| | |
|---|---|
| Workgroup: | Independent Submission |
| independent: | g3fc-spec-01 |
| Published: | 27 July 2025 |
| Intended Status: | Informational |
| Expires: | 28 January 2026 |
| | |
|---|---|
| Author: | L. Guimaraes |
| | *G3Pix* |

# G3FC (G3 File Container) File Format Specification

## Abstract

This document provides the complete technical specification for the G3FC (G3 File Container) binary format, version 1.0. G3FC is designed to store multiple files and directories in a single, robust container, which can optionally be split into multiple data blocks. The format includes features for data integrity via checksums, security through authenticated encryption, and resilience via forward error correction (FEC). This specification details the file structure, data types, field layouts, and the algorithms required to implement compatible software for reading and writing G3FC archives.

## License



## Status of This Memo

This is the final technical specification for version 1.0

## Copyright Notice



https://g3pix.com.br/g3fc/
https://github.com/guimaraeslucas/g3fc/

## Table of Contents

# 1. Introduction

The G3FC (G3 File Container) format provides a structured method for archiving multiple files and directories into a single container or a set of segmented files. It was designed with a focus on robustness, data integrity, security, and failure recovery. The format defines a clear layout with a header, a file index, data blocks, and a footer, allowing for efficient access and manipulation of the contained data.

This specification is intended for developers who need to implement G3FC-compatible tools for creating, reading, or modifying archives.

## 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 1.2. General Concepts

Byte Order (Endianness):
: All multi-byte integer data types (e.g., uint16_t, int64_t) SHALL be stored in Little Endian byte order.

Text Encoding:
: All textual data, including filenames and paths, SHALL be encoded using UTF-8. Fixed-size text fields SHALL be padded with null bytes (0x00) if the content is smaller than the field size.

Timestamps:
: All timestamps SHALL be stored as a 64-bit signed integer (int64_t), representing the number of 100-nanosecond intervals since 00:00:00 UTC on January 1, 0001, in the Gregorian calendar. This is compatible with .NET Ticks.

## 2. Overall File Structure

A G3FC archive consists of a main index file (with a .g3fc extension) and zero or more data block files (with .g3fc<n> extensions). A non-split archive contains all components within the single .g3fc file.

The logical structure of a non-split archive is as follows:

1. **Main Header:** A fixed-size block containing essential metadata about the container.
2. **File Index:** A catalog of all files and directories stored in the archive.
3. **File Data Block:** The actual content of the archived files, concatenated.
4. **Data FEC Block (Optional):** Parity data for the File Data Block section.
5. **Metadata FEC Block:** Parity data protecting the Main Header and the uncompressed File Index.
6. **Footer:** A fixed-size block at the end of the file for quick access to key structures.

In a split archive, the .g3fc file contains the Header, File Index, Metadata FEC Block, and Footer. The File Data is stored in separate data block files (e.g., archive.g3fc0, archive.g3fc1).

## 3. Main Header

The Main Header is a fixed-size block of 331 bytes located at the beginning of the .g3fc file. It MUST NOT be compressed or encrypted. The fields are aligned sequentially with 1-byte packing.

| Offset (B) | Size (B) | Data Type | Field Name | Description |
|---|---|---|---|---|
| 0 | 4 | char[4] | magic_number | MUST contain the ASCII characters "G3FC". |
| 4 | 2 | uint16_t | format_version_major | Major version of the specification. SHALL be 1. |
| 6 | 2 | uint16_t | format_version_minor | Minor version of the specification. SHALL be 0. |
| 8 | 16 | byte[16] | container_uuid | A 16-byte UUID (v4 RECOMMENDED) that uniquely identifies the container. |
| 24 | 8 | int64_t | creation_timestamp | Timestamp of the container's creation. |

