HORIZON-EIC-2022-PATHFINDERCHALLENGES-01



Project No. 101115115

Interoperable end-to-end **P**latform of scalabl**E** and sustain**A**ble high-th**R**oughput techno**L**ogies for **DNA**-based digital data storage

Deliverable 1.4
PEARL-DNA Channel Codec

WP1 – Data Preparation System

Authors	Yeremia G. Adhisantoso (LUH), Jan Voges (LUH), Tomaz Karcnik (BioSis)
Lead participant	LUH
Delivery date	08 September 2026
Dissemination level	PU = Public
Type	Demonstrator, pilot, prototype

Version 1.0

European Innovation Council

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Innovation Council and SMEs Executive Agency (EISMEA). Neither the European Union nor the granting authority can be held responsible for them. Swiss participants in this project are supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract numbers 23.00332 and 23.00328.

Revision History

Author(s)	Description	Date
Jan Voges (LUH)	Draft deliverable	17.08.2026
Jan Voges (LUH)	Revision 1 (update reference Bar+25)	25.08.2026
Jan Voges (LUH)	Revision 2 (add example in Section 4)	27.08.2026
Jan Voges (LUH)	Revision 3 (add storage format section [T1.5])	31.08.2026
Tomaž Karčnik, Tjaša Stare (BioSis)	Final version	31.08.2026
Andreia Cruz (acceICH)	Formal check	08.09.2026

Contents

Revision History 2
Contents 3
Partner Short Names 4
Abbreviations 4
Executive Summary 5
1 Introduction 6
2 High-Level Architecture 6
3 Demonstrator Validation 7
4 Requirements and Usage Guide 8
5 Future Work 9
6 References 9
7 Appendix: DNA Storage File System 10
7.1 Introduction 10
7.2 Design Requirements 10
7.3 Archive Architecture 12
7.3.1 Archive Header 12
7.3.2 Archive Directory 14
7.3.3 File Objects 15
7.4 Archive Processing 16
7.4.1 Archive Access 16
7.4.2 Writing 17
7.5 Notes on Versioning and Compatibility 17
7.6 Discussion 18
Complete Example of a Single-Generation Archive 19
A.1 Serialization assumptions 19
A.2 Overall archive layout 20
A.3 Archive Header 21
A.4 Archive Directory 22
A.5 File Object 1 23
A.6 File Object 2 24
A.7 Complete archive 25

Partner Short Names

No.	Organisation	Short name
1	**BioSistemika**	BioSis
2	**Gottfried Wilhelm Leibniz Universität Hannover**	LUH
3	**Imagene SA**	IMG
4	**Technische Hochschule Wildau**	THWi
5	**Haute École Specialisée de Suisse Occidentale**	HES-SO
6	**accelopment Schweiz AG**	accelCH

Abbreviations

Abbreviation	Term
API	Application Programming Interface
EC	European Commission
EU	European Union
HEU	Horizon Europe
LDPC	Low-Density-Parity-Check
SLURM	Simple Linux Utility for Resource Management
WP	Work Package

Executive Summary

Background

Deliverable D1.4 “PEARL-DNA Channel Codec” serves as the formal documentation and entry point for the software demonstrator of the PEARL-DNA channel codec. While the comprehensive theoretical framework and mathematical foundations of the channel model are detailed extensively in deliverable D4.6 “DNA Channel Model,” this document acts as a companion guide to the Python implementation itself. The codebase is shared with deliverable D1.5 “End-To-End Software Simulation,” which—in addition to the codec—features a simulation of synthesis, storage, and sequencing, as well as the compression module of deliverable D1.1 “PEARL-DNA Compression Codec”. In addition to the codec, this document details the PEARL-DNA Archive format as a logical description of the format for data stored in physical block-by-block assembled DNA.

Objectives

The primary objective is to ship a runnable codec that realises the encoding and decoding layers. The encoder architecture consists of compression, hierarchical error correction (Reed–Solomon [RS60], LDPC interleaving), modulation, and braid assembly. The decoder architecture mirrors this with clustering (alignment [NW70], aggregation, consensus), braid reconstruction, demodulation, and hierarchical error correction.

Methodology and Implementation

The codec is implemented within the `pearl_dna` Python package and is accessible via the `pearl-dna` console entry point. It provides a comprehensive encoding pipeline, and it provides a complementary deterministic decoding pipeline to reconstruct the encoded data. The primary documentation for the demonstrator resides within the repository itself; the codebase is rigorously documented with comprehensive README files and detailed Python docstrings to ensure straightforward usage, integration, and reproducibility.

Outcomes

The reference configuration successfully executes a bit-exact round-trip reconstruction in the “Voyager” experiment. In this scenario, approximately 3.4 kB of data were encoded, the synthesis, storage, and sequencing processes were simulated using real-world parameters, and the data was perfectly retrieved. This roundtrip was also successfully validated in a wet-lab setting.

Impact

This software demonstrator operationalises the WP1 codec theory, providing the executable substrate for downstream validation in WP2 and WP3. Furthermore, it enables reproducible benchmarking against established literature baselines such as [Bar+25].

Next Steps

The software will undergo continuous refinement. Planned algorithmic upgrades include the integration of belief-propagation ordered-statistics decoding for the inner LDPC code, alongside engineering enhancements such as improved SLURM offloading mechanisms for production-scale deployments.

