
5. fosters reuse of processes and infrastructure services
6. has built in status reporting, timings & visualization of state
7. provides for exceptions identification and management

Two traditional elements of workflow -- conditional actions and branching -- are not directly supported as such in the workflow metadata since each WorkDo process represents a condition that needs resolution. This has not been an issue since "no action", where appropriate, is a legitimate outcome of a process. WorkDo does not operate at a small granular scale, such as that of an individual program's logic, where conditions and branches are more critical constructs.

### Concepts

Three overall concepts make WorkDo work:

1. WorkDo is controlled by metadata that is part of the object being managed
2. Work to be done is manifested as queues that can be acted on as well as counted and tracked *[not that work can't be triggered by messages concerning the arrival of an object in a queue, but the independently reified queue is the point here]*
3. Independent applications and idempotent processes provide scalability and durability through a loosely coupled architecture

## WorkDo is controlled by metadata in the object

The workflow controlling metadata is the key to the WorkDo approach. It leverages data placed in the objects themselves as follows:

- A special form of datastream is added to a DOR object to describe processing requirements for a particular workflow
- The same datastream also tracks progress and state of the object in that workflow
- Thus objects themselves can be queried in DOR about their state and the status of workflow processes
- Workflow related information is indexed (using SOLR) alongside other useful processing information from the object, like collection and selector identity, to manage and report on work
- Simple queries can be used to establish queues and such queues define the work ready for a particular interaction at any given time
- The same queries provide ongoing management information about the flow of objects through the system. They can be exposed as facets in an administrative discovery environment
- Simple REST based interactions are used to identify queues and update state.

In order to support transactional semantics for simultaneous workflow processes, the data for a workflow datastream is physically stored as rows in a table that extends Fedora's relational database. Inside the datastream the workflow information is declared as "externally managed content" using the URL of the REST service that materializes the datastream contents dynamically. This makes it appear to the outside world that the workflow information is part of the object. The workflow services interact directly with the database information, but other parts of the infrastructure can interact with workflow as data in the object, hiding the current database plumbing.

### A WorkDo datastresm

A workflow datastream in each object describes processing requirements and status. For example, there are 9 processes in our workflow for

accessioning a Google Scanned book, which we can use to illustrate much of WorkDo's flexibility:

- **register-object** -- All accessioning workflows start with this, to be able to track an item from the moment it enters our infrastructure. It is also a bootstrapping step for workflow in that the workflow itself is created as part of the object with this step already set to *completed*.
- **descriptive-metadata** -- An external call to our catalog accesses a MARC record in Symphony, converts it to MODS, and inserts it as a datastream into the object.
- **google-convert** and **google-download** -- These two steps work in parallel together, one polling a Google service for the availability of scanned resources and requesting they be converted for download, the other independently watching for the arrival of requested objects. The latter process receives and unwraps the content into the DOR workspace.
- **process-content** -- Here we massage the content as appropriate for our digital library -- interpret the Google METS, perform checksum verification and file validation, and prepare the technical metadata to be kept with the object. This is where the Fedora object gets built.
- **ingest-deposit** and **ingest** -- These two processes manage DORs interaction with the preservation repository. The first prepares the object for preservation and places it into a preservation staging area, using bagit as a transfer mechanism. Ingesting an object into our preservation repository is itself a multi-part process with its own workflow. We would not have an executing process wait indefinitely for this lengthy external processing to complete. The second process represents nothing more than a callback from the ingest process that it has successfully completed.
- **shelve** -- We use a "shelving" metaphor into our Digital Stacks, that part of our infrastructure that supports access and delivery of digital content. This process performs conditional actions, creating indexes as appropriate for discoverable materials and keeping those parts of the object that are required for online delivery.
- **cleanup** -- This is a final housekeeping step to close out accessioning.

