QBIN Specification (v1.0-draft)

Status: DRAFT
Version: 1.0 (file format major=1, minor=0)
Date: 2025-08-25

QBIN is a compact, binary container for OpenQASM circuits. It encodes gates and operands as typed binary records to minimize size and parsing overhead, while remaining extensible and round-trip safe to and from OpenQASM (2.x/3.x).

This document defines:

- File layout and section catalog
- Types, encodings, and alignment
- Instruction set and operand model
- Metadata and debugging facilities
- Versioning and compatibility
- Validation rules and error codes

Non-goals for v1 (can arrive in v1.1+ or v2):

- Full general-purpose program bytecode (loops, call stacks, heap)
- Arbitrary user-defined classical compute beyond simple compares

---

1. Terminology

- QASM: OpenQASM source text.
- QBIN: This binary format.
- Section: A typed region within the file.
- Varint: Unsigned LEB128 variable-length integer.
- ULEB128: Unsigned LEB128.
- SLEB128: Signed LEB128.
- BE/LE: Big-Endian / Little-Endian.
- Qubit index: Zero-based integer identifying a qubit.
- Bit index: Zero-based integer identifying a classical bit.

---

2. File overview

A QBIN file consists of:

```
Header (fixed-size)
Section Table (array of entries)
Sections... (payloads, each 8-byte aligned)
```

Sections are independent typed blobs, aligned to 8 bytes. This enables:

- Streaming decode of the instruction stream
- Lazy reading of metadata, strings, debug info
- Forward/backward compatibility via unknown-section skipping

---

3. Endianness and numeric conventions

- All fixed-width integers are Little-Endian (LE) unless a section explicitly overrides.
- Floating-point values in v1 are IEEE-754 binary32 (float32, LE).
- Angles are in radians.
- Complex numbers (if present in metadata) are encoded as two float32 values: real, imag.

---

4. File header

Offset	Size	Field
0x00	4	Magic = 0x51 0x42 0x49 0x4E (ASCII "QBIN")
0x04	1	Major version (0x01)
0x05	1	Minor version (0x00)
0x06	1	Flags:bit0 endianness (0=LE, 1=BE; v1 MUST be 0)bit1 section_table_hash_presentbits2..7
0x07	1	Header size (bytes). v1 = 24
0x08	4	Section count (u32, LE)
0x0C	4	Section table offset (u32, LE; bytes from file start)
0x10	4	Section table size (u32, LE)
0x14	4	Header CRC32 (u32, LE) over bytes [0x00..0x13]

Notes:

- Readers MUST verify magic and major_version.
- If flags.bit1 == 1, the Section Table trailer (see 5.3) includes a hash.

---

5. Section table

Each entry describes one section.

5.1 Section ID space

Section IDs are 32-bit tags. Recommended core set:

ID (u32)	ASCII	Meaning
0x4D455441	META	Metadata (key-value, binary)
0x53545253	STRS	String table (UTF-8)
0x51554253	QUBS	Qubit table
0x42495453	BITS	Classical bit table
0x50415253	PARS	Parameter table
0x47415445	GATE	Gate table
0x494E5354	INST	Instruction stream (**mandatory**)
0x44454247	DEBG	Debug info
0x5349474E	SIGN	Detached signature
0x43505253	CPRS	Compression dictionary (optional)
0x45585453	EXTS	Extensions registry

Vendor sections: use 0x56xxxxxx ("Vxxx") and namespace keys in META (e.g., "com.acme.*") to avoid collisions.

5.2 Section table entry

Offset	Size	Field
0x00	4	Section ID (u32)
0x04	4	Section offset (u32, LE; 8-byte aligned)
0x08	4	Section size (u32, LE)
0x0C	4	Section flags (u32): bit0 compressed, bit1 checksummed, others reserved

The table consists of `section_count` entries.

5.3 Section table trailer (optional)

Present only if header.flags.bit1 == 1:

Offset	Size	Field
+0x00	4	Hash algorithm (1=CRC32C, 2=xxh3_64)
+0x04	8	Hash value (u64; for CRC32C store in low 32 bits)

---

6. Types and encodings

- `u8/u16/u32/u64`: unsigned integers (LE).
- `varint`: ULEB128 (non-negative integers: indices, counts).
- `svarint`: SLEB128 (signed immediates).
- `f32`: IEEE-754 float32 (LE).