1 Introduction

This deliverable introduces the software demonstrator for the PEARL-DNA channel codec. The primary theoretical underpinnings of this codec are documented separately in deliverable D4.6 “DNA Channel Model.” Therefore, this document functions primarily as a high-level guide to the software repository where the implementation resides.

The codec is designed specifically for block-by-block synthesised DNA, in which individual strands—referred to as braids—are assembled from a library of 16 predesigned oligonucleotides (strings). Each oligonucleotide string spans 398 base pairs, containing information-bearing coding regions flanked by fixed, non-coding gluing regions. The architecture of this codec is driven by the necessity to mitigate distinct error modalities introduced sequentially along the synthesis, storage, and sequencing pipeline.

2 High-Level Architecture

Figure 1 depicts the PEARL-DNA pipeline. It is operationalized through the deliverables D1.1. “PEARL-DNA Compression Codec”, D1.3 “End-To-End Software Simulation”, D1.4 “PEARL-DNA Channel Codec” (this deliverable), and D1.5 “End-To-End Software Simulation” (an update to D1.3).

This deliverable (D1.4) covers the following encoder modules: compression, hierarchical error correction (Reed–Solomon [RS60], LDPC interleaving), modulation, braid assembly. It also covers the corresponding decoder modules: clustering (alignment [NW70], aggregation, consensus), braid reconstruction, demodulation, hierarchical error correction (Reed–Solomon [RS60], LDPC interleaving).

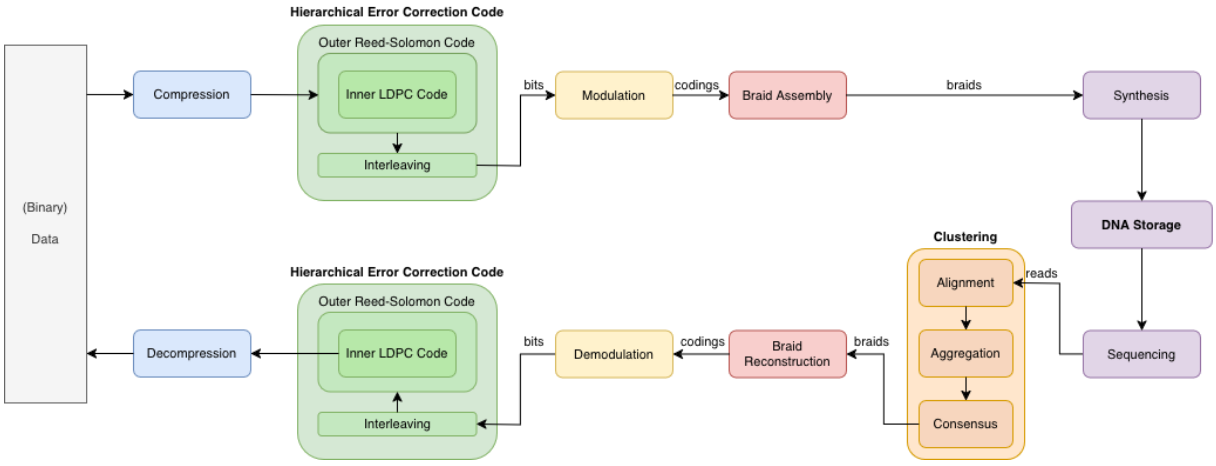


Figure 1: Block diagram of the PEARL-DNA pipeline.

3 Demonstrator Validation

The reference configuration successfully executes a bit-exact round-trip reconstruction in the "Voyager" experiment.

In this experiment, the Voyager Golden Record greeting text (3,475 bytes) was encoded, while synthesis, storage, and sequencing were simulated using real-world parameters. These parameters are documented in the configuration file `configs/experiment/voyager_greetings/s5_b4096_v2.yaml`, where "s5" refers to the fifth and current codec architecture, and "v2" refers to version 2 of the codec parameters, which are closely fitted to the empirical data.

This roundtrip was also successfully repeated in a wet-lab setting. The process is documented in the repository's main README file and is summarized below (all commands assume the active environment is `pearl_dna`).

1. Encode (data → DNA)

```
mamba run -n pearl_dna python -m pearl_dna.cli encode \
  --experiment voyager_greetings/s5_b4096_v2 \
  --input data/experiments/voyager_greetings/voyager_greetings.txt \
  --output /tmp/voyager_reference.fasta
```

2. Simulate (DNA → noisy reads)

```
mamba run -n pearl_dna python -m pearl_dna.cli simulate \
  --experiment voyager_greetings/s5_b4096_v2 \
  --input /tmp/voyager_reference.fasta \
  --output_r1 /tmp/reads_R1.fastq \
  --output_r2 /tmp/reads_R2.fastq \
  --coverage 10 --read_length 150
```

3. Reconstruct (reads → data)

```
mamba run -n pearl_dna python -m pearl_dna.cli reconstruct \
  --experiment voyager_greetings/s5_b4096_v2 \
  --mode full \
  --r1 /tmp/reads_R1.fastq \
  --r2 /tmp/reads_R2.fastq \
  --output /tmp/recovered
```

4. Verify

```
import bz2
original = open("data/experiments/voyager_greetings/voyager_greetings.txt",
"rb").read()
recovered = open("/tmp/recovered/stage5/decoded-*.bin", "rb").read()
decompressed = bz2.decompress(recovered)
assert original == decompressed, "Mismatch!"
```

4 Requirements and Usage Guide

The most comprehensive and up-to-date documentation for the codec is embedded directly within the software repository. Users should consult the extensive README files and the inline Python docstrings for exhaustive usage instructions, API referencing, and deployment guides.