The WorkDo workflow datastream is initialized by adding in the XML that declares these steps:

```
<workflow id="googleScannedBookWF">
     <process name="register-object" status="completed" lifecycle="inprocess" attempts="1" />
     <process name="descriptive-metadata" status="waiting" />
     <process name="google-convert" status="e" />
     <process name="google-download" status="waiting" />
     <process name="process-content" status="waiting" />
     <process name="ingest-deposit" status="waiting" />
     <process name="ingest" lifecycle="archived" status="waiting" />
     <process name="shelve" lifecycle="released" status="waiting" />
     <process name="cleanup" lifecycle="accessioned" status="waiting" />
</workflow>
```

Note the first step is already completed by the addition of this workflow, and it also signals that the item is beginning its accessioning process (in other workflows, an item might be registered ahead of being picked up for an accessioning process).

You get a more complete picture of WorkDo features with this same example once it has participated in managing work:

```
<workflow id="googleScannedBookWF" status="active">
     <process name="register-object" status="completed" lifecycle="inprocess" attempts="1" elapsed=
".937"/>
     <process name="descriptive-metadata" status="completed" attempts="1" elapsed="1.114"/>
     <process name="google-convert" status="completed" attempts="1" elapsed="2.294" />
     <process name="google-download" status="exception"
               message="Item for barcode 0339518 not found" attempts="3" />
     <process name="process-content" status="waiting" attempts="0" />
     <process name="ingest-deposit" status="waiting" attempts="0"/>
     <process name="ingest" lifecycle="archived" status="waiting" attempts="0" />
     <process name="shelve" lifecycle="released" status="waiting" attempts="0" />
     <process name="cleanup" lifecycle="accessioned" status="waiting" attempts="0" />
</workflow>
```

Each process has

- a ***name***
- a ***status*** that should start as "waiting" and end with "completed". Though we don't have a use-case yet, in theory there could be intermediate statuses defined by the process for its own convenience.
- a ***datetime*** stamp, showing the datetime of the status shown
- an ***attempts*** count to support automatic replays up to a threshold (for recoverable conditions) and trigger alerts on unresolvable errors.
- (optional) **elapsed** time measure, in seconds, to provide ongoing performance metrics.
- (optional) a **lifecycle** declaration indicating a step satisfies one of several basic basic lifecycle events (see below)
- (optional) ***message*** to accompany exceptions, something short as a label for reports, email, etc, for recognition
- (optional) ***text*** to accompany exceptions, for more complete information, e.g., a stack trace

Overall the workflow datastream supports identifying the tasks that must be done to "complete" the object. An instance of such a workflow is

"active" until completed, then it should be considered "inactive" and will not participate in any further workflow related queries.

## How WorkDo works -- queues, robots and applications

Objects that need to be processed as part of a workflow are organized into **workflow queues** that correspond to each step in the WorkDo definition. Automated steps can be processed by simple scripts called **robots**, while steps that require human interaction are addressed through web based applications.

### Workflow Queues

Each process declared in a WorkDo workflow datastream represents a task to be performed. Each process has a corresponding workflow queue that identifies objects that need to be processed by that process. In the simplest cases, workflow tasks are wholly independent and may be performed in any order. In these cases a queue might be defined as all items for a specific processing step in a workflow whose status is "waiting". But while some tasks can often be done in parallel, there are also common situations where tasks cannot be performed until preceding tasks are completed, like *"you can't archive the object before the page files are processed"* or *"you can't approve the Dissertation before all files are uploaded".* The definition of a WorkDo queue will take such prerequisites into consideration in order to include in the queue only those objects that are both *waiting* and *ready* for processing. We find that a simple scheme of pre-requisite conditions is sufficient without more general facilities for choreographing tasks.