- `bool`: u8 (0 or 1).
- Offsets: u32 LE from file start unless stated otherwise.
- Strings (STRS): UTF-8, zero-terminated, deduplicated, referenced by varint string_id.

---

7. Core sections

7.1 STRS (String Table)

Purpose: deduplicate human-readable strings (names, metadata keys).

Layout:

```
u32 magic = 0x53545253 ("STRS")   // inside payload for self-check
u32 count
for i in [0..count):
  varint byte_len
  u8[byte_len] utf8_bytes
  u8 0x00  // terminator, not counted in byte_len
```

String IDs are implicit, 0..count-1. ID 0 SHOULD be the empty string "".

7.2 META (Metadata)

Purpose: small, structured metadata without requiring a JSON parser.

Encoding (compact KV map):

```
u32  magic = "META"
varint pair_count
repeat pair_count:
  varint key_str_id
  u8     value_type   // 0=nil, 1=bool, 2=varint, 3=svarint, 4=f32, 5=str_id, 6=blob
  value  as per type:
         bool:   u8
         varint: ULEB128
         svarint:SLEB128
         f32:    4 bytes
         str_id: varint
         blob:   varint len + u8[len]
```

Recommended keys (by convention, STRS entries): `qasm.version` ("2.0" or "3.0"), `source.name` (filename), `generator` (tool name/version), `target` (backend name), `created.utc` (ISO-8601). Vendors SHOULD namespace keys (e.g., `com.acme.foo`).

7.3 QUBS (Qubit Table) [optional]

Purpose: map logical indices to named registers/ranges; optional geometry.

```
u32  magic = "QUBS"
varint qubit_count
u8    layout_present  // 0/1
if layout_present:
  repeat qubit_count:
    f32 x, f32 y, f32 z  // optional coordinates
varint alias_count
repeat alias_count:
  varint first_index
  varint count
  varint name_str_id  // e.g., "q"
```

Without QUBS, qubits are [0..N) as referenced by INST.

7.4 BITS (Classical Bit Table) [optional]

```
u32  magic = "BITS"
varint bit_count
varint alias_count
repeat alias_count:
  varint first_index
  varint count
  varint name_str_id  // e.g., "c"
```

7.5 PARS (Parameter Table) [optional]

Purpose: define symbolic parameters that may appear in instructions.

```
u32  magic = "PARS"
varint param_count
repeat:
  varint name_str_id       // e.g., "theta"
  u8     kind              // 0=angle(rad), 1=scalar, 2=duration(ns)
  u8     value_tag         // 0=unbound, 1=const_f32, 2=expr_ref
  if value_tag==1: f32 value
  if value_tag==2: varint expr_id  // future: expression DAG in EXTS
```

7.6 GATE (Gate Table) [optional]

Purpose: register custom gate signatures for decompile/validation.

```
u32  magic = "GATE"
varint decl_count
repeat:
  varint name_str_id       // gate name
```

```
  varint num_qubits
  varint num_params
  u8     flags             // bit0: opaque, bit1: unitary-known, etc.
  varint body_len
  u8[body_len] body_qbin   // optional inline QBIN sub-stream implementing gate
```

7.7 INST (Instruction Stream) [mandatory]

Purpose: the program body, a linear stream of encoded instructions.

7.7.1 Layout

```
u32   magic = "INST"
varint instr_count
repeat instr_count:
  u8   opcode
  u8   operand_mask        // bitmask indicates which operand slots follow
  ...  operands            // present iff corresponding mask bit is 1
```

Operand slots (in order), presence indicated by operand_mask bits:

```
bit0: qubit_a      (varint)
bit1: qubit_b      (varint)
bit2: qubit_c      (varint)
bit3: angle_0      (tagged: f32 or param_ref)
bit4: angle_1      (tagged: f32 or param_ref)
bit5: angle_2      (tagged: f32 or param_ref)
bit6: param_ref    (varint param_id)  // for gates with 1+ params or CALLG gate_id
bit7: aux_u32      (u32)              // misc (e.g., duration ns or bit index)
```