Technically, the codec requires Python version 3.11 or higher and operates within a dedicated mamba environment named `pearl_dna`. The implementation relies on robust computational libraries including `numpy`, `scipy`, `biopython`, and `galois` for finite field arithmetic.

As an example, as documented in the main README file, the following commands run the full production pipeline with the S5-v4 config (voyager_greetings/s5_b2720_v4, 2720 braids) to reproduce the bit-exact round-trip reconstruction in the “Voyager” experiment.

```
# 1. Encode (Data → DNA)
conda run -n pearl_dna-develop python -m pearl_dna.cli encode \
  --experiment voyager_greetings/s5_b2720_v4 \
  --input data/experiments/voyager_greetings/voyager_greetings.txt \
  --output /tmp/voyager_v4_reference.fasta

# 2. Simulate (DNA → noisy reads)
conda run -n pearl_dna-develop python -m pearl_dna.cli simulate \
  --experiment voyager_greetings/s5_b2720_v4 \
  --input /tmp/voyager_v4_reference.fasta \
  --output_r1 /tmp/voyager_v4_R1.fastq \
  --coverage 50 --read_length 500

# 3. Reconstruct (reads → Data) — Stage 5 recovers the payload, Stage 6 restores
the file
conda run -n pearl_dna-develop python -m pearl_dna.cli reconstruct \
  --experiment voyager_greetings/s5_b2720_v4 --mode full \
  --r1 /tmp/voyager_v4_R1.fastq --num_workers 8 ++samples=140000

# 4. Verify — the Stage 6 output is the byte-identical original file (BER=0)
cmp data/experiments/voyager_greetings/voyager_greetings.txt \
  outputs/10_nato_data_recovery/recovered-voyager_greetings.txt \
  && echo "Roundtrip OK (byte-identical)"
```

5 Future Work

The software will be improved continuously. Planned algorithmic enhancements include the implementation of belief-propagation ordered-statistics decoding for the inner code. Engineering improvements will focus on tightening SLURM offloading mechanisms to support high-throughput, production-scale deployments efficiently.

6 References

[Bar+25] Bar-Lev, D., Orr, I., Sabary, O. et al. Scalable and robust DNA-based storage via coding theory and deep learning. *Nat Mach Intell* **7**, 639–649 (2025).

[NW70] Needleman & Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, *Journal of Molecular Biology* **48**, 443–453 (1970).

[RS60] Reed & Solomon, “Polynomial Codes Over Certain Finite Fields”, *Journal of the Society for Industrial and Applied Mathematics* **8**, 300–304 (1960).

7 Appendix: DNA Storage File System

7.1 Introduction

The PEARL-DNA project develops technologies for long-term, high-density and energy-efficient DNA data storage. Task T1.5 addresses the logical organization of stored information through the DNA Storage Format Module, which defines how archive metadata and user data are structured independently of their physical representation in DNA.

The PEARL DNA Archive Format is separated from the storage codec. The Archive Format defines Archive Headers, Archive Directories, File Objects and Archive Generations, while the codec handles DNA encoding, strand organization, error correction and reconstruction from sequencing data. This separation keeps the format independent of specific synthesis and sequencing technologies, although Version 1 was developed in the context of the PEARL-DNA block-by-block approach.

The design reflects the immutable nature of DNA storage. Previously synthesized information is not modified in place; instead, Version 1 supports append-only evolution through successive Archive Generations. This avoids the complexity of implementing conventional writable file systems.

The Canonical Archive is self-contained and stores all information required to identify the archive, determine its size, enumerate files and locate File Objects. An optional External Archive Index may accelerate access, but it is never required for interpretation.

Version 1 targets archives of approximately 1 GB and focuses on essential archival functions: identification, metadata retrieval, file enumeration, addressing and reconstruction through the codec. The format is intended to be openly documented to support long-term accessibility and interoperability.

7.2 Design Requirements

The Storage Format Module provides a standardized logical organization for information stored in a DNA archive. It defines how files and archive metadata are represented after processing by lower storage and protection layers, forming the interface between the DNA storage codec and the user-visible archive. The format is designed for long-term archival storage while remaining independent of the underlying DNA writing and reading technologies.

DNA archives differ from conventional storage because they are not intended for frequent modification or interactive access. Data are written by DNA synthesis and retrieved through sequencing. The format is therefore optimized for immutable, write-once archives that are read only occasionally. This simplifies archive organization and reflects the expected characteristics of first-generation DNA storage systems.

The design favors simplicity, long-term readability and technology independence rather than functionality required by frequently modified storage media. A clear separation is maintained between the Archive Format and the codec. The Archive Format defines the Archive Header, Archive Directory, File Objects and Archive Generations, while the codec handles DNA encoding, strand organization, strand-level addressing, error correction and sequencing reconstruction. The Archive Format defines the logical organization of information, while the codec defines its physical realization, providing a clean and extensible interface for long-term DNA data storage. This separation makes the archive format independent of the underlying implementation.

