Electronic Records Modules
Electronic Records Committee
Congressional Papers Roundtable
Society of American Archivists

File Fixity Tools

Elisabeth F. Butler

Senate Historical Office

Elisabeth_Butler@sec.senate.gov

Katie Delascenserie

Senate Committee on Homeland Security and Governmental Affairs

Katie_Delacenserie@hsgac.senate.gov

Date Published: December 15, 2016

Module#: ERCM009

File Fixity Tools

Overview and Rationale

One of the key goals of digital preservation is to ensure that objects have not changed over time, thereby retaining their authenticity. This process is known as fixity. Fixity can be defined as "Information used to verify whether an object has been altered in an undocumented or unauthorized way."[1] Fixity information is created by generating checksums and cryptographic hashes (MD5, SHA1, and SHA256). This is important because fixity indicates the existence of data change or corruption and therefore ensures the validity of data when coming to the repository, when moving from one medium to another, and when in long-term storage (whether on servers, drives, or a dedicated system). Some things to keep in mind about fixity:

- **Where to Store/Reference**: Record fixity information with the object's/accession's metadata file; for checks that are run at given intervals, you do not want to touch the object/accession metadata records but run the fixity information in databases and logs.
- **Storage Media:** Doing fixity checks typically increases the usage of the media and of the mechanical devices that read and handle the media, which are factors contributing to the overall reliability of the media.
- **Throughput:** Your rate of fixity checking is going to be dependent on how quickly you can run the checks, the complexity of your chosen fixity instrument, and how much of your resources (e.g., CPU, memory, bandwidth) can be used for this workflow. This becomes a choke point as the amount of digital content increases but the infrastructure to perform the checks stays the same. In a situation like this, the fixity checking activities can adversely affect other important functions like delivery of the content to users.
- **Number of Files or Objects and Sizes of Files or Objects:** Different resource requirements emerge as the scale of digital files and objects increases both in number and size.
- **Redundancy Level in Content and Process:** Depending on your system design, you may want to have different practices for checking fixity on redundant copies. For example, if the fixity is already being checked for the primary and secondary copies on a regular basis, you might decide that you don't need to check the tertiary copies as often. Similarly, you may have files that serve as preservation masters and files that serve as derivatives or access copies in which case it likely makes sense to have different practices for the files that serve different roles.
- **Assurance from Third Party, like a Cloud Storage Provider:** Instead of running your own checks, you may be in a situation to trust the claims of a third party about their

[1] PREMIS Data Dictionary: http://www.loc.gov/standards/premis/v2/premis-dd- 2- 2.pdf

persistent checking of the copies they maintain in their system. Though understanding what the cloud provider is supporting and how often and detailed fixity checks are done is important to understand.

- **Covered at the File System or Object System Level:** If the file or object system itself performs frequent checks at the block level you may not need to be particularly concerned with bit rot as a threat to fixity, in which case frequent checks of files may be unnecessary. For example, file systems like ZFS maintain and check block level fixity information. [2]

This module focuses on describing and comparing fixity tools currently available. They are File Verifier++, RHash, Bagger, File Checksum Integrity Verifier (FCIV), and DROID. These tools were chosen because they are accepted by the archival community, exhibit preservation best practices, and were recently evaluated by the Senate Historical Office's National Digital Stewardship Resident (2015-2016) John Caldwell. Other information is derived from the Library of Congress' *The Signal* blog and from the source location of the tools.

Procedural Assumptions

- The repository does not have a digital document/object storage system or digital repository that has built-in fixity
- Most tools are designed for digital content that has been accessioned or "bagged" before fixity creation
- Ability to use and manipulate XML databases and Excel/.CSV spreadsheets
- Ability to download applications onto local computers and use command line operations

Hardware and Software Requirements

All of the tools are open source, actively supported, and platform neutral (with the exception of FCIV which is Windows-based). These tools also can also be applied to establishing the fixity of audio/visual files.

