sgffp

On this page

SnapGene .dna File Format Specification

This document describes the binary format of SnapGene `.dna` files based on reverse engineering and analysis of files produced by SnapGene versions 5.x–7.x.

File Header

Every `.dna` file starts with a fixed 19-byte header:

Offset	Size	Value	Description
0	1	`0x09` (`\t`)	Magic byte
1	4	`0x00000E` (14)	Header length (big-endian uint32)
5	8	`SnapGene`	ASCII title
13	2	varies	`type_of_sequence` (big-endian uint16)
15	2	varies	`export_version` (big-endian uint16)
17	2	varies	`import_version` (big-endian uint16)

type_of_sequence values:

Value	Meaning
1	DNA
2	Protein
7	RNA

Versioning

`export_version` identifies the SnapGene format revision that wrote the file, and some blocks change encoding across revisions. The known cutoff is **`export_version` 15**: at and above it (SnapGene ~8.x) the history tree (block 7) is LZMA-compressed; below it (older SnapGene) block 7 is plain XML.

sgffp resolves the correct parser per block from the cookie's `export_version` — the parsers themselves stay single-purpose. On write, the file is normalized to `export_version` 15 by default (modern encodings); a preserve option keeps the original version and its block encodings so the file re-opens in the SnapGene release that produced it.

TLV Block Format

After the header, the file contains a sequence of TLV (Type-Length-Value) blocks:

Field	Size	Description
type	1 byte	Block type ID (unsigned)
length	4 bytes	Data length (big-endian uint32)
data	`length` bytes	Block payload

Blocks appear in any order. Some types can appear multiple times (e.g., block 11 for each history node, block 16 for each trace). Unknown block types should be skipped.

Block Reference

Block 0 — DNA Sequence

Uncompressed DNA sequence with property flags.

Offset	Size	Description
0	1	Property flags byte
1	N	ASCII sequence data

Property flags (bitmask):

Bit	Mask	Meaning
0	`0x01`	Circular topology (0=linear)
1	`0x02`	Double-stranded (0=single)
2	`0x04`	Dam methylated
3	`0x08`	Dcm methylated
4	`0x10`	EcoKI methylated

Block 1 — Compressed DNA Sequence

Compact encoding for sequences that may contain plain DNA, IUPAC ambiguity codes, all-N runs, and lowercase ranges. Used for snapshots inside block 11 history nodes, and (rarely) as a top-level block 1.

The payload is a chain of self-describing **sections** followed by optional lowercase range pairs. A small descriptor at the front of the block tells the decoder how many sections to read, how many lowercase ranges follow, and the type+length of the first section.

```
[cl][ul][stamp][chunks][lowercases][marker][count][section data...][lowerca
 4   4    1       4         4         1      4
```

Offset	Size	Description
0	4	`cl` — outer compressed length (big-endian uint32)
4	4	`ul` — final decoded character count (big-endian uint32)

Offset	Size	Description
8	1	`writer_stamp` — opaque writer-side byte (SnapGene's parser ignores it; preserved for round-trip)
9	4	`chunks` — section count (big-endian uint32)
13	4	`lowercases` — lowercase pair count (big-endian uint32)
17	1	`marker` — first section's type marker (0x01 / 0x02 / 0x03)
18	4	`count` — first section's character count (big-endian uint32)
22	…	section data, then any remaining sections, then lowercase pairs

Every section beyond the first is introduced by its own 5-byte frame: a 1-byte marker plus a big-endian uint32 character count, followed by the section's data bytes (if any).

Section markers:

Marker	Meaning	Data bytes
`0x01`	Plain DNA — 2-bit GATC packed	`ceil(count * 2 / 8)`
`0x02`	IUPAC ambiguity codes — 4-bit nibble per char	`ceil(count / 2)`
`0x03`	All-N run	none

2-bit DNA encoding (used inside `0x01` sections, MSB first; the final byte right-aligns a partial tail):

Bits	Base	Bits	Base
00	G	10	T
01	A	11	C

4-bit IUPAC encoding (used inside `0x02` sections, two nibbles per byte, high nibble first):

Nibble	Code	Nibble	Code
0x4	N	0xA	R

Nibble	Code	Nibble	Code
0x5	B	0xB	S
0x6	D	0xC	V
0x7	H	0xD	W
0x8	K	0xE	Y
0x9	M		

Lowercase pairs (trailing the section data): `lowercases` × 8 bytes, each pair encoding `(start, end)` as big-endian uint32s. Each pair lowercases the inclusive character range `[start..end]` of the assembled sequence.