The following requirements define Version 1 of the PEARL DNA Archive Format.

- Functional requirements

The archive shall be self-contained and store all information needed to identify the archive, determine its size, enumerate files, locate a selected file and retrieve its contents. No external metadata or proprietary information shall be required. Optional external metadata may be used only to accelerate access.

- Independence from DNA encoding

The Archive Format shall be independent of nucleotide encoding, synthesis technology, sequencing platform and error-correction scheme. Version 1 was developed in the context of the PEARL-DNA block-by-block implementation. DNA encoding, error correction and sequence reconstruction remain responsibilities of the storage codec.

- Immutability

Previously written archive data shall not be modified in place. Version 1 supports append-only archive extension, avoiding mechanisms such as in-place updates, deletion, block allocation and space reclamation.

- Self-containment

Archive metadata shall be sufficient to identify the archive, specify the format version, describe its logical organization and enable navigation without external systems. Reserved fields and version identifiers support future extensions.

- Addressing

Each stored file shall have an address that uniquely identifies its location within the archive, independently of the physical distribution of data across DNA strands.

- Scalability and versioning

Version 1 targets archives of approximately 1 GB. Explicit version information shall enable future extensions, including larger archives and additional metadata, while preserving compatibility with Version 1. Larger files are handled manually by user through standard split/merge utilities.

- Metadata before data

Archive metadata shall be accessible before File Objects, allowing archive properties and file lists to be obtained without reconstructing the complete archive.

- Simplicity

Version 1 deliberately limits functionality to archive identification, directory traversal and file retrieval, providing a compact and straightforward implementation.

- DNA-oriented design

The format is optimized for DNA storage characteristics, including sequencing-based retrieval and immutable data. These characteristics favor append-oriented archive evolution rather than in-place modification.

- Public specification

The Archive Format is intended to be openly documented, enabling independent interpretation of correctly reconstructed archives without proprietary software, external registries or “Rosetta Stone” mechanisms.

7.3 Archive Architecture

The Archive Format organizes data into three logical structures: Archive Header, Archive Directory, and File Objects. These structures operate on data reconstructed by the storage codec and are independent of their physical representation in DNA.

- Archive Header

The Archive Header is the archive entry point. It identifies the archive and format version, provides archive-wide metadata, and contains the address of the Archive Directory. Each Archive Generation contains one Archive Header.

- Archive Directory

The Archive Directory acts as the logical table of contents. It contains one record for each stored file, including its name, size, and address of the corresponding File Object. Each Archive Generation has its own Directory.

- File Objects

Each stored file is represented by one File Object containing its logical file contents. File segmentation and physical DNA representation are handled by the codec and are outside the scope of the Archive Format.

- Addressing

Archive structures are referenced by addresses independent of physical DNA locations. This abstraction enables compatibility with different codecs, DNA strand organizations, and future storage technologies. Translation to the physical DNA representation is performed by the codec.

To summarize: the Archive Header points to the Archive Directory, which in turn points to individual File Objects as shown in Figure 2. Such hierarchical organization keeps the archive structure simple and separates logical organization from physical DNA storage.

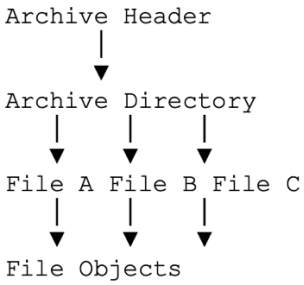


Figure 2: Archive logical structure.

7.3.1 Archive Header

The Archive Header is the root metadata structure of the PEARL DNA Archive Format. Each Archive Generation contains exactly one Header, which identifies the archive, specifies the format version, provides archive-wide metadata and locates the corresponding Archive Directory.

Functional requirements include:

- uniquely identify the archive generation;
- specify the archive format version;
- locate the corresponding Archive Directory;
- provide basic archive statistics;

- support archive evolution without modifying previously synthesized DNA;
- remain independent of physical DNA organization.

Header structure Version 1 is structured as follows:

Field	Description
Magic Number	Identifies the PEARL DNA or any other Archive Format.
Archive Format Version	Version of the archive specification.
Archive Identifier	Globally unique identifier shared by all generations of the archive.
Generation Number	Sequential archive generation number.
Previous Header Address	Logical address of the previous archive generation. Null for the first generation.
Next Header Address	Logical address of the next archive generation. If the last one, the header structure at pointed to logical address is not valid.
Archive Directory Address	Logical address of the Archive Directory belonging to this generation.
Archive Size	Total logical size of all stored data, including all headers.
Number of Files	Number of File Objects described by the Archive Directory.
Creation Timestamp	Date and time when this archive generation was created.
Reserved	Reserved for future extensions.

- Archive identification

The Magic Number identifies the archive format, while the Archive Format Version defines the specification revision. The Archive Identifier is shared by all generations belonging to the same logical archive.

- Archive Generation Chain

Archive Generations are connected through Previous and Next Header Addresses, forming a doubly linked chain. The Next Header Address is assigned when a Header is created and reserves the location for a future Header. If no valid Header exists at that address, the current Header represents the most recent generation. Existing Headers are never modified.