Processes can query the workflow data directly via a REST service, specifying both a target and any prerequisite steps. For example, these calls:

'https://dor-prod.stanford.edu/workflow_queue?workflow=GoogleScannedBookWF&waiting=ingest-deposit&completed=process-content'
'https://dor-prod.stanford.edu/workflow_queue?workflow=GoogleScannedBookWF&waiting=shelve&completed=process-content'

identify objects ready for both ingest and shelving, processes that can run in parallel as soon as the content files are processed. The result of such a call is a simple XML results document identifying items in the queue, e.g.,

```
<objects>
   <object id="druid:tr346yr4493" url="https://dor-prod.stanford.edu/objects/druid:tr346yr4493">
   <object id="druid:jc826sq7352" url="https://dor-prod.stanford.edu/objects/druid:jc826sq7352">
   <object id="druid:mk913nh8271" url="https://dor-prod.stanford.edu/objects/druid:mk913nh8271">
</objects>
```

The API limits queus to references to prior steps in the same workflow. But a queue should be able to be anything that a group of records have in common, plus appropriate filtering. For instance, ETDs can be sent to Google once they have been catalogued, but not if they are under embargo.  Embargo is a condition that can be identified in the object but is not part of the workflow per se. Work is needed to allow such full flexibility of queue definitions.

Future work will reify workflow queues, defining them as objects in the infrastructure. This will allow calls to the queues by name, with the logic of what populates the queues encapsulated in the queue object. For example

'https://dor-prod.stanford.edu/workflows/googleScannedBookWF/shelve'

would correspond to the second sample above. It would produce same results with a different wrapper:

```
<workflowQueue id="shelve">
   <object id="druid:tr346yr4493" url="https://dor-prod.stanford.edu/objects/druid:tr346yr4493">
   <object id="druid:jc826sq7352" url="https://dor-prod.stanford.edu/objects/druid:jc826sq7352">
   <object id="druid:mk913nh8271" url="https://dor-prod.stanford.edu/objects/druid:mk913nh8271">
</workflowQueue>
```

See Workflow queues for more details on this.

WorkDo workflow queues are currently not FIFO -- first in is not necessarily first out. WorkDo queues are a collection of items in a specific state, and it has not been critical to process the queue in the same order as items may have achieved that state. Indeed, other ordering may be more useful in a particular project, such as oldest objects first or other forms of prioritization. We assume that processing by creation date (oldest to newest) would be a likely preferred order, and a queue-defining query  can also sort the entries appropriately if based on accessible data like this in the object.

### Robots!

Automated processes, those that can be performed programmatically, are handled by simple scripts assigned to each step in the workflow. Such scripts are meant to be run independently, each assigned a given task. The image of autonomous robots on an assembly line gave these processes the name **robots**, a term which also distinguishes them from callable services in the infrastructure. A robot may be specialized to a job for a specific workflow or it may be an instance of a more generic script configurable to the task at hand. At Stanford robots are generally written in Ruby, though any scriptware capable of http interactions and XML parsing will do. Suites of robots associated with workflows are developed together as part of a cohesive development.

A typical robot will ...

- perform a task -- simplest robots mainly coordinate reusable infrastructure service calls
- create or update relevant DOR/Fedora objects and datastreams through the DOR/Fedora RESTful service calls
- update workflow process status on the completion of a task

A robot can work directly on a current queue of work, or on one or more object references passed on invocation. The latter capability is useful for testing or to process prioritized work.

Stanford robots are currently queue based and set to run on a regular schedule as they address ongoing work in a project or a unit in the Library. While not the most sophisticated mechanism, it is safe and robust, allowing for simplified monitoring, batching of work, on-demand processing, etc. Some notable characteristics of a robot implementation are:

*Atomic* -- Having more workflow steps with smaller, more focused robots is better than putting too much work into one robot. It keeps the coding simpler and promotes a greater level of reuse. Our early robots have been purpose-built to our original project workflow needs, but are already evolving to be more configurable to similar processing needs in other projects.

*Restartable* -- One critical characteristic of robots is that they must be fully restartable without ill effects to the objects or the data. Ideally they would engage technologies with full rollback-on-error capabilities, but the reality is that they deal with files in a file system as well as objects and datastreams in Fedora. So robots are always coded to check if an object already exists, if the task had already been started, etc., in order to skip over work done (or simply reprocess it).