A sequence of pure `A/C/G/T` collapses to a single `0x01` section, no lowercase pairs — the byte layout becomes equivalent to flat 2-bit packed DNA, plus the fixed-size descriptor. Files containing ambiguity codes or lowercase ranges use additional sections and pairs as needed; the same parser handles every case.

Block 5 — Primers (XML)

XML block parsed by `xmltodict`. Top-level element: `<Primers>`.

xml

```xml
<Primers nextValidID="3">
  <HybridizationParams
    minContinuousMatchLen="10"
    allowMismatch="1"
    minMeltingTemperature="40"
    showAdditionalFivePrimeMatches="1"
    minimumFivePrimeAnnealing="15"
  />
  <Primer name="Forward" sequence="ATGCATGCATGC">
    <BindingSite
      location="100-112"
      boundStrand="0"
      annealedBases="ATGCATGCATGC"
      meltingTemperature="36"
    >
      <Component hybridizedRange="100-112" bases="ATGCATGCATGC"/>
```

```
      </BindingSite>
      <BindingSite simplified="1" location="100-112" boundStrand="0"
        annealedBases="ATGCATGCATGC" meltingTemperature="36">
        <Component hybridizedRange="100-112" bases="ATGCATGCATGC"/>
      </BindingSite>
    </Primer>
  </Primers>
```

Each `<Primer>` can contain multiple `<BindingSite>` child elements. SnapGene generates both detailed and simplified (attribute `simplified="1"`) versions. The `boundStrand` attribute is `"0"` for forward/top and `"1"` for reverse/bottom.

The `location="A-B"` attribute is **0-based inclusive** — a binding site covering 0-based positions `A` through `B` inclusive, i.e. the half-open range `[A, B+1)`. This differs from feature segment `range` values, which are **1-based inclusive**. sgffp exposes both as 0-based half-open `[start, end)`, so a primer stored as `location="100-119"` parses to `start=100, end=120`.

Block 6 — Notes (XML)

File-level metadata. Top-level element: `<Notes>`.

xml

```
  <Notes>
    <UUID>550e8400-e29b-41d4-a716-446655440000</UUID>
    <Type>Synthetic</Type>
    <Description>My plasmid</Description>
    <ConfirmedExperimentally>0</ConfirmedExperimentally>
    <CustomMapLabel>plasmid.dna</CustomMapLabel>
    <UseCustomMapLabel>1</UseCustomMapLabel>
    <SequenceClass>UNA</SequenceClass>
    <Created UTC="2024-01-15.12:00:00">2024-01-15.12:00:00</Created>
    <LastModified UTC="2024-06-01.15:30:00">2024-06-01.15:30:00</LastModifiec
  </Notes>
```

Block 7 — History Tree (XML)

Contains XML with a recursive `<Node>` tree describing the cloning operation history. Root node represents the current file state; children are previous states (tree grows

backward in time).

The on-disk encoding is **version-dependent**: files with `export_version >= 15` (SnapGene ~8.x) LZMA-compress the block; older files (`export_version < 15`) store it as plain XML. sgffp selects the right parser from the cookie's export version (see Versioning).

xml

```
<HistoryTree>
  <Node ID="2" name="Final.dna" type="DNA" seqLen="5000"
        circular="1" strandedness="double" operation="insertFragment"
        upstreamModification="Unmodified" downstreamModification="Unmodifie
    <InputSummary manipulation="insert" val1="100" val2="4900"
                  name1="EcoRI" siteCount1="1" name2="BamHI" siteCount2="1"
    <Node ID="0" name="Vector.dna" type="DNA" seqLen="4000"
          circular="1" strandedness="double" operation="invalid"
          resurrectable="1" .../>
    <Node ID="1" name="Insert.dna" type="DNA" seqLen="1000"
          circular="0" strandedness="double" operation="invalid"
          resurrectable="1" .../>
  </Node>
</HistoryTree>
```