- Archive Directory reference

Each Archive Header contains the address of the Archive Directory belonging to the corresponding generation. Different generations therefore reference different Archive Directories.

- Archive statistics

The Archive Header stores summary information, including Archive Size and Number of Files, allowing basic archive properties to be obtained without reading the full Archive Directory.

- Reserved fields

Reserved fields support future extensions. Implementations shall ignore fields they do not recognize, preserving compatibility with Version 1 archives.

- Header processing

Archive interpretation starts from the predefined initial Header. The reader verifies the Magic Number and format version, follows the Header Chain to the most recent valid generation, reads its Archive

Directory, locates the requested File Object and reconstructs the file. If an External Archive Index is available, it may be used to bypass Header Chain traversal.

The Canonical Archive contains all Headers, Directories and File Objects required for archive interpretation and is therefore fully self-contained. An optional External Archive Index may store the latest Header Address or archive statistics, but it is never required and is not authoritative. When an External Archive Index is available, an implementation may use it to directly locate the most recent Archive Header, thereby avoiding traversal of the Archive Header Chain. Such use is strictly optional and shall not alter the logical interpretation of the archive.

7.3.2 Archive Directory

The Archive Directory is the logical index of one Archive Generation. It lists all stored files and provides the information needed to locate their File Objects. Each Archive Generation contains exactly one Archive Directory referenced by its Archive Header.

Functional requirements can be summarized as:

- o list all files in the generation;
- o provide each File Object Address;
- o provide each file size;
- o remain independent of the storage codec and physical DNA representation;
- o support future archive versions.

Directory structure

The Archive Directory is an ordered table consisting of a Directory Header followed by Archive Directory Records (ADRs). Each ADR describes one File Object. The position of an ADR implicitly defines its record number, so no separate record identifier is stored.

Directory Header

Table 6-1. Directory Header Version 1

Field Description

Directory Version Version of the Archive Directory structure.

Archive Generation Number Generation number of the Archive Generation described by this directory.

Number of Records Number of Archive Directory Records contained in the ordered table.

Directory Size Logical size of the complete Archive Directory.

Reserved Reserved for future extensions.

- Each Active Directory record (ADR) contains:
- o File Name;
- o File Object Address;
- o File Size;
- o Reserved field.

Each ADR references exactly one File Object.

- File identification

Files are identified internally by the ADR record number, derived from their position in the ordered table. The File Object Address is used to locate stored data, while the File Name is intended for human interpretation only.

- File location

The File Object Address uniquely identifies the corresponding File Object. Translation to the physical DNA representation is performed by the storage codec.

- Directory evolution

Every Archive Generation contains a new Archive Directory describing that generation. Earlier Archive Directories remain unchanged, preserving the immutability of previously stored DNA.

- Directory processing

Processing consists of reading and verifying the Directory Header, scanning the ordered ADR table, locating the requested record, obtaining its File Object Address, and retrieving the corresponding File Object.

Archive Directory

Directory Header



Ordered ADR Table



Requested Record



File Object Address



File Object

7.3.3 File Objects

A File Object represents one stored file and contains the archive payload. Each File Object is referenced by an Archive Directory Record. A File Object functional requirements can be summarized as:

- be immutable;
- represent exactly one archived file;
- have a unique Logical Address;
- preserve the complete file contents;
- remain independent of DNA representation and the storage codec.

- File Object representation

A File Object is an uninterpreted sequence of bytes containing the complete stored file. Its internal meaning is application-dependent and outside the scope of the Archive Format.

- File Object identification

Each File Object is identified by a unique Logical Address assigned during archive creation. File Objects are referenced through these addresses stored in the Archive Directory; direct references between File Objects are not defined in Version 1.

- Archive Directory relationship

Each Archive Directory Record provides:

o File Name;

o File Object Address;

o File Size.

The Archive Directory therefore maps user-visible file names to the corresponding File Objects.

Archive Directory

```
|
├— Record 1 —▶ File Object A
├— Record 2 —▶ File Object B
└— Record 3 —▶ File Object C
```

7.4 Archive Processing

7.4.1 Archive Access

Retrieval of information from the archive follows a deterministic sequence.

1. Read the Archive Header.
2. Verify compatibility with the supported archive format version.
3. Locate and read the Archive Directory.
4. Enumerate the files stored in the archive.
5. Select the required file.
6. Obtain the logical address of the corresponding File Object.
7. Retrieve and reconstruct the file contents.

This access procedure enables archive metadata to be inspected without reconstructing the complete archive. Consequently, operations such as obtaining the list of stored files or determining the archive size require interpretation only of the Archive Header and Archive Directory. External archive index, if available and valid, may be used to speed up the access. Archive-wide information, including the number of files and logical archive size, is obtained from the Archive Header and Archive Directory without requiring reconstruction of File Objects. If an archive structure cannot be interpreted, processing shall terminate. Detection and correction of physical storage errors remain the responsibility of the storage codec.

Retrieval of any data from DNA archive is done through sequencing.

7.4.2 Writing

Version 1 supports archive growth through creation of additional Archive Generation while existing archive structures shall never be modified.

A new Archive Generation consists of:

- a new Archive Header;
- a new Archive Directory;
- new File Objects corresponding only to newly added files.

