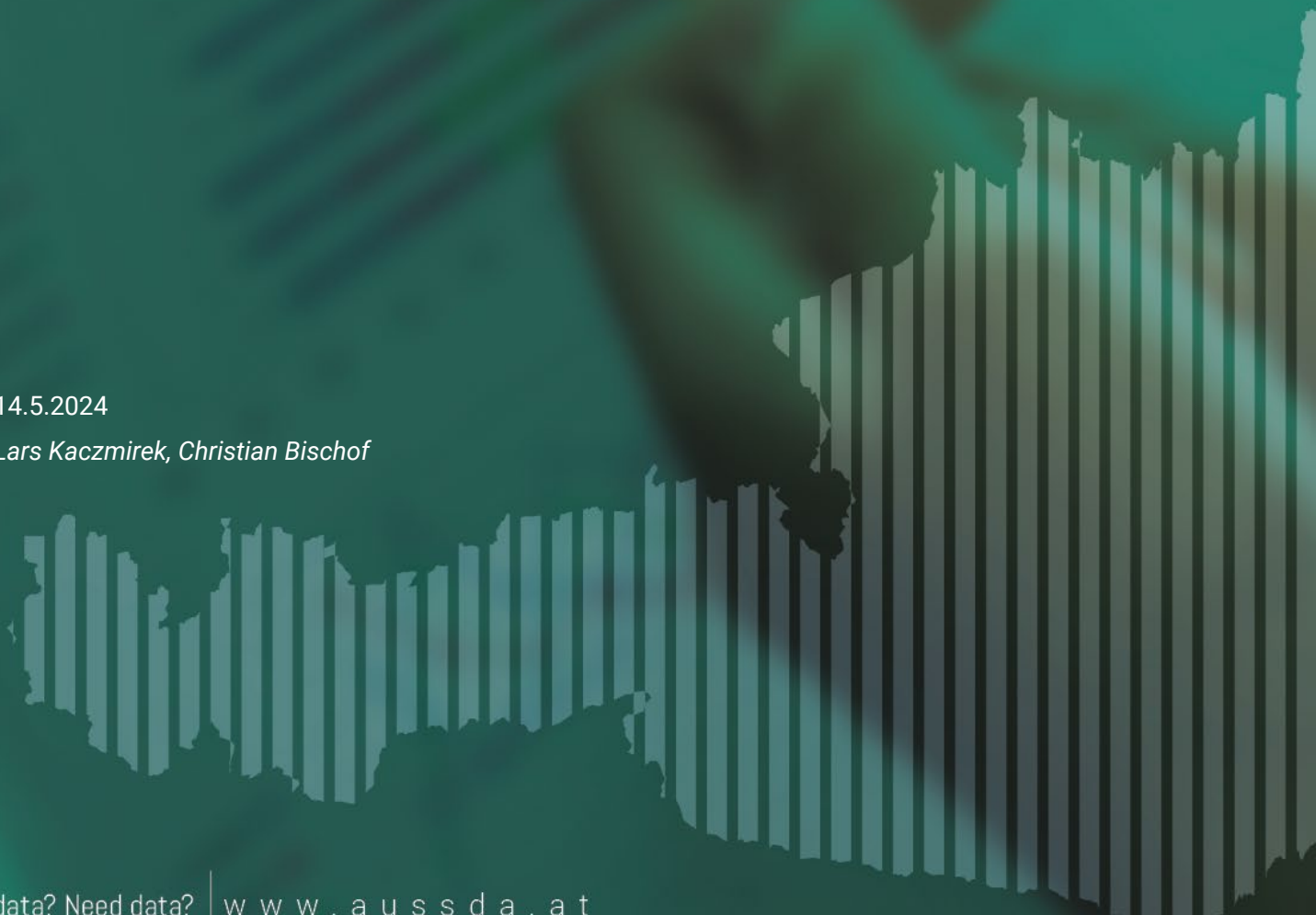
AUSSDA
AUSTRIAN SOCIAL SCIENCE DATA ARCHIVE

