Checking Your Digital Content: How, What and When to Check Fixity?

How do I know what I have and the file/object is not corrupted, changed or altered? Further how can I prove that I know what I have? How can I be confident that the content I am providing is in good condition, complete, or reasonably complete? How do I verify that a file/object has not changed over time or during transfer processes? In digital preservation, a key part of answering these questions comes through establishing and checking the "fixity" or stability of digital content. At this point, many in the preservation community know they should be checking the fixity of their content, but how, when and how often? This document aims to help stewards answer these questions in a way that makes sense for their organization based on their needs and resources.

Defining Fixity and Fixity Information

Fixity in this context is the property of a digital file or object being fixed or unchanged. In this sense, it is synonymous with bit-level integrity. *Fixity information* is information that offers evidence that one set of bits is identical to another. The PREMIS data dictionary (named after the Preservation Metadata: Implementation Strategies Working Group) defines fixity information as "information used to verify whether an object has been altered in an undocumented or unauthorized way." Collecting and maintaining fixity information over time can help to support any of the following activities. The most widely used tools for creating fixity information are checksums (like CRCs) and cryptographic hashes (like MD5, and various SHA algorithms) but there are other methods such as expected file size and file count that provide basic fixity information. The instruments for creating fixity information are explained later in this document, but before getting to those, it is worth pausing to explain the diverse range of reasons to collect, check, maintain, and verify fixity information.

Reasons to Collect, Check, Maintain, and Verify Fixity Information

Fixity information helps answer three primary questions:

- **Do you have the data/files you expected?** When fixity information is provided with objects upfront it can be used to validate you have received what was intended for the collection
- **Is the data corrupted or altered from what you expected?** Once you have generated fixity information for files or objects comparing that information with future fixity check information will tell you if a file has changed or been corrupted.
- **Can you prove you have the data/files you expected and it is not corrupt or altered?** By providing fixity information alongside content, you enable your users to verify that what they have is identical to what you say it should be. This supports assertions about the authenticity and trustworthiness of digital objects

Fixity information has other uses and benefits as well.

- Support the repair of corrupt or otherwise altered digital files or objects: If you have multiple copies of digital objects and you have stored fixity information to refer to you can refer to that information to verify which copy is correct and then replace the corrupted digital files or objects.
- Monitor hardware degradation: If checks against fixity information for a set of objects begin to rapidly degrade, it can offer advanced warning of failures of the media you are storing the objects on.
- To be confident when providing someone with a copy of an item, or a segment or portion of an item that the file or object is correct: By comparing fixity information on a file or object against recorded fixity information for it you can be sure that what you are giving them is exactly what you assert it should be.
- Permit an update to a portion of a content file or object while being able to determine that the "other" portion is unchanged: If you maintain very granular fixity information you can use comparisons and revisions of that information to be sure that other parts of it are not affected.
- Meet requirements or best practice guidelines like ISO 16363/TRAC and the NDSA Levels of Digital Preservation
- Support the monitoring of production or digitization processes: Generating and checking fixity information across these processes provides a means to monitor content integrity across the process.
- Document provenance and chain of custody: By maintaining fixity information provided with the content and logs documenting fixity over time along with any necessary file repairs, you can provide evidence of the integrity of content across the time it has been under stewardship.
- Help diagnose possible systemic or human error in the lifecycle management of preserved content**:** Regularly computing fixity information and comparing it with initial fixity information provides a continual documentation of changes or damage to files. As such, the process of checking fixity works to help surface issues relate to operator error or problems in system glitches.

General Approaches to Fixity Check Frequency

The following list describes different approaches to checking the fixity of digital content. These include a wide range of approaches, including those that are built into storage systems, that can be automated through scripts and applications and that might involve manual workflows.

- **Generating/Checking Fixity Information on Ingest:** When you bring content under stewardship, it is important to check the fixity of content transferred to you. Whenever possible, it’s ideal to encourage content providers or producers to submit fixity information along with content objects. And what that might look like in cases where fixity information isn’t provided as part of the transfer, you should create fixity information once you have received the materials because any future checking of your content is going to require that you have these initial values.