Previously created File Objects remained unchanged. The new Archive Header is linked into the Archive Header Chain by a predetermined-next-address mechanism. Archive readers supporting this mechanism may traverse the chain to obtain the most recent logical state of the archive or access it through External Archive Index, if available.

The resulting Archive Header Chain represents the complete evolution of the archive.

Version 1 does not support:

- deletion of files;
- in-place modification of files;
- directory reorganization;
- archive compaction.

7.5 Notes on Versioning and Compatibility

Every Archive Header contains an Archive Format Version.

Applications shall verify version compatibility before interpreting additional archive structures. Any future version shall preserve the interpretation of Version 1 archives. Contrary, Version 1 ignores unknown reserved fields. In case of errors the interpretation of archive will terminate. Error correction is not resolved at format level.

Potential future extensions include:

- richer metadata;
- directory attributes;
- file permissions;
- archive authentication;
- larger logical address spaces.

None of these extensions should invalidate Version 1 archives.

7.6 Discussion

The proposed Archive Format provides a lightweight logical organization specifically intended for immutable DNA-based archival storage. Unlike conventional disk-oriented file systems, the proposed format deliberately omits mechanisms associated with writable media, such as free-space management, block allocation, journaling and transaction recovery. This simplification reflects both the physical characteristics and the current economics of DNA data storage.

A fundamental architectural feature of the proposed format is the strict separation between logical archive organization and physical DNA representation. The archive format defines only logical structures—Archive Headers, Archive Directories and File Objects, while all aspects related to segmentation, DNA strand organization, addressing within DNA molecules and error protection are delegated to the storage codec. This separation allows the archive format to remain independent of synthesis technology, sequencing technology and DNA encoding methodology.

The archive is fully self-contained. All information required to identify the archive, enumerate stored files and retrieve their contents is stored within the Canonical Archive. Optional implementation-specific mechanisms, such as an External Archive Index, may improve performance but are never required for archive interpretation.

The archive evolution mechanism is based on successive Archive Generations linked through the Archive Header Chain. Existing archive structures remain immutable.

Version 1 intentionally provides only the functionality required for practical archival applications. It also does not address writing large files exceeding the Version1 specifications; that is left to the user. It does not attempt to replicate the capabilities of conventional large-scale archiving file systems. Instead, it establishes a compact and deterministic logical archive format that may serve as a stable foundation for future generations of DNA data storage systems.

Complete Example of a Single-Generation Archive

This annex provides an illustrative byte-level serialization of a PEARL DNA Archive containing two files in a single Archive Generation. The logical structures follow the current archive specification: one Archive Header, one Archive Directory containing two Archive Directory Records, and two File Objects.

The two stored files are:

- `file1.txt`, containing `This is file 1.` — 15 bytes.
- `file2.txt`, containing `010203040506070809` — 18 bytes.

The example is intended to demonstrate the complete archive structure and byte sequence. Values that are not yet fixed by the specification, such as the Magic Number and Archive Identifier, are selected illustratively.

A.1 Serialization assumptions

All addresses are represented as fixed-size 4-byte unsigned integers containing absolute byte offsets from the beginning of the serialized archive.

All 4-byte integer values are encoded in big-endian byte order.

The following field sizes are used:

Field	Size
Magic Number	4 bytes
Archive Format Version	1 byte
Archive Identifier	16 bytes
Generation Number	1 byte
Previous Header Address	4 bytes
Next Header Address	4 bytes
Archive Directory Address	4 bytes
Archive Size	4 bytes
Number of Files	4 bytes
Creation Timestamp	4 bytes
Header Reserved	1 byte
Directory Version	1 byte

Field	Size
Archive Generation Number	1 byte
Number of Records	4 bytes
Directory Size	4 bytes
Directory Reserved	1 byte
File Name	12 bytes
File Object Address	4 bytes
File Size	4 bytes
ADR Reserved	1 byte

The value `0xFFFFFFFF` is used as the NULL address. File names are stored in fixed 12-byte fields. Shorter names are padded with zero bytes. Field `Archive Size` represents the complete serialized size of the Archive Generation, including its Archive Header, Archive Directory and File Objects.

The following illustrative values are used:

- Magic Number: ASCII `PDNA`
- Archive Format Version: `1`
- Archive Identifier: `7D A4 1C 92 6B 3F 4E 11 8A C7 D2 50 91 6E B8 03`
- Generation Number: `1`
- Creation Timestamp: `68 A3 13 20`
- Reserved fields: zero

A.2 Overall archive layout

The sizes of the archive structures are:

Structure	Address	Size
Archive Header 1	`0x00000000`	47 bytes
Archive Directory 1	`0x0000002F`	53 bytes
File Object 1	`0x00000064`	15 bytes
File Object 2	`0x00000073`	18 bytes
Reserved location for Archive Header 2	`0x00000085`	—

The total archive size is therefore: `47 + 53 + 15 + 18 = 133 bytes`

The complete Archive Generation occupies byte addresses: `0x00000000` to `0x00000084`

The next available byte is: `0x00000085`

This address is stored as the Next Header Address. Because no valid Archive Header exists at this address, Archive Header 1 represents the most recent Archive Generation. The resulting layout is shown in Figure 3.