*Exception handling* -- When a robot encounters a problem with an object or data, or simply can't complete its task for some detectable reason (e.g., an external service is unavailable), it will update the workflow with an exception, providing both a terse message (for human identification) and any relevant supporting data as text. As workflow statuses, such exceptions can be tracked and counted alongside successful processes. In addition, any exception update to a workflow step will increment the *attempts* count. This provides a mechanism to support automatic replays up to a threshold (for recoverable conditions) and to trigger alerts on unresolvable errors.

*Monitoring* --

*Metrics* -- Timestamps for each step provide information about the time it takes for items to progress through a workflow. A robot can also record an elapsed time specific to the processing of each object to provide more detailed analysis of processing performance.

**Item by item processing**

The current WorkDo interactions are based on the queues as described above; a robot will take a queue and work on the items as a group. This means the work for that task is serialized through one process. In general this isn't an issue -- tasks are typically short, queues are not large, and there is a great deal of parallel work being accomplished through the use of multiple robots. But any individual step could become aq bottleneck, and a parallel processing solution for a single task will be needed eventually for high volume work. WorkDo itself could implement its own scheme for doling out items in a queue one at a time, but currently we are considering some simple messaging to parcel out items in a queue to multiple waiting instances of a robot. A technical solution has not yet been specified.

**Future dated workflows?**

Date-based actions are treated simply as delayed work, and a workflow can be created with a future start date. A standard script based robot can be used for processing. Since its work is calendar based it typically need only run once a day. No queues defined by that workflow will include objects whose workflow has not reached its start date.

**Web applications**

Web-based applications generally support tasks that require human input (though any user tool that can interact with the WorkDo api could also do this).

Examples

- ETD submission, final reader verification, Registrar approval, cataloging
- EEMs submission, copyright clearance, cataloging
- Digitization scanning, QA ...

more to say ??

## Workflow Services

There are only three service calls necessary to support WorkDo.

1. Query workflow -- a query for items with a waiting status and any necessary prerequisite completed step yields "queues" of work to be done

```
GET 'https://dor.stanford.edu/workflow/workflow_queue?query'
```

   A future call to reified queues will supplant this for general use, through this call will be used internally to create those queues.

2. Initiate workflow -- This call is used to add a workflow datastream to specified object

```
PUT 'https://dor.stanford.edu/workflow/dor/objects/{druid}/workflows/{workflow}'
```

3. Update workflow - Updates the status of a workflow step

```
PUT 'https://dor.stanford.edu/workflow/dor/objects/{druid}/workflows/{workflow}/{process}'
```

### Multiple Workflows

One workflow usually addresses one end-to-end unit of work. Larger processes can be broken into multiple workflows however, either overlapping or sequential. In our Electronic Thesis and Dissertation (ETD) support, one workflow simply tracks progress during the student submission and approval phase, a wholly human driven process, then a second workflow takes over when more automated accessioning kicks in. In the current design, the last step in the first workflow initiates the second workflow.

There is another sense in which multiple workflows can cooperate. To the accessioning process, ingest into the archives is a process which passes control to another system rather than to a simple task-fullfillment script. Ingest is itself a complex process that has its own workflow to track progress, exceptions, etc. But the details of ingest processing, while key to effective Repository management, are not relevant to accessioning. WorkDo workflow covers a level of details only as granular as desired for tracking and managing one workflow-based process.

### Lifecycle milestones

For our Digital Object Registry, there is a small number of common lifecycle events we want to track apart from the workflows and robots and services that make up distinct processes that can vary from collection to collection:

- When an object is **registered**
- When accessioning starts (object is **inprocess**)
- If an item is **rejected** during provess (e.g., fails copyright clearance)
- When an object has been ingested into the preservation repository (**archived**)
- When an object has been **released** into our Digital Stacks (the online library)
- When initial processing has completed (**accessioned**)
- If any item is **deaccessioned**

These statuses and timestamps provide a basic set of *how-many* and *how-long* metrics across all objects.