Node fields:

Field	Type	Description
`ID`	string	Unique node identifier (integer as string)
`name`	string	Sequence filename
`type`	string	`"DNA"` , `"RNA"` , or `"Protein"`
`seqLen`	string	Sequence length (integer as string)
`circular`	string	`"0"` (linear) or `"1"` (circular)
`strandedness`	string	`"single"` or `"double"`
`operation`	string	Operation that created this state (see below)
`upstreamStickiness`	string	Upstream sticky end length

Field	Type	Description
`downstreamStickiness`	string	Downstream sticky end length
`upstreamModification`	string	e.g., `"Unmodified"`, `"FivePrimePhosphorylated"`
`downstreamModification`	string	e.g., `"Unmodified"`
`resurrectable`	string	`"1"` if the user can restore this state

Attribute order convention: SnapGene expects attributes in this order: `name`, `type`, `seqLen`, `strandedness`, `ID`, `circular`, `[resurrectable]`, `operation`.

Operation types:

Operation	Category	Description
`invalid`	base	Original/imported file (leaf node)
`makeDna`	create	Created new DNA sequence
`makeRna`	create	Created new RNA sequence
`makeProtein`	create	Created new protein sequence
`amplifyFragment`	cloning	PCR amplification
`insertFragment`	cloning	Single fragment insertion
`insertFragments`	cloning	Multiple fragment insertion
`replace`	edit	Sequence edit/substitution
`digest`	cloning	Restriction digest
`ligateFragments`	cloning	Ligation of fragments
`gatewayLRCloning`	cloning	Gateway LR reaction
`gatewayBPCloning`	cloning	Gateway BP reaction
`gibsonAssembly`	cloning	Gibson assembly
`goldenGateAssembly`	cloning	Golden Gate assembly

Operation	Category	Description
`restrictionCloning`	cloning	Restriction cloning
`taCloning`	cloning	TA cloning
`topoCloning`	cloning	TOPO cloning
`inFusionCloning`	cloning	In-Fusion cloning
`flip`	edit	Reverse complement
`newFileFromSelection`	edit	Extract subsequence to new file
`primerDirectedMutagenesis`	edit	Site-directed mutagenesis
`changeMethylation`	metadata	Change methylation status
`changePhosphorylation`	metadata	Change phosphorylation status
`changeStrandedness`	metadata	Change single/double stranded
`changeTopology`	metadata	Change linear/circular topology

Nested elements:

- `<InputSummary>` — **Required on every non-leaf node** (SnapGene segfaults without it). Contains: `manipulation`, `val1`, `val2`, `name1` / `siteCount1` (enzymes). An empty `<InputSummary/>` is valid.
- `<Oligo>` — Primer used in PCR: `name`, `sequence`, `phosphorylated` (optional)
- `<Parameter>` — Key-value pairs: `name`, `val`
- `<RegeneratedSite>` — Restriction site regenerated by cloning
- `<HistoryColors>` — Strand coloring for history map display
- `<Features>` — Snapshot of features at this history state
- `<Primers>` — Primer binding sites used in this operation
- `<Node>` — Child nodes (single dict or list)

Block 8 — Sequence Properties (XML)

Additional sequence properties. Top-level element:
`<AdditionalSequenceProperties>`.

xml

```xml
<AdditionalSequenceProperties>
  <UpstreamStickiness>0</UpstreamStickiness>
  <DownstreamStickiness>0</DownstreamStickiness>
  <UpstreamModification>Unmodified</UpstreamModification>
  <DownstreamModification>Unmodified</DownstreamModification>
</AdditionalSequenceProperties>
```

Block 10 — Features (XML)

Annotation features with segment ranges, qualifiers, and strand mapping.

xml

```xml
<Features>
  <Feature name="GFP" type="CDS" directionality="1">
    <Segment range="101-820" color="#00ff00"/>
    <Q name="label"><V text="GFP"/></Q>
    <Q name="note"><V text="Green fluorescent protein"/></Q>
    <Q name="translation"><V text="MVSK..."/></Q>
  </Feature>
</Features>
```