[2] "Checking Your Digital Content" National Digital Stewardship Alliance, 2014. http://memory.loc.gov/ammem/lccat/ontheway.html

Workflow/Comparison

	Source	**Interface**	**Use**	**Recommended Use**	**Limitation**
FileVerifier++	SourceForge; Programming Unlimited	GUI	Specifically generates checksums using multiple algorithms (hashes); exports to .csv file, plain text file, or specific file type	Point of ingest, validating for records stored for long periods of time	Can't use to validate if the original checksum was not done by FileVerifier++
RHash	SourceForge	Command Line	Specifically generates and validates checksums; exports to any file format of choice	Point of ingest, transfer of records among different media, and validating records periodically	Must print checksums to an output file in order to validate; may not generate hash values for files with long file paths/names (Windows)
File Checksum Integrity Verifier (FCIV)	Microsoft	Command Line	Specifically generates and validates checksums in Windows; exports to a text file or XML database	Point of ingest, moving records among different media, and validating records periodically	May not generate hash values for files with long file paths/names; will only validate from the XML database

Bagger	Library of Congress; GitHub	GUI version of BagIt	Creates hash values when packaging content for long term storage, tests completeness of package, makes checksums for validation over time	Point of ingest and validation periodically; for archives which use BagIt	Default hash value is MD5; may not successfully "bag" large data sets—use BagIt on the command line or the "Bag in Place" operation
DROID	National Archives; GitHub	GUI	Generates format identification and checksums (workflow can be found in the module ERCM001)	Point of ingest	Hashing has to be turned on and hash value chosen before it will create a checksum; no native ability to validate over time

Examples

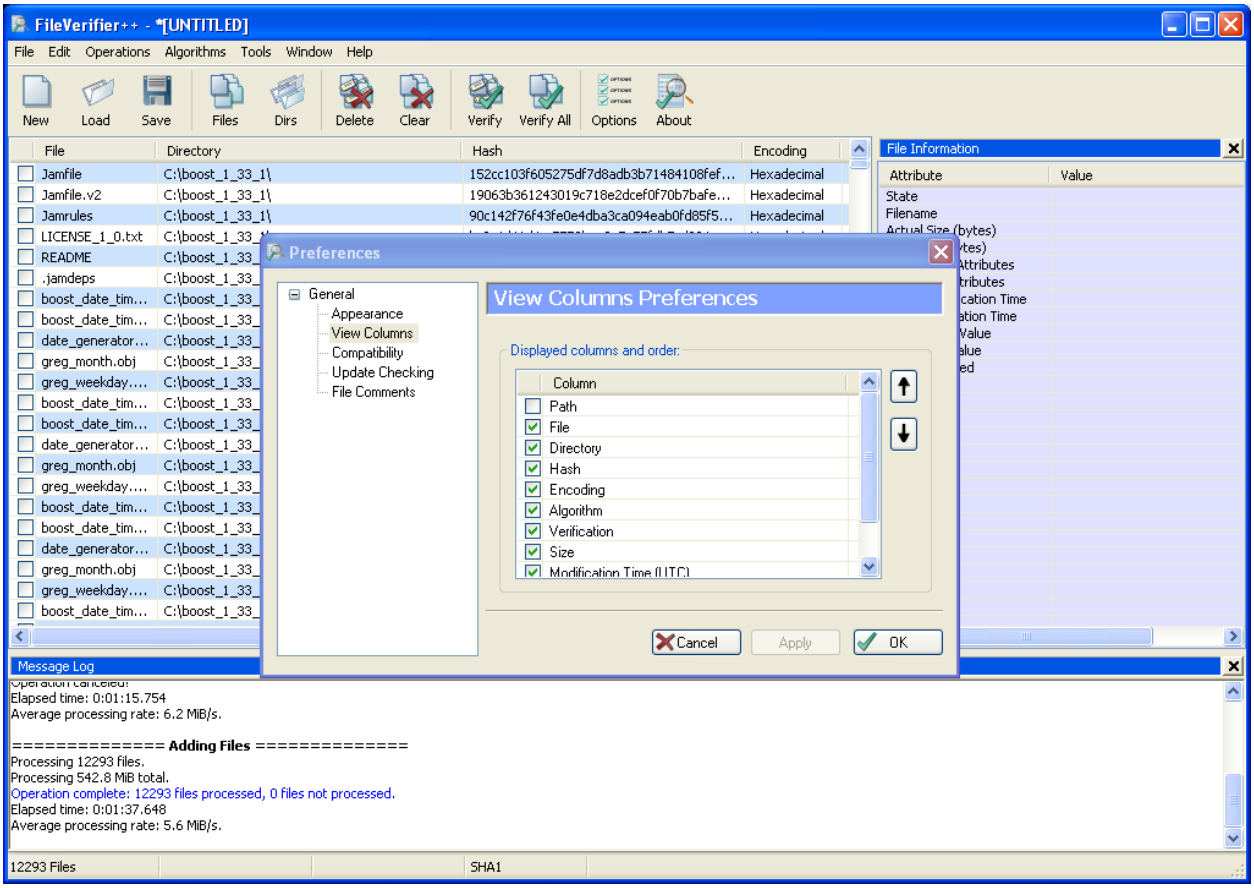
RHash (command line instructions)