- **Checking Fixity Information on Transfer**: Transferring data from one storage system to another is a potential point at which your digital content could be damaged. As such, it is critical to check the fixity of your content whenever it is being moved. Assuming you have additional copies of your data, you can use any failures as an opportunity to recover/repair by checking any files or objects that show up as not matching their fixity values against those other copies.
- **Checking Fixity at Regular Intervals:** In addition to fixity checking before and after transfer, collections of digital files and objects should be checked on a regular basis. There are a range of systems and approaches focused on checking the fixity values of all objects at regular intervals. This could be monthly, quarterly, or yearly for example. By checking more frequently it is far more likely that you can detect errors, and (assuming you have additional copies) you can repair errors from those additional copies.
- **Building Fixity checking into Object Systems:** Most object systems using REST (Representational State Transfer) interfaces have fixity built into the system so that data is regularly checked. Systems with this kind of architecture may require less checking as the checks are built into the architecture.
- **Using Fixity Checking in Archive Systems (Hierarchical Storage Management):** Most archive systems such as HSMs support per-file checksums on tape and, with new tape technologies, per-block checksums can be validated on tape systems without reading the data back to the host.
- **Building Fixity Checking into File Systems:** File systems like ZFS compute block level checksums (using SHA256) on a regular basis with the goal of protecting against bit rot, but ZFS only validates data when a block is read, so it is impossible to know if the media or channel are having issues until the data is read.
- **Checking Fixity for Process Monitoring:** For certain classes of content--there are notable examples in the realm of digital audio and video--checksums (or the equivalent) support process or production monitoring. For example, some organizations have used checksums to help determine that the underlying picture and sound data from a DV videotape had been successfully extracted when migrating that content to a file.
- **Checking Fixity on a Segment or Portion of a File** when that segment is to be provided to an end user, or when other portions of the file are to be changed. Examples of this implementation include checksuming the encoded audio data within an audio file or individual frame-level checksums for video files.

Considerations for Fixity Check Frequency

You might now ask yourself, why doesn't everybody just run fixity checks and compare fixity information for all their content at fixed intervals? The answers to that question are tied up in resource constraints and, in some cases, the lack of fixity support.

- **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 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 which is automatically used to support the repair of data that is damaged through silent data corruption.

How Do Common Fixity Instruments Compare Against One Another?

The table below presents some basic information on a range of instruments for creating and checking fixity information. Each instrument takes different levels of effort/resources to use and results in varying degrees of detail and quality of the fixity information they generate.

Fixity Instrument	Definition	Level of Effort and Return on Investment
Expected File Size	File size that differ from the expected can be an indicator of problems, for example highlighting zero byte files	**Low level of effort and low level detail**. File size is auto-generated technical metadata and can be viewed in Windows Explorer or other common tools.
Expected File Count	File count that differ from the expected can be an indicator that files are either added or dropped from the package.	**Low level of effort and low level detail.** File count is often auto-generated technical metadata and can be viewed in Windows Explorer or other common tools.
CRC	Typically checking network error	**Low level of effort and moderate level of detail**. CRC function values, which are variable but typically 32 or 64 bit, are relatively easy to implement and analyze.
MD5	Cryptographic hash function	**Moderate level of effort and high level of detail.** CPU and processing requirements to compute the hash values are low to moderate depending on the size of the file. The output size of the hash value is the lowest of the cryptographic hash values at 128 bits.
SHA1	Cryptographic hash function	**Moderate level of effort, high level of detail and added security assurance.** Due to its higher 160-bit output hash value, SHA-1 requires more relative time to compute for a given number of processing cycles CPU and processing time than MD5.
SHA256	More secure cryptographic hash function	**High level of effort and very high level of detail, and added security assurance**. With an output hash value of 256 bits, SHA-256 requires more relative time to compute for a given number of processing cycles CPU and processing time than SHA-1.