Strand mapping (from `directionality` attribute):

Value	Strand
`0`	`.` (none)
`1`	`+` (forward)
`2`	`-` (reverse)
`3`	`=` (both)

Parsed format (after sgffp processing):

json

```json
{
    "features": [
        {
            "name": "GFP",
            "type": "CDS",
```

```
            "strand": "+",
            "start": 100,
            "end": 820,
            "color": "#00ff00",
            "segments": [{"range": "101-820", "color": "#00ff00"}],
            "qualifiers": {"label": "GFP", "note": "Green fluorescent prote
        }
    ]
}
```

`start` is 0-based (XML range `"101"` minus 1), `end` is 1-based.

Block 11 — History Nodes (Binary)

Each block 11 entry stores a sequence snapshot for one history state. Multiple block 11 entries exist (one per non-root tree node).

Binary layout:

Offset	Size	Description
0	4	`node_index` (big-endian uint32) — links to tree node `ID`
4	1	`sequence_type` (see below)
5+	varies	Sequence data (type-dependent)
...	varies	Nested TLV blocks (node content)

sequence_type values:

Value	Format	Description
0	Block 0 format	Uncompressed DNA (4B length + ASCII)
1	Block 1 format	Compressed DNA (recommended, SnapGene default)
21	Block 0 format	Protein sequence
29	—	Modifier-only (no sequence data)
32	Block 0 format	RNA sequence

Block 14 — Custom Enzyme Sets (XML)

User-defined enzyme groups. Top-level element: `<CustomEnzymeSets>`.

xml

```xml
<CustomEnzymeSets>
  <CustomEnzymeSet type="1" name="MCS Enzymes"
    enzymeNames="Acc65I AscI BamHI EcoRI HindIII KpnI NotI SmaI XmaI"/>
</CustomEnzymeSets>
```

Block 16 — Trace Container

Container wrapping a ZTR trace (block 18) with optional properties (block 8). Block 18 **never** appears as a standalone top-level block.

Multiple traces = multiple block 16 entries.

Offset	Size	Description
0	4	Flags (big-endian uint32): `0` = forward, `1` = reverse
4	N	Nested TLV blocks (block 18, optionally block 8)

Block 17 — Alignable Sequences (XML)

Sequence alignment data. Top-level element: `<AlignableSequences>`.

xml

```xml
<AlignableSequences trimStringency="Medium">
  <Sequence name="ref_seq" sequence="ATGC..."/>
</AlignableSequences>
```

Block 18 — ZTR Trace Data

Only appears inside block 16 containers.

ZTR format (Staden Package) for Sanger sequencing chromatograms.

Header:

Offset	Size	Description
0	8	Magic: `\xaeZTR\r\n\x1a\n`
8	2	Version (typically `\x01\x02`)

Chunks follow the header:

Field	Size	Description
type	4	ASCII chunk type
metadata_length	4	Metadata length (big-endian uint32)
metadata	N	Metadata bytes
data_length	4	Chunk data length (big-endian uint32)
data	N	Chunk data (may be compressed)

Compression: chunk data is not compressed once but through a *chain* of filters. Each layer prefixes its own format byte, so decoding loops until that byte is `0x00` (RAW). A typical `SMP4` chunk reads:

```
zlib -> RLE -> FOLLOW1 -> 16TO8 -> DELTA2 -> raw
```

Reading only the outermost layer leaves the rest of the chain to be misinterpreted as sample values.

Byte	Filter	Notes
`0x00`	RAW	Innermost layer; decoding stops here
`0x01`	RLE	`[1][uint32 LE length][guard]` ; a literal guard byte is escaped as `guard 0`
`0x02`	ZLIB	`[2][uint32 LE length][deflate stream]`
`0x40` – `0x42`	DELTA1/2/4	Successive differences over 1-, 2- or 4-byte words; `[fmt][level]` , DELTA4 pads to 4 bytes

Byte	Filter	Notes
`0x46` / `0x47`	16TO8 / 32TO8	Store 16/32-bit words in one byte, escaping wide values with `-128`
`0x48`	FOLLOW1	256-byte "most frequent successor" table, then residuals

Chains observed in SnapGene files: `SMP4` as above (sometimes without RLE), `BPOS` as `zlib -> 32TO8 -> DELTA4`, `CNF4` as `zlib -> RLE -> DELTA1`, `BASE` / `TEXT` as plain zlib, and `CLIP` raw.

io_lib also defines XRLE, XRLE2, CHEB445, ICHEB, LOG2, STHUFF, QSHIFT and TSHIFT. None have been observed in SnapGene output. `LOG2` is lossy by construction and `CHEB445` is disabled upstream.