- To generate hash values: *rhash -r [-M or --sha256 (2 dashes)] <file location> -v -l <log output location and file name>.txt -o <hash output location and file name>.txt*
- To validate hash values: *rhash –c [or --check] <hash output location and file name>.txt*
    - Validation results will display in the terminal.
- To print validation results: *rhash –c [or --check] <hash output location and file name>.txt -o <hash output location and file name>.txt*

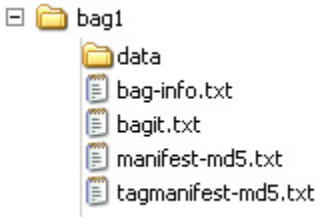
FCIV (command line instructions)

- To print to a text file: *fciv.exe –add [directory path for files to analyze] –r –md5 {or –sha 1} > [file path\Directory Name_Fixity Algorithm_Date (as yyyymmdd)].txt*
- To create an XML database: *fciv.exe –add [directory path for files to analyze] –r –md5 {or –sha 1} –xml [file path\Directory Name_Fixity Algorithm_Date (as yyyymmdd)]db.xml*
- To verify fixity values from an XML database: *fciv.exe –v [directory path for files to analyze] –md5 {or –sha 1} –xml [file path\Directory Name_Fixity Algorithm_Date (as yyyymmdd)]db.xml*

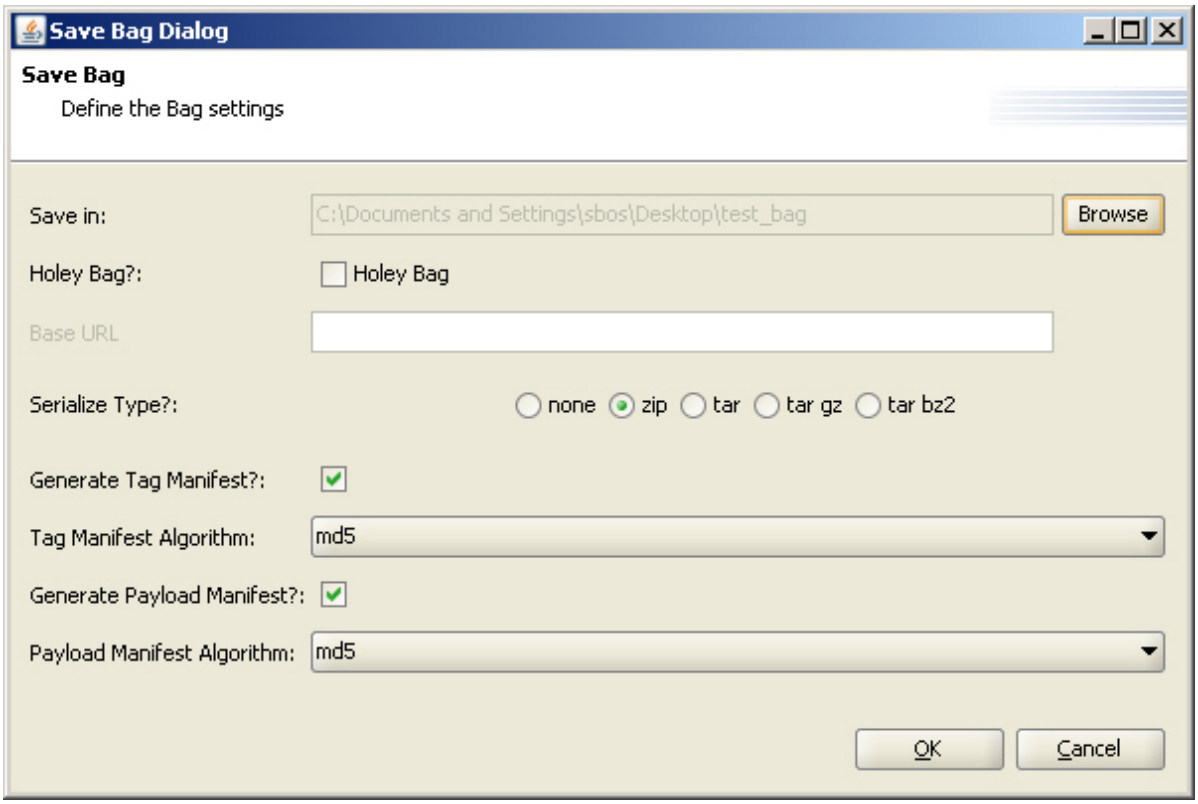
FileVerifier++ (screenshot)