ENSURING LONG-TERM DATA PRESERVATION

Data integrity and fixity checks within the OAIS framework

14.5.2024

Lars Kaczmirek, Christian Bischof



Date	14.5.2024
Version	1.0
License	 This work is licensed under a Creative Commons Attribution 4.0 International License.
Access	Public
Suggested citation	Kaczmirek, L., Bischof, C. (2024). Ensuring long-term data preservation: data integrity and fixity checks within the OAIS framework. Vienna: AUSSDA - The Austrian Social Science Data Archive.

Scope

This document describes the procedure for fixity checks with checksums at AUSSDA – The Austrian Social Science Data Archive.

"Checksums have three main uses:

- To know that a file has been correctly received from a content owner or source and then transferred successfully to preservation storage.
- To know that file fixity has been maintained when that file is being stored.
- To be given to users of the file in the future so they know that the file has been correctly retrieved from storage and delivered to them."

(Digital Preservation Coalition, 2024, Digital Preservation Handbook, retrieved from: https://www.dpconline.org/handbook/technical-solutions-and-tools/fixity-and-checksums)

What is covered by fixity checks?

The fixity checks cover the long-term preservation storage that includes SIP, AIP, DIP. The files are stored on a server, on premises, in a shared folder with restricted write access to few staff members. The fixity checks and its related procedures ensure that no unnoticed changes are made to files or folders.

Backup procedures for the Dataverse production environment https://data.aussda.at/ and other data is not part of the specific process described in this document.

The fixity setup

We use the following software: hashdeep64.exe, version 4.4 (29 Jan 2014), downloaded from https://github.com/jessek/hashdeep/releases

Documentation: https://md5deep.sourceforge.net/

The licence for hashdeep is public domain. Here is an excerpt from the licence text: "This program is a work of the US Government. In accordance with 17 USC 105, copyright protection is not available for any work of the US Government. As such this code is considered public domain. There is NO warranty for this program; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE."

What files: fixity checksums are calculated for all files in the storage folder AUSSDA_Repository which is the storage location for long-term preservation. In OAIS terms, this includes SIP, AIP, DIP.

How often: at least once per quarter. The procedure can be started more often to reduce the workload for checking and documenting changes, if needed. For example, the checks can be run after a batch of files have been added to the repository. Changes made to the repository storage and fixity checks should not run on the same day to reduce the risk of human errors.

Checksum type used: SHA-256

Security measures: Write access is only granted to a few specified persons.

Fixity procedure

The fixity procedure describes the steps and tasks that need to be conducted to run one successful cycle of fixity checks.

1. Make sure that there have not been made any changes to the long-term storage today. Otherwise delay the fixity procedure until at least tomorrow.
2. Compute a new set of “knowns” for the next cycle.
3. Run the software with the knowns from the previous cycle. This calculates new checksums and compares them with the previously stored checksums. It does not store a new set of knowns. The output is a report.
4. Check the report and document the changes or correct the errors
    a. Correct any mistakes. Examples incl. corrections in the folder structure, naming of files to comply to naming conventions, requesting missing files, restore from backup
    b. Document changes that are correct, e.g. new files
    c. If errors were fixed, rerun the audit
5. Send the report to the head of AUSSDA
6. The head of AUSSDA checks the documentation for plausibility and signs off on the report.
7. The cycle is completed.

How to document changes

Usually the audit will fail because it is the very nature of the data repository to add files to its long-term storage. The most common and expected error will therefore be New File (“No match").