Since every filter records its own parameters in the stream, a chain read from a file can be replayed exactly on write. SnapGene's zlib layer is `deflateInit2(level=1, windowBits=15, memLevel=8, Z_HUFFMAN_ONLY)`; any other level or strategy yields different bytes for identical input.

Chunk types. Each payload below begins with the RAW format byte:

Type	Description	Data format
`BASE`	Base calls	1 byte + ASCII bases
`BPOS`	Base-to-sample positions	1 byte + 3 padding + big-endian uint32 per base
`CNF4`	Confidence scores	1 byte + 4 bytes per base: the called-base score for every base, then the A/C/G/T scores
`SMP4`	Combined ACGT samples	1 byte + 1 padding + big-endian uint16, channels A, C, G, T back to back
`SAMP`	Single channel samples	Metadata: channel letter. 1 byte + 1 padding + big-endian uint16
`TEXT`	Metadata key-value pairs	1 byte + null-terminated key/value pairs, terminated by an empty identifier (a trailing extra `\0`)

Type	Description	Data format
`CLIP`	Quality clip boundaries	1 byte + left uint32 + right uint32
`COMM`	Comments	1 byte + ASCII text

Chunk order is significant for byte-identical output. SnapGene writes `SMP4, BASE, BPOS, CNF4, TEXT, CLIP`.

Block 20 — Strand Colors (XML)

Per-strand color highlighting. Top-level element: `<StrandColors>`.

xml

```xml
<StrandColors>
  <TopStrand><ColorRange range="314..354" colors="magenta"/></TopStrand>
  <BottomStrand><ColorRange range="318..358" colors="magenta"/></BottomStra
</StrandColors>
```

Block 21 — Protein Sequence

Same format as Block 0. Sequence contains amino acid single-letter codes.

Block 23 — File Attachments

Embeds arbitrary files (images, documents) inside the `.dna` file. Two sub-formats share the same block type ID:

File data block (one per attached file):

Offset	Size	Description
0	4	`file_id` (big-endian uint32, starts at 1)
4	N	Raw file bytes

Manifest block (one per file, lists all attachments):

Offset	Size	Description

Offset	Size	Description
0	4	`0x00000000` (discriminator — file IDs start at 1)
4	4	`decompressed_size` (big-endian uint32)
8	N	zlib-compressed XML

Manifest XML:

```xml
<Files>
  <File id="1" name="gel_image.jpg" size="23996" mtime="2024-06-01" compres
  <File id="2" name="protocol.pdf" size="51200" mtime="2024-06-01" compress
</Files>
```

Discrimination: if the first 4 bytes are zero, the block is a manifest; otherwise it is file data.

Block 27 — Trace Alignment (BGZF BAM)

Stores alignment of sequencing traces (block 16) against the reference sequence as BGZF-compressed BAM data.

BGZF structure: Concatenated gzip blocks, each with a BC extra field containing the block size (BSIZE). Ends with a standard 28-byte EOF marker.

Component	Description
Header block	BAM magic (`BAM\x01`) + SAM header text + reference sequences
Data block(s)	Alignment records (one per trace)
EOF block	28-byte empty gzip block (standard BAM EOF marker)

BAM header:

Offset	Size	Description
0	4	Magic `BAM\x01`
4	4	`l_text` — SAM header length (little-endian int32)

Offset	Size	Description
8	l_text	SAM header text (e.g. `@HD\tVN:1.6\tSO::unsorted\n`)
8+l_text	4	`n_ref` — number of reference sequences

Per reference: `l_name` (int32) + name (null-terminated) + `l_ref` (int32, sequence length).