| Offset (B) | Size (B) | Data Type | Field Name | Description |
|---|---|---|---|---|
| 32 | 8 | int64_t | modification_timestamp | Timestamp of the last modification. |
| 40 | 4 | uint32_t | edit_version | Starts at 1 and MUST be incremented on each modification. |
| 44 | 32 | char[32] | creating_system | Name of the creating software, UTF-8, null-padded. |
| 76 | 32 | char[32] | software_version | Version of the creating software, UTF-8, null-padded. |
| 108 | 8 | uint64_t | file_index_offset | Absolute offset (in bytes) from the beginning of the file to the start of the File Index. |
| 116 | 8 | uint64_t | file_index_length | Length of the File Index in bytes (after compression and encryption). |
| 124 | 1 | uint8_t | file_index_compression | 0: None, 1: Zstandard. Current implementations SHALL use 1. |
| 125 | 1 | uint8_t | global_compression | 0: Per-file compression, 1: Zstandard compression on the entire data block. |
| 126 | 1 | uint8_t | encryption_mode | 0: None, 1: Single password for read/write. |
| 127 | 64 | byte[64] | read_salt | A 64-byte salt for the read password's KDF. MUST be zero-filled if not used. |
| 191 | 64 | byte[64] | write_salt | A 64-byte salt for the write password's KDF. MUST be zero-filled if not used. |
| 255 | 4 | uint32_t | kdf_iterations | Number of iterations for PBKDF2. SHOULD be a high value (e.g., >= 100,000). |
| 259 | 1 | uint8_t | fec_scheme | Forward Error Correction scheme. 0: None, 1: Reed-Solomon. |
| 260 | 1 | uint8_t | fec_level | Percentage of parity data for the Data FEC Block (0-50). Ignored for split archives. |
| 261 | 8 | uint64_t | fec_data_offset | Absolute offset to the Data FEC Block. In a split archive, this MUST be 0. |
| 269 | 8 | uint64_t | fec_data_length | Length of the Data FEC Block. In a split archive, this MUST be 0. |
| 277 | 4 | uint32_t | header_checksum | CRC-32 (IEEE 802.3 polynomial) of the header from byte 0 to 276. |

| Offset (B) | Size (B) | Data Type | Field Name | Description |
|---|---|---|---|---|
| 281 | 50 | byte[50] | reserved | Reserved for future use. MUST be filled with null bytes (0x00). |

*Table 1*

# 4. File Index

The File Index is a data block containing a catalog of all files and directories. The index SHALL be serialized using Concise Binary Object Representation (CBOR) [RFC8949]. The root object is a CBOR array, where each element is a CBOR map representing a file entry.

The entire serialized CBOR byte stream is then compressed using Zstandard [RFC8878] and MAY be encrypted.

## 4.1. File Entry Object (CBOR Map)

Each entry in the CBOR array is a map with the following keys. Analysis of the reference implementations reveals extra fields for handling large, split files (chunking), which are included here.

| Key (string) | Value Type (CBOR) | Description |
|---|---|---|
| path | text string | Full, POSIX-style path using forward slashes (/). |
| type | text string | MUST be "file" or "directory". |
| uuid | byte string (16) | Unique 16-byte UUID for this entry. |
| creation_time | integer | int64_t creation timestamp. |
| modification_time | integer | int64_t modification timestamp. |
| permissions | unsigned integer | uint16_t POSIX-style permissions (e.g., 0o755). |
| status | unsigned integer | uint8_t entry status. 0: Normal, 1: Hidden, 2: Deleted. |
| original_filename | text string | (Files only) The original filename. |
| data_offset | unsigned integer | (Files only) uint64_t offset to the file's data within its data block. |
| data_size | unsigned integer | (Files only) uint64_t size of the file's data in bytes (after per-file compression). |
| uncompressed_size | unsigned integer | (Files only) uint64_t original size of the file in bytes. |

| Key (string) | Value Type (CBOR) | Description |
|---|---|---|
| compression | unsigned integer | (Files only) uint8_t. 0: None, 1: Zstandard. Ignored if global_compression is active. |
| checksum | unsigned integer | (Files only) uint32_t CRC-32 checksum of the uncompressed file data. |
| block_file_index | unsigned integer | (Split files) uint32_t index of the data block file (e.g., 0 for .g3fc0). For non-split archives, this is 0. |
| chunk_group_id | byte string (16) | (Split files) A 16-byte UUID shared by all chunks of a single original file. This is used to reassemble the file. |
| chunk_index | unsigned integer | (Split files) uint32_t sequential index of this chunk for a given file (0, 1, 2...). |
| total_chunks | unsigned integer | (Split files) uint32_t total number of chunks for the file this piece belongs to. |

*Table 2*

## 5. Data Blocks

The File Data section contains the actual content of the files. Its structure depends on whether the archive is split.

Non-Split Archive: All file data is concatenated into one large data block. This block is then subject to global compression and encryption as specified in the header. It is located between the File Index and the Data FEC Block.