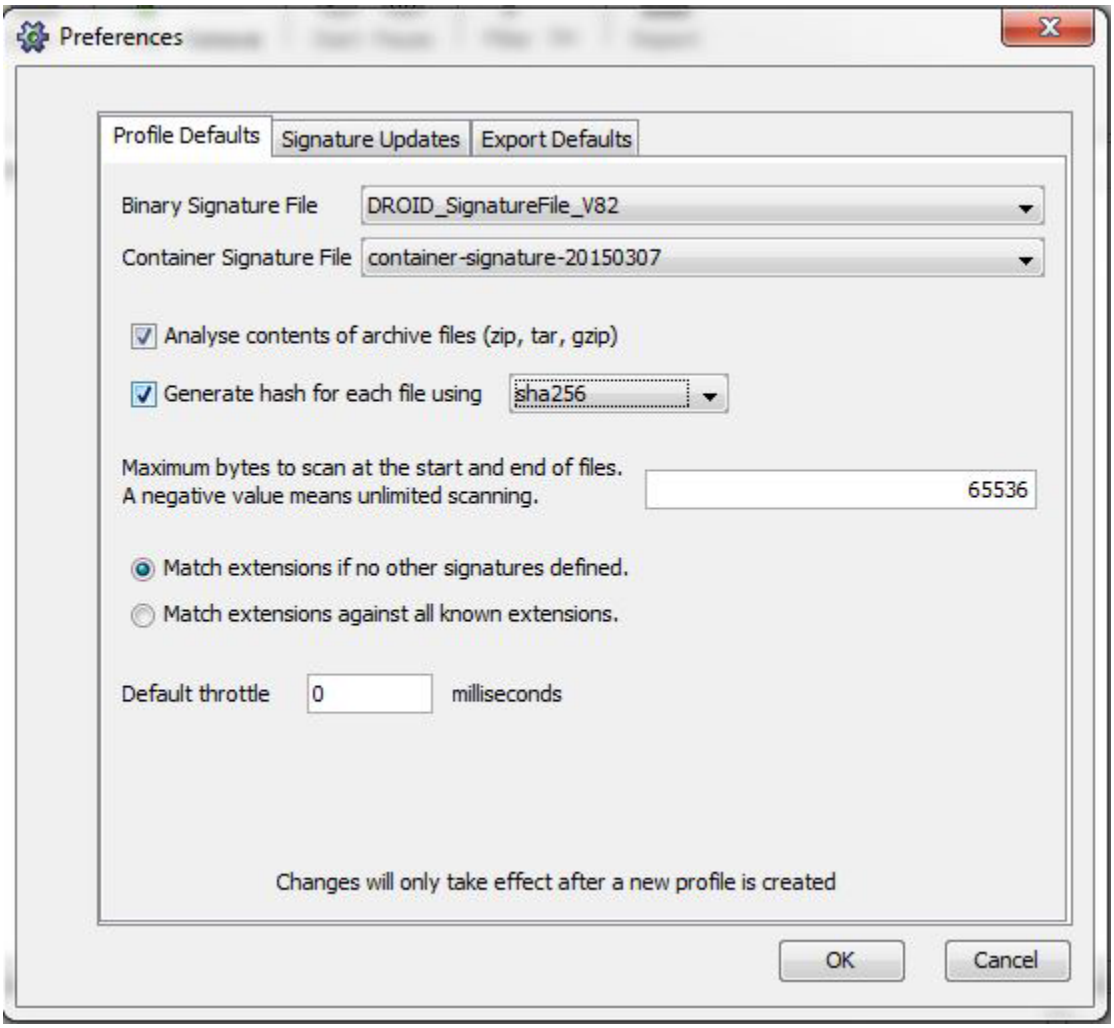


Bagger



Completed "bag." The manifest.md5.txt is a text file that is an inventory of the files along with their checksums and is automatically generated once the "bag" is complete.



Validating a "bag" takes place towards the end of its creation. It can do three things.
1. It checks whether all the files in the manifest are still there.
2. It checks whether any files have been added to the bag since the manifest was created.
3. It checks the checksums of each file to determine whether the file itself has been modified.
Note: Windows machines create hidden thumbnail files for image directories. You will need to turn off this functionality on the validating machine in order to run the validation process.

DROID (screenshot of hash preferences)