Alignment record fields: `ref_id`, `pos`, `mapq`, `bin`, `flag`, CIGAR ops, 4-bit packed sequence, quality scores, auxiliary tags.

Relationship with block 17: Block 17 (`AlignableSequences` XML) holds alignment metadata — trim range, display settings. Block 27 holds the actual BAM data. They are linked by ID: block 17's `Sequence/@ID` matches block 27's BAM record `read_name`.

Block 28 — Enzyme Visibilities (XML)

Override enzyme visibility. Top-level element: `<EnzymeVisibilities>`.

xml

```xml
<EnzymeVisibilities vals=""/>
```

Block 29 — History Modifier (LZMA XML)

Metadata-only history changes. LZMA-compressed XML. Rarely present.

Block 30 — History Node Content (LZMA TLV)

LZMA-compressed nested TLV blocks inside block 11 `node_info`. Contains the full file state (features, primers, notes, etc.) at that history point.

Block 32 — RNA Sequence

Same format as Block 0. Contains RNA sequence (ACGU bases).

Block 34 — RNA Structure Predictions (LZMA JSON)

LZMA-compressed JSON with RNA secondary structure prediction data.

json

```
{
  "bondedPairs": [[22, 30], [23, 29]],
  "parameters": {"noClosingGU": false, "noLP": false, "temperature": 37},
  "probabilityMatrix": [[0, 5, 0.077]],
  "revision": 1,
  "stats": {
    "ensembleDiversity": 213.88,
    "freeEnergyEnsemble": -97.31,
    "freeEnergyOptimal": -75.7,
    "freqOptimalEnsemble": 0.0
  },
  "suboptimalStructures": [{"bondedPairs": []}]
}
```

Blocks Not Parsed

Blocks 2, 3, 13, and 35 are SnapGene enzyme-related caches/preferences. For modern files (`export_version >= 15`) sgffp skips them on read and does not emit them on write — modern SnapGene rebuilds them on demand, so stripping them is equivalent to clearing a cache.

Legacy exception: older SnapGene (`export_version < 15`) does *not* regenerate these caches, so a stripped file shows no enzymes when re-opened there. For legacy files sgffp therefore retains blocks 2, 3, 13 as raw bytes. A `preserve` write keeps them (the file re-opens with enzymes in old SnapGene); a default normalize-to-v15 write drops them (modern SnapGene regenerates). See Versioning.

Block 2 — rare optional metadata. Layout: 1-byte marker `0x01` followed by a packed array of big-endian uint32 values. Content appears to be a precomputed enzyme/feature index — exact semantics unknown but the field is preserved verbatim by SnapGene when present.

Block 3 — enzyme analysis cache. Three sections:

Section	Layout	Content
1	`0x01` + BE uint32 length=4,763 + 4,763 ASCII bytes	Global enzyme recognition database, comma-separated sites (`GACNNNNNNGTC,CCAGGCCTGG,...`). Byte-identical across files — a SnapGene-

Section	Layout	Content
		embedded constant, not file-specific.
2	`0x00` + BE uint32 length (0, 256, 512, 768, 1024, or 1792) + uint32-aligned payload	Per-enzyme selection / display state.
Tail	Variable, mostly zeros	Derived enzyme analysis (cut counts / positions) computed against the file's sequence.

Block 3 is fully derivable from `sequence + enzyme DB + user selections` and SnapGene recomputes it on first enzyme-panel access — it is a performance cache, not user data.

Block 13 — enzyme display preferences. Always exactly 345 bytes, almost entirely fixed boilerplate. Only a handful of bytes vary across files and encode the active enzyme-set name (e.g. `Unique 6+ Cutters`) plus a small struct of display flags (colours, sort order, numeric vs sequence view). The ASCII name lives at offset 18 with a 1-byte length prefix.

Block 35 — LZMA-compressed empty JSON array (`[]`) in every observed file; purpose unconfirmed. Treated as an auto-generated cache like blocks 2/3/13.

SnapGene's CLI `--convert SnapGene` (normalize) does not re-add these blocks when sgffp strips them. They appear only when the user actively interacts with SnapGene's enzyme panel in the UI.

Edit this page

Released under the MIT License.