The "registered" milestone is implicit in creating a DOR object. We have integrated the setting of the remaining milestones into WorkDo by making them an attribute of work being done rather than tracking them as separate events. If a successful WorkDo process will always indicate a lifecycle milestone, then they can be predeclared in the workflow template as shown above:

```
<workflow id="googleScannedBookWF" status="active">
        :
     <process name="ingest" status="waiting" lifecycle="archived"/>
     <process name="shelve" status="waiting" lifecycle="released" />
     <process name="cleanup" status="waiting" lifecycle="accessioned" />
</workflow>
```

They can also be set on the workflow update call if the milestone is conditional.  For instance, the shelving robot for Electronic Thesis and Dissertations (ETDs) will only "release" items not being embargoed. A later, future dated workflow will release the resource when it is time.

### Admin interface

Current queue status

Planned UI ... show workflow facets

### WorkDo Pros and Cons

This approach was an expedient way to support the needs of several projects underway in the Stanford digital library and is still evolving. It does not have all the capabilities of a fully featured workflow system -- it is associated with a specific repository and its objects and tasks could not coordinate work across environments. It is a comfortable fit for a certain sized "lifecycle" unit of work and would not be suited for the orchestration

of many small processes. It does not support complex and highly dynamic workflows, though it is young and will evolve as needed.

But WorkDo's strengths outweigh these issues:

- The integration of the workflow data with the object has been effective in satisfying the informational and processing needs of our digital resource management
- Lightweight? WorkDo does not require external rule or state engines, messaging, or separate process orchestration software
- It was quick and easy to implement
- It promotes a high level of reuse through lightweight scripts and simple application logic build on generalized infrastructure and services [Reification of Queues, Queries & Robots leads to reusability]
- State is captured within each object and tracked across the object's entire lifetime, providing a complete and durable audit trail
- The same in-object data, when coupled with SOLR indexing, provides flexible and transparent discovery, filtering and statistics capabilities through an administrative interface
- Its youth is an advantage, so while it needs to evolve, we can rapidly shape this solution specifically and only as needed
- We got robots!

### Possibilities & Plans for Extension

Three queue related future developments:

- Item-by-item processing of items in a queue -- this will enable multiple instances of a robot process a given step in parallel as a scaling option that we might been in the future.
- Integration of messaging to signal the presence of a new item in the queue. This would be one approach to supporting item-by-item processing.
- Built-in ordering of queues

---

*Example -- Electronic Theses and Dissertations (ETDs) workflows*

A workflow can reflect critical activity for management reasons even if the process is wholly defined through an external application. The initial WorkDo description for the Stanford ETD application merely tracks progress milestones in the submission process:

```
<workflow id="etdSubmitWF" status="active">
   <process name="register-object" status="completed" attempts="1" datetime="2009.012.06..." />
   <process name="submit" status="waiting" ... />
   <process name="final-reader-approval" status="waiting" />
   <process name="registrar-approval" status="waiting" ... />
   <process name="initiate-accession" status="waiting" ... />
</workflow>
```

It then initiates the technical Services accessioning workflow as its last step:

```
<workflow id="etdProcessWF" status="active">
   <process name="create-catalog-record" status="waiting" />
   <process name="catalog" status="waiting" />
   <process name="descriptive-metadata" status="waiting" />
   <process name="ingest-deposit" status="waiting" />
   <process name="ingest" status="waiting" lifecycle="archived" />
   <process name="shelve" status="waiting" />
   <process name="cleanup" status="waiting" lifecycle="accessioned" />
</workflow>
```

Note the similarity to the Google scanned book workflow earlier in this document. Several of the steps are the same, with two differences of note. First, places the descriptive metadata in the DOR object is preceded by seeding the ILS (Symphony) with a stub record ("create-catalog-record"), then waiting for manual cataloging to finish in that system ("catalog"). Second, the "shelve" process is not assumed to satisfy the "released" lifecycle milestone since some dissertations will be embargoed.

In the implementation of this second workflow, the two steps relating to ingest and the cleanup step are reusable robots. The shelving step, while sharing a name with the parallel Google step, may be distinct enough to warrant a separate robot. But it will no doubt be refactored together with other shelving robots as the Digital Stacks shelving strategy evolves.