The auditor will sort the audit output according to the error types. The error types will lead to different tasks as described in the following:

"No match"
Likely cause: A new file was added. The file has no previous hash in the knowns. In such a case the auditor will add a note to the report, for example:

- OK. Files for the project were added to long term storage

"Moved from"
Check if it is a rename or the file has moved to a different location. Check the file name and the paths. If this is a mistake, correct the error and note the outcome. For example:

- ERROR CORRECTED Fixed human error, files have been moved to correct location. See AUDIT RERUN
- OK. File name is now correct. It was changed to comply with naming schema
- OK. File is now in correct location. It was moved to comply with storing rules

"Known file not used"
Likely causes: The file has been deleted or cannot be read. The file could not be found in any directory. Example notes:

- ERROR CORRECTED. Data was lost. File was restored from backup (backup date #). Backup successful.
- ERROR CORRECTED. Data has been corrupted. File was restored from backup (backup date #). Backup successful.

Related material

- Dataverse provides MD5 checksums and UNF numerical fingerprints for files for users. “The Dataverse Software calculates checksums for uploaded files so that users can determine if their file was corrupted via upload or download. This is sometimes called “file fixity”: https://en.wikipedia.org/wiki/File_Fixity“ (https://guides.dataverse.org/en/latest/installation/config.html#filefixitychecksumalgorithm)
- The software 7-zip adds CRC SHA and other checksum algorithms to the context menu of the Windows explorer. This enables users to check whether any file is the same file in the Dataverse repository or whether changes have been made or a download was corrupted.
- The software Teracopy can be used to copy files with integrated checksum checks. See also https://www.dpconline.org/handbook/tool-demos/teracopy-demo

Test of an implementation

The fixity check cycle was tested with a synthetic directory structure and a couple files.

Sample directory structure for testing

```
 Directoy of \hashtest

24.11.2022  12:26    <DIR>          .
24.11.2022  12:26    <DIR>          ..
24.11.2022  11:48    <DIR>          dataset1234
24.11.2022  11:47                 3 file.spss
               1 file(s),              3 Bytes

 Directoy of \hashtest\dataset1234

24.11.2022  11:48    <DIR>          .
24.11.2022  11:48    <DIR>          ..
24.11.2022  18:59    <DIR>          aip
24.11.2022  11:48    <DIR>          sip
               0 file(s),              0 Bytes

 Directoy of \hashtest\dataset1234\aip

24.11.2022  18:59    <DIR>          .
24.11.2022  18:59    <DIR>          ..
24.11.2022  11:47                 3 foo2.spss
               1 file(s),              3 Bytes

 Directoy of \hashtest\dataset1234\sip

24.11.2022  11:48    <DIR>          .
24.11.2022  11:48    <DIR>          ..
24.11.2022  11:46                 3 foo1.txt
24.11.2022  11:47                 3 foo2.spss
24.11.2022  11:48                 7 foo3.txt
               3 file(s)),             13 Bytes

     Number of shown files:
               5 file(s),             19 Bytes
              11 directories
```

Initial creation of hashes

Code used to compute initial hashes (referred to as knowns.txt):

```
hashdeep64 -r hashtest > knowns.txt
```

Parameters:

`-r`	incl. all subdirectories
`> knowns.txt`	pipe output of program into textfile

Run an audit

Code used to create audit output. This computes new hashes and compares them with the knowns.txt, incl. its locations. Note: a checksum remains the same, if only the file name changes.

```
hashdeep64 -a -vvv -k knowns.txt -r hashtest
```

Parameters:

`-a`	make audit
`-vvv`	very very verbose audit output if it fails (three v is the max output)
`-k`	input the previous hashes, i.e. the knowns
`-r`	incl. all subdirectories
`hashtest`	the directoy to check

The following tests were performed

The following checks were performed to test the various scenarios to detect changes

Error types that are detected are: file is renamed, file is moved, file is deleted, new file is added. These types can be distinguished by their corresponding error messages in the audit log:

File rename error: "Moved from". It is a rename, if the checksum is identical but the file name has changed.

File moved error: "Moved from". It is a move, if the paths of the known and new file are not identical

File deleted: "Known file not used". The file could not be found in any directory

New File: "No match". The file has no previous hash in the knowns