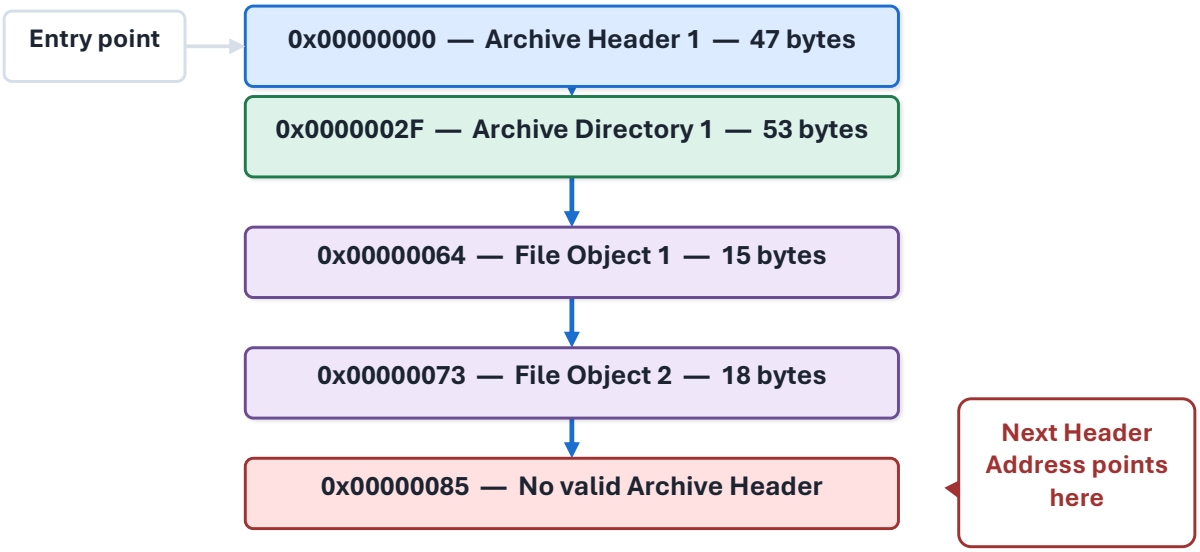


Figure 3: Addresses and data structures in the example

A.3 Archive Header

Archive Header 1 begins at the predefined initial archive address `0x00000000`, and its complete size is 47 bytes.

Offset	Field	Value	Serialized bytes
`0x00`	Magic Number	`PDNA`	`50 44 4E 41`
`0x04`	Archive Format Version	1	`01`
`0x05`	Archive Identifier	illustrative	`7D A4 1C 92 6B 3F 4E 11 8A C7 D2 50 91 6E B8 03`
`0x15`	Generation Number	1	`01`
`0x16`	Previous Header Address	NULL	`FF FF FF FF`
`0x1A`	Next Header Address	`0x00000085`	`00 00 00 85`
`0x1E`	Archive Directory Address	`0x0000002F`	`00 00 00 2F`
`0x22`	Archive Size	133 bytes	`00 00 00 85`
`0x26`	Number of Files	2	`00 00 00 02`

Offset	Field	Value	Serialized bytes
0x2A	Creation Timestamp	illustrative	68 A3 13 20
0x2E	Reserved	0	00

The complete Archive Header is therefore:

```
50 44 4E 41 01 7D A4 1C 92 6B 3F 4E 11 8A C7 D2
50 91 6E B8 03 01 FF FF FF FF 00 00 00 85 00 00
00 2F 00 00 00 85 00 00 00 02 68 A3 13 20 00
```

A.4 Archive Directory

The Archive Directory begins at the address stored in Archive Header 1: `0x0000002F`

The Archive Directory contains:

- one 11-byte Directory Header;
- one 21-byte Archive Directory Record for `file1.txt`;
- one 21-byte Archive Directory Record for `file2.txt`.

Its total size is: `11 + 21 + 21 = 53 bytes`

The Directory Header begins at: `0x0000002F`

Address	Field	Value	Serialized bytes
0x2F	Directory Version	1	01
0x30	Archive Generation Number	1	01
0x31	Number of Records	2	00 00 00 02
0x35	Directory Size	53	00 00 00 35
0x39	Reserved	0	00

Complete Directory Header: `01 01 00 00 00 02 00 00 00 35 00`

Archive Directory Record 1 begins at: `0x0000003A`. Its implicit record number is 1.

The file name `file1.txt` occupies nine bytes and is padded to the fixed 12-byte File Name field using three zero bytes:

```
66 69 6C 65 31 2E 74 78 74 00 00 00
```

The complete record is:

Field	Value	Serialized bytes
File Name	`file1.txt`	`66 69 6C 65 31 2E 74 78 74 00 00 00`
File Object Address	`0x00000064`	`00 00 00 64`
File Size	15 bytes	`00 00 00 0F`
Reserved	0	`00`

Complete Archive Directory Record 1:

```
66 69 6C 65 31 2E 74 78 74 00 00 00 00 00 00 64 00 00 00 0F 00
```

Archive Directory Record 2 begins at: `0x0000004F`. Its implicit record number is 2.

