DIGITAL LIBRARIES
University Libraries
UNT

UNT Libraries SIP-to-AIP Conversion Workflow

Date: October 2015

Version: 1.0

Contributors:

Mark Phillips Assistant Dean for Digital Libraries
Hannah Tarver Department Head, Digital Projects Unit
Ana Krahmer Supervisor, Digital Newspaper Unit
Daniel Alemneh Supervisor, Digital Curation Unit
Laura Waugh Repository Librarian for Scholarly Works

UNT Libraries SIP-to-AIP Conversion Workflow

Introduction

The UNT Libraries has a formal definition of the required features and elements for a Submission Information Package (SIP), documented in the UNT Libraries OAIS Information Package Specification. That document, while describing what should be present in a valid Submission Information Package (SIP) and Archival Information Package (AIP), does not cover the process used for converting a SIP to an AIP. The goal of this document is to establish the process and document the workflow for the conversion of a SIP to an AIP for the UNT Libraries' Digital Collections.

Ingest Dropbox

The conversion process starts when a SIP is submitted to one of the different ingest dropboxes configured for the ingest process. The difference in the dropboxes is in how the names (ARK identifiers) for objects are assigned. One dropbox per namespace is assigned. We currently create identifiers under the metapth, metadc and metarkv namespace.

Dropboxes define the steps involved in the ingest process of our digital content. They are responsible for creating the Archival Information Packages (AIPs) and Dissemination Information Packages (DIPs). Locally, DIPs are called Access Content Packages or ACPs for our Coda repository and Aubrey access systems. A dropbox encapsulates a series of steps that are executed in sequential order on submitted digital objects. These steps are represented by folders in the dropbox that hold the input, output, and errors for each of the steps. The management python scripts such as makeAIP.py and makeACP.py are used to convert a SIP to an AIP and an AIP to a DIP/ACP.

A dropbox is organized like this:

```
dropbox/
|-- 0.Staging/
|-- 1.ToAIP/
|-- 2.ToAIP-Error/
|-- 3.ToACP/
|-- 4.ToACP-Error/
|-- 5.ToArchive/
|-- 6.ToAubrey/
|-- 7.ToAubreySorted/
|-- 8.ToAubreySorted-Error/
|-- dropbox_config.py
|-- makeACP.py
|-- makeACPSort.py
|-- makeAIP.py
`-- moveToCODA.sh
```

Verification steps

The verification steps that makeAIP.py execute before processing a submitted SIP include the following:

1. Check to see that the proper utilities and versions are installed on the ingest server.
2. Check that the appropriate number server is online (number server provides object names)
3. Check the Namaste tag in the bag to make sure it is a SIP bag
4. Fully validate the bag based on the BagIt specification
5. Verify the conformance to UNTL-FileSets for each manifestation in the SIP.

Objects to ingest are loaded by an ingest operator into the 1.ToAIP folder and then the makeAIP.py script is exectuted, which walks through the five steps listed above to determine if a supplied SIP can be converted into a UNTL-AIP.

If makeAIP.py fails on check 1 or 2 in the above list, the whole process stops. If these two steps pass, makeAIP.py will begin multiple sub-processes that allow for parallel processing of digital objects and therefore take advantage of available processors on the ingest server. If there is an error in one of the 3, 4 or 5 checks in the list above, the process moves the object to the 2.ToAIP-Error folder with log file describing the error encountered and will continue to process the next object in the queue. When makeAIP.py is finished running, it will let the operator know how many SIPs were processed, the number that were successfully converted into AIPs, and the number that failed.

SIP to AIP Conversion

Once the makeAIP.py script verifies that the submitted SIP is valid and well formed, it executes a series of steps to complete the AIP creation. The steps include the following:

1. Retrieve instructions from coda_process.py
2. If magicknumbers is set to true, verify that files are valid magicknumbers and use these to create sequential ordering for the resulting METS document.
3. If pageNumbers.txt is present use these values as order labels in the resulting METS document.
4. Identify primary bitstream of fileSet
5. Process all files and and create a PREMIS Object XML file, JHOVE output stream and UNIX File output.
6. Associate optional order labels, or annotations with fileSets.
7. Generate a UNTL METS Archival Information Package Profile XML document conforming to the METS Profile - http://www.loc.gov/standards/mets/profiles/00000045.xml

If all of these steps are completed successfully then an AIP is created in the 3.ToACP folder and its resulting BagIt bag is updated with the newly created JHOVE and METS files. If there were any errors encountered (usually caused by malformed filesets or incorrectly formatted magicknumbers) it is moved to the 2.ToAIP-Error folder and the issue is logged with the object so it can be fixed and reprocessed in the future.

Example SIP and AIP

Input SIP

```
1999.001.001/
|-- 0=UNTL_SIP_1.0
|-- bag-info.txt
|-- bagit.txt
|-- coda_directives.py
|-- data/
|   |-- 01_tif/
|   |   |-- 1999.001.001_01.tif
|   |   `-- 1999.001.001_02.tif
|   `-- metadata.xml
`-- manifest-md5.txt
```

Resulting AIP

```
metapth1234/
|-- 0=UNTL_AIP_1.0
|-- bag-info.txt
|-- bagit.txt
|-- coda_directives.py
|-- data/
|   |-- data/
|   |   `-- 01_tif/
|   |       |-- 1999.001.001_01.tif
|   |       `-- 1999.001.001_02.tif
|   |-- metadata/
|   |   |-- 17711fdb-2e25-4566-bf3f-daa172a12190.jhove.xml
|   |   `-- 9639acae-397a-4c90-8851-52b6f04c4d8d.jhove.xml
|   |-- metadata.xml
|   `-- metapth1234.aip.mets.xml
`-- manifest-md5.txt
```