Verifying auto-generated metadata such as expected file counts and file size can provide a simple and easy to access first line of inquiry into the health and well-being of the digital objects, both as individual units and as an aggregate group or package. CRCs are typically used for verifying network transfers and have somewhat limited effectiveness

for the high-speed data transfers available on today's or future networks, though networks protocols such as ethernet, SAS, fiber channel and others have low level CRCs built into the hardware protocol. The ANSI T10 PI standard is used in addition the standard channel CRCs for an added level of protection and detection.

CRCs are useful for quickly generating fixity information and are used often on the intra-file level. However, because MD5, SHA1, and SHA256 are so significantly superior, whenever resources permit it may be better to rely on any of these cryptographic hash functions for whole file and object level fixity documentation. As noted above, MD5, SHA1 and SHA256 are cryptographic hash functions with different sizes of checksums and with increasing levels of security. In many cases, for data fixity purposes, either MD5 or SHA1 are often more useful than SHA256 due to the higher compute time and CPU requirements for the latter. With increased levels of security comes increased time and resources to compute, so depending on the amount of data in a collection and resources at hand each has a place in different fixity checking workflows.

Where to Store and Reference Fixity Information

In different situations, it makes sense to store fixity information in different places. Here are some examples of places you might store fixity data toward different objectives.

- **In Object Metadata Records:** In many cases, you will want to record some file or object fixity information wherever you store and manage the metadata records. These might be actually stored in discrete files or databases. This is particularly useful for maintaining originally submitted or generated fixity information as part of the long-term object metadata.

- **In databases and logs:** For checks you run at given intervals you likely do not want to be touching your object metadata records all the time. In this case, it makes sense to keep running fixity information in databases and logs that you use and can return to when needed.

- **Alongside content:** It's often ideal to have fixity information right alongside the content itself. For example, the BagIt specification includes a hash value for the bagged content right alongside the content itself. Similarly, some workflows involve creating .md5 files , which are simply text files with the md5 hash, named identical to the file it refers to, but with an additional .md5 extension.

- **In the files themselves:** When a checksum is for a portion of a file, it may make sense to store the information directly in the file. Note, this only makes sense when storing sub-file fixity information within a file. Adding this information into a file will result in that file as a whole generating different fixity information.

Further Reading

- Hashing Out Digital Trust, Kate Zward: http://blogs.loc.gov/digitalpreservation/2011/11/hashing-out-digital-trust/
- Fixity Checks: Checksums, Message Digests and Digital Signatures, Audrey Novak http://www.library.yale.edu/iac/DPC/AN_DPC_FixityChecksFinal11.pdf
- The NDSA Levels of Digital Preservation: An Explanation and Uses, Megan Phillips, Jefferson Bailey, Andrea Goethals, Trevor Owens http://www.digitalpreservation.gov/ndsa/working_groups/documents/NDSA_Levels_Archiving_2013.pdf
- Authenticity of Electronic Federal Government Publications, Kate Zwaard and Lisa LaPlant : http://www.gpo.gov/pdfs/authentication/authenticationwhitepaper2011.pdf
- Reconsidering the Checksum for Audiovisual Preservation, Dave Rice: http://dericed.com/papers/reconsidering-the-checksum-for-audiovisual-preservation/

Relevant Standards and Specifications

- PREMIS data dictionary http://www.loc.gov/standards/premis/v2/premis-dd-2-2.pdf
- PREMIS Conformance http://www.loc.gov/standards/premis/premisConformance_v4.pdf
- BagIt Specification http://www.digitalpreservation.gov/documents/bagitspec.pdf
- MD5: http://en.wikipedia.org/wiki/MD5; https://tools.ietf.org/html/rfc1321
- SHA-1: http://en.wikipedia.org/wiki/SHA-1; http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
- SHA-256: http://en.wikipedia.org/wiki/SHA-2; http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf
- Cyclic redundancy check (CRC): http://en.wikipedia.org/wiki/Cyclic_redundancy_check;