Split Archive: The file data is chunked and written to separate files named <archive_name>.g3fc0, .g3fc1, etc. Each of these data block files is independently compressed and encrypted according to the global settings. The File Index entry's `block_file_index`, `data_offset`, and `data_size` fields are used to locate a specific chunk of data.

## 6. Compression

The G3FC format uses Zstandard (Zstd) for compression [RFC8878].

Per-File Compression: If the header's `global_compression` flag is 0, each file can be compressed individually before being added to the data block. The `compression` field in the file's index entry indicates if it was compressed.

Global Compression: If `global_compression` is 1, the entire data block (either the concatenated data in a single archive or each split block file) is compressed as a whole with Zstd. In this mode, the per-file `compression` flag is ignored.

# 7. Data Integrity and Recovery

## 7.1. Checksums

Data integrity is verified using CRC-32 checksums with the IEEE 802.3 polynomial (0xEDB88320).

- **File Data Checksum:** Each file entry in the index contains a CRC-32 checksum of its original, uncompressed data.
- **Header Checksum:** The main header contains a checksum of its own content (bytes 0-276) to detect corruption.
- **Footer Checksum:** The footer contains a checksum of its first 32 bytes to ensure its integrity.

## 7.2. Forward Error Correction (FEC)

If the `fec_scheme` in the header is 1, Reed-Solomon is used to generate parity data, allowing for recovery from corruption.

- **Data FEC Block:** Calculated over the entire File Data Block (after global compression and encryption). Its size is determined by the `fec_level` percentage. This block is only present in non-split archives.
- **Metadata FEC Block:** A separate FEC block that provides resiliency for the most critical parts of the archive. It is calculated over the concatenated bytes of the Main Header and the **uncompressed** File Index. It is stored just before the Footer for robust recovery.

# 8. Security and Encryption

## 8.1. Key Derivation

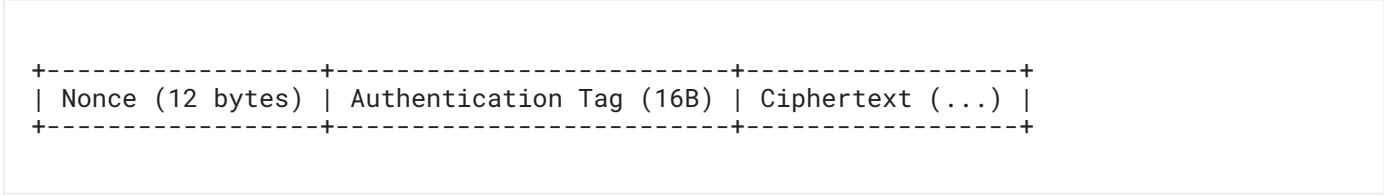
Cryptographic keys SHALL be derived from user-supplied passwords using PBKDF2 with HMAC-SHA256 as the pseudo-random function. The inputs are the password, the `read_salt` from the header, and the `kdf_iterations` count from the header. The derived key MUST be 32 bytes (256 bits) long.

## 8.2. Encryption

Data encryption SHALL be performed using AES-256 in GCM (Galois/Counter Mode). GCM provides both confidentiality and authenticity.

## 8.3. Encrypted Payload Structure

For interoperability between implementations (specifically C#, Python, and Go), the encrypted payload SHALL be structured as follows:



### 8.4. Scope of Encryption

When encryption is active (`encryption_mode` > 0), the following blocks are encrypted:

- The File Index block.
- The File Data Block (in non-split archives) or each individual Data Block file (in split archives).

The Main Header and the Footer MUST NOT be encrypted to allow for initial parsing of the archive.

## 9. Footer

A fixed-size footer of 40 bytes is located at the very end of the .g3fc file. Its purpose is to allow an application to quickly find the File Index and Metadata FEC block without scanning the entire file. This is crucial for efficient operation, as it means the application can read the last 40 bytes, validate the footer's integrity, and then seek directly to the File Index location without needing to read or decompress the (potentially very large) file data block that precedes it.

