Ingest Tool

> This tool has been superseded by the Object Ingest Tool.

This is the tool that will ingest/update items in the Fedora repository. The intention is to provide an interface that multiple cataloging tools could use to create and manage fedora-based collections. Please see Ingest Tool Comparisons to compare the goals of this Ingest Tool to some other similar projects by other organizations.

Important note: For purposes of the ingest tool, the term "collection" means a group of digital objects that share a common theme **and** that subscribe to the same content model.

The process involves building the prerequisite data, then calling the IngestTool with the proper parameters.

(This page explains the new ingest tool setup: Object Ingest Tool)

For information about the entire digitization/ingest process, see Ingest Tool Workflow.

Names

If we ever clean this tool up enough for general release, we will need a catchy name:

- Fedora Ingest, Composition, and Update Service (FICUS)
- General Automated Ingest and Update Service (GAIUS)

Use scenarios

See the Use Scenarios page.

Setting up a new ingest

1. Place Source Metadata in an accessible location (on the same machine as the ingest tool, or somewhere accessible via URL).
2. Place media files (master and derivative) in an accessible location. If you don't already have derivative files, see the Derivative Creation p rocess.
3. Define the Control data – The ingest process is controlled by XML defined by a "collection configuration" file.
4. Edit the JSP page to allow easy running of the ingest process – you'll probably have to run it multiple times.

Calling the Ingest Tool

The ingest tool was originally designed as a servlet. However, the preferred way to run it is from the command line, using the fedora-ingest.sh script in the root of the webapp.

Old webapp interface (deprecated)

Currently there is a very simple struts webapp (.../infrastructure/) in the workspace that shows how to interact with the servlet.

The ingest tool servlet has 4 main operations:

- Request ID – this generates a unique ingestID and accessCode for use in a collectionConfiguration file.
- Ingest Data – this performs an actual ingestion.
- Request Status – the reports on the state of an ingestion.
- Stop Ingest – this stops (forcably, if needed) an ingestion.

The servlet parameters are:

- action (requestID, requestStatus, ingestData, stopIngest) [required]
- location (url, localfs) [optional]
- url [optional]
- path [optional]
- id [optional]
- accessCode [optional]

- force [optional]

Request ID

Required: action=requestID
Returns: XML

```
<requestID>
    <id>6</id>

<accessCode>96DE88341F3B64385107288B71FBFF9D6CB1B3DE07601E46086FF7894A2E53
5</accessCode>
</requestID>
```

Ingest Data

Required: action=ingestData, location, url (if location is url) or path (if location is localfs), id, and accessCode
Returns: nothing useful at this time. It should probably return an XML page with some state info, or maybe an HTML page with some links to kill the ingest, or get it's state... I really haven't put too much thought into this yet.

Request Status

Required: action=ingestData, id, and accessCode
Returns: XML

```
<ingestStatus>
    <state>Ingestion complete</state>
</ingestStatus>
```

It should probably also have a numeric indicator for programmatic building of the edu.indiana.dlib.fedora.server.ingestTool.types.IngestState.java object (using the static .FromInt method).

Stop Ingest

Required: action=ingestData, id, and accessCode
Optional: force – true or 1 is treated as true, anything else is treated as false, the default is false.
Returns: same as request status (above)

If force is false, it sets the state in the DB to IngestState.TERMINATE which indicates to the worker to stop after the item it is working on. However, it may be a dead process, in which case you have to indicate to the DB that it's okay to re-run the ingest. Setting force to true will flip the status in the DB to IngestEvent.STOPPED indicating it's okay to run another ingest. This could cause a problem where 2 ingest processes are concurrently processing the same ingestID at the same time. The errors in such a case are unknown, but probably ugly. Be careful with force.

Ingesting a SIP and reproducing an equivalent DIP

When we receive a SIP from another DLP application, it can be ingested normally, because the contents will always fit our requirements.

When we receive a SIP from elsewhere, it may not meet our requirements. In this case, we will transform the object into our format to meet our needs, but retain information necessary to re-create the original object. The metadata (typically METS) associated with the SIP should be stored in a datastream (called METS_ORIGINAL). Files with non-conforming names should be given a new (automatically generated) name, and the mapping between the old and new filenames stored (where? formatted how?).

When we need to return an object, it should be possible to create both a DIP in our format (from the normal datastreams) as well as a DIP in "original" format (from the additional datastreams).

Open Question: How do we handle changes that happen to the object between ingestion and dissemination? Metadata-only changes could be ignored when creating an "original" DIP. But what about migrations to new media formats? Should we return the new media files with the original names? Retain and return the original media files?

Ingest Tool Technical Details

The main servlet is edu.indiana.dlib.fedora.server.IngestTool. It spawns the process edu.indiana.dlib.fedora.server.ingestTool.IngestToolWorker.

The ingest tool is a Java Servlet on the surface, but the actual ingestion is done by a backgrounded java process. Both the servlet and the app are configured through a shared configuration file.

To handle the problems that will inevitably occur during ingestion, the ingest tool tracks the status of all ingests in a database.

The ingest tool is in need of some refactoring.

Altering existing objects

Sometimes, we might need to update the properties (e.g. image, metadata, disseminator binding, etc.) of some objects that have already been ingested by Fedora. For small updates, the Fedora administration client might be enough but for large updates, we might need an Update tool either as part of the existing Ingest tool or separately. So, some questions to consider for this kind of functionality:

- Should "update" be implemented
    - within the Ingest tool as envisioned before, has any effort been put into this already, or,
    - is it more feasible to implement it as a separate tool using the parts from the ingest tool?
- When and how such updates are needed?
- What is to be updated, image datastreams, METS metadata records, disseminator and behavior bindings?
- Would such an update break other things such as requiring re-bindings?
- What kind of user interface would be appropriate, similar to Ingest with CollectionConfiguration, Mods, etc.?

Open Questions

- Can we make use of the 7train METS generator?
- Can we integrate with the Directory Ingest Service?
- How best to integrate with Xubmit?
- How do we properly ingest paged objects? The metadata will typically be at the item level, while images will be at the page level.