The file name `file2.txt` is likewise padded to 12 bytes:

```
66 69 6C 65 32 2E 74 78 74 00 00 00
```

The complete record is:

Field	Value	Serialized bytes
File Name	`file2.txt`	`66 69 6C 65 32 2E 74 78 74 00 00 00`
File Object Address	`0x00000073`	`00 00 00 73`
File Size	18 bytes	`00 00 00 12`
Reserved	0	`00`

Complete Archive Directory Record 2:

```
66 69 6C 65 32 2E 74 78 74 00 00 00 00 00 00 73 00 00 00 12 00
```

The complete Archive Directory is therefore:

```
01 01 00 00 00 02 00 00 00 35 00
66 69 6C 65 31 2E 74 78 74 00 00 00 00 00 00 64 00 00 00 0F 00
66 69 6C 65 32 2E 74 78 74 00 00 00 00 00 00 73 00 00 00 12 00
```

A.5 File Object 1

File Object 1 begins at:

```
0x00000064
```

The object is the content of `file1.txt`. containing:

```
This is file 1.
```

Serialized ASCII/UTF-8 representation:

```
54 68 69 73 20 69 73 20 66 69 6C 65 20 31 2E
```

File size is `15 bytes = 0x0000000F.`

The File Object occupies:

```
0x00000064–0x00000072
```

A.6 File Object 2

File Object 2 begins at:

```
0x00000073
```

The object contains: `010203040506070809`

Serialized ASCII/UTF-8 representation:

```
30 31 30 32 30 33 30 34 30 35 30 36 30 37 30 38 30 39
```

File size is `18 bytes = 0x00000012.`

The File Object occupies:

```
0x00000073–0x00000084
```

The next byte address is therefore:

```
0x00000085
```

which is identical to the Next Header Address stored in Archive Header 1.

No valid Archive Header is currently present at this address. Archive Header 1 is therefore the most recent Archive Header and Generation 1 is the current Archive Generation.

A.7 Complete archive

The complete logical structure is shown in Figure 4:

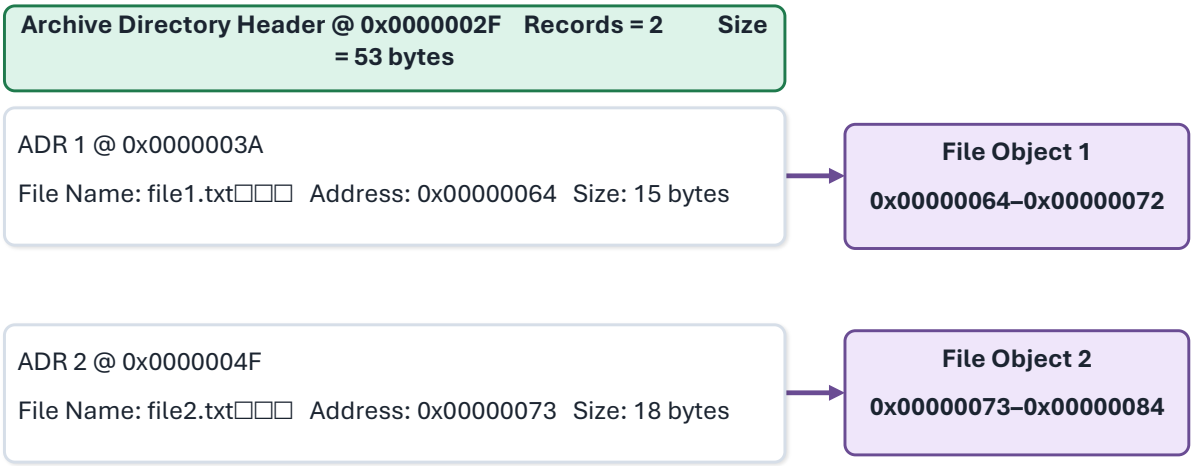


Figure 4: Archive structure

The final archive contains exactly 133 bytes and it is shown as a hexadecimal dump with byte offsets:

```
0000: 50 44 4E 41 01 7D A4 1C 92 6B 3F 4E 11 8A C7 D2
0010: 50 91 6E B8 03 01 FF FF FF FF 00 00 00 85 00 00
0020: 00 2F 00 00 00 85 00 00 00 02 68 A3 13 20 00 01
0030: 01 00 00 00 02 00 00 00 35 00 66 69 6C 65 31 2E
0040: 74 78 74 00 00 00 00 00 00 64 00 00 00 0F 00 66
0050: 69 6C 65 32 2E 74 78 74 00 00 00 00 00 00 73 00
0060: 00 00 12 00 54 68 69 73 20 69 73 20 66 69 6C 65
0070: 20 31 2E 30 31 30 32 30 33 30 34 30 35 30 36 30
0080: 37 30 38 30 39
```

The byte boundaries can be summarized as:

Byte range	Structure
0x00000000–0x0000002E	Archive Header 1
0x0000002F–0x00000039	Directory Header
0x0000003A–0x0000004E	Archive Directory Record 1
0x0000004F–0x00000063	Archive Directory Record 2
0x00000064–0x00000072	File Object 1
0x00000073–0x00000084	File Object 2
0x00000085	Predetermined location for a possible Archive Header 2

This example demonstrates that a reader starting at byte address `0x00000000` can determine the complete structure of the archive solely from the information stored within the archive. The Archive Header identifies the Archive Directory, the Archive Directory provides the addresses and sizes of both File Objects, and the Next Header Address determines whether a subsequent Archive Generation is present. No external metadata are required for interpretation.