| Offset from End | Size (B) | Data Type | Field Name | Description |
|---|---|---|---|---|
| -40 | 8 | uint64_t | main_index_offset | Absolute offset to the File Index. MUST be identical to the header value. |
| -32 | 8 | uint64_t | main_index_length | Length of the File Index. MUST be identical to the header value. |
| -24 | 8 | uint64_t | metadata_fec_block_offset | Absolute offset to the Metadata FEC Block. |
| -16 | 8 | uint64_t | metadata_fec_block_length | Length of the Metadata FEC Block. |
| -8 | 4 | uint32_t | footer_checksum | CRC-32 checksum of the preceding 32 bytes of the footer. |
| -4 | 4 | char[4] | footer_magic | MUST contain the ASCII characters "G3CE" (G3 Container End). |

*Table 3*

## 10. File Operations

**Deletion:** To mark a file or directory as deleted, its `status` field in the File Index SHALL be changed to 2. The actual file data is not removed from the data blocks. This allows for "undelete" functionality. A separate "compact" or "purge" operation MAY be implemented by an application to physically remove data marked as deleted and reclaim space.

## 11. IANA Considerations

This document requests the registration of a new media type in the "application" tree, as follows:

Type name: application

Subtype name: vnd.g3pix.g3fc

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 12 of this document.

Interoperability considerations: This document specifies format version 1.0. All fields use little-endian byte order.

Published specification: This document.

Applications that use this media type: G3FC Archiver Tool and other compatible archiving utilities.

Fragment identifier considerations: N/A

Additional information: Magic number(s): The first 4 bytes are 0x47 0x33 0x46 0x43 (ASCII "G3FC").

File extension(s): .g3fc, .g3fc<n>

Macintosh file type code(s): N/A

Person & email address to contact for further information: g3fc@g3pix.com.br

Intended usage: COMMON

Restrictions on usage: None

Author: Lucas Guimaraes

Change controller: G3Pix

## 12. Security Considerations

Implementers of this specification should be aware of the following security aspects:

- **Password Strength:** The security of an encrypted archive is entirely dependent on the strength of the user's password. Implementations SHOULD encourage or enforce strong password policies.
- **KDF Iterations:** The number of PBKDF2 iterations (`kdf_iterations`) is critical for resisting brute-force attacks. The recommended value of 100,000 is a baseline and SHOULD be increased over time as computing power grows.
- **Salt:** The use of a unique, randomly generated salt for each archive is crucial to prevent rainbow table attacks. The `read_salt` MUST be cryptographically random.
- **Authenticated Encryption:** The use of AES-256-GCM is REQUIRED as it provides authenticated encryption, protecting against certain types of attacks that can be performed on unauthenticated ciphers (e.g., bit-flipping attacks).
- **Metadata Protection:** While the file content is protected, some metadata in the Main Header is not encrypted. This includes timestamps and the name of the creating software. Users should be aware that this information is visible even in an encrypted archive.

- **Path Traversal:** Implementations that extract files MUST validate and sanitize the `path` field from the File Index to prevent path traversal attacks (e.g., writing files outside the intended destination directory). Paths containing ".." or absolute paths SHOULD be rejected or handled with extreme care.
- **Compression:** The media type employs Zstandard compression. As with any format that uses compression, G3FC files are susceptible to "compression bomb" denial-of-service attacks, where a small file decompresses to an extremely large size, potentially exhausting system memory or disk space. Implementations that parse this format SHOULD mitigate this risk by first checking the uncompressed_size field in the file's metadata index and enforcing reasonable limits on resource allocation before attempting decompression.
- **Executable content:** The media type does not contain any active or executable content. The G3FC format is a container for passive data files and metadata only.

# 13. References

## 13.1. Normative References


**[RFC2119]** Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

**[RFC8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

**[RFC8949]** Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.


## 13.2. Informative References


**[RFC8878]** Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the 'application/zstd' Media Type", RFC 8878, DOI 10.17487/RFC8878, February 2021, <https://www.rfc-editor.org/info/rfc8878>.


# Contributors

The G3FC format relies on several established technologies. The authors of the specifications for these technologies are acknowledged for their foundational work.

**S. Bradner**
RFC 2119

**B. Leiba**
RFC 8174

**C. Bormann**
RFC 8949

**P. Hoffman**
RFC 8949

**Y. Collet**
RFC 8878

## Author's Address

**Lucas Guimaraes**
G3Pix
Rua Santa Clara, 1049, Centro
Braganca Paulista-Sao Paulo
12900-190
Brazil
Email: g3fc@g3pix.com.br
URI: https://g3pix.com.br/g3fc/