Angle slot encoding:

```
u8 tag    // 0 = literal f32, 1 = param_ref
if tag==0: f32 value
if tag==1: varint param_id
```

7.7.2 Opcodes (v1 core)

Single-qubit:

```
0x01 X      qubit_a
0x02 Y      qubit_a
0x03 Z      qubit_a
0x04 H      qubit_a
0x05 S      qubit_a
0x06 SDG    qubit_a
0x07 T      qubit_a
0x08 TDG    qubit_a
0x09 SX     qubit_a
0x0A SXDG   qubit_a
0x0B RX     qubit_a, angle_0
```

```
0x0C RY     qubit_a, angle_0
0x0D RZ     qubit_a, angle_0
0x0E PHASE  qubit_a, angle_0
0x0F U      qubit_a, angle_0, angle_1, angle_2
```

Two-qubit and multi:

```
0x10 CX     qubit_a (control), qubit_b (target)
0x11 CZ     qubit_a, qubit_b
0x12 ECR    qubit_a, qubit_b
0x13 SWAP   qubit_a, qubit_b
0x14 CSX    qubit_a, qubit_b
0x15 CRX    qubit_a, qubit_b, angle_0
0x16 CRY    qubit_a, qubit_b, angle_0
0x17 CRZ    qubit_a, qubit_b, angle_0
0x18 CU     qubit_a, qubit_b, angle_0, angle_1, angle_2

0x20 RXX    qubit_a, qubit_b, angle_0
0x21 RYY    qubit_a, qubit_b, angle_0
0x22 RZZ    qubit_a, qubit_b, angle_0
```

IO and barriers:

```
0x30 MEASURE  qubit_a, aux_u32(bit_index)  // measure q -> c[bit_index]
0x31 RESET    qubit_a
0x32 BARRIER  (no operands)
```

Timing and frame:

```
0x38 DELAY   qubit_a, aux_u32(duration_ns)
0x39 FRAME   qubit_a, angle_0  // frame change / phase ref
```

Custom gate call:

```
0x40 CALLG   param_ref(gate_id), qubit operands as per GATE entry
// operand_mask MUST include bit6 for gate_id, plus qubit bits (0..2) as needed
```

Reserved ranges:

- 0x80..0xBF: conditional and control flow (v1.1+)
- 0xC0..0xFF: vendor extensions (gated by EXTS)

7.7.3 Classical conditions (v1 minimal) A light-weight conditional wrapper encoded inline:

```
0x81 IF_EQ   aux_u32(bit_index), u8(value 0/1)
0x82 IF_NEQ  aux_u32(bit_index), u8(value 0/1)
0x8F ENDIF   (no operands)
```

Semantics:

- IF_* starts a guard that applies to subsequent instructions until ENDIF.

- Guards can nest; max nesting depth is implementation-defined but SHOULD be >= 8.
- Maps to: `if (c[bit_index] == value) { ... }` in QASM3.

(Full while/for/switch are deferred to v1.1+ EXTS.)

---

8. Compression

If `section_flags.bit0` (compressed) is set, the section payload is wrapped:

```
u32 comp_magic = "CPRZ"
u8  alg        // 1=zstd, 2=lz4, 3=deflate
u32 raw_size
u8[] compressed_blob
```

Only the payload is compressed; the Section Table remains uncompressed.

---

9. Integrity and signatures

9.1 Section-level checksum (optional)

If a section entry has `section_flags.bit1` set, append:

```
u32 checksum_kind   // 1=CRC32C
u32 checksum_value
```

Checksum covers the uncompressed payload bytes.

9.2 Detached signature (SIGN)

SIGN covers:

- header (0x00..0x13)
- section table bytes (including trailer if present)
- every section payload (uncompressed form)

Layout:

```
u32 magic = "SIGN"
u8  sig_kind   // 1=CMS, 2=COSE_Sign1
varint sig_len
u8[sig_len] sig_bytes
```

---

10. Versioning and compatibility

- Readers MUST accept unknown sections by skipping via the Section Table.
- Readers MUST report unknown opcodes via `ERR_UNSUPPORTED_OPCODE` unless EXTS declares a decode hook.
- Minor versions (1.x) MUST NOT break existing encodings.
- Major version (2.0) may introduce breaking changes (distinct major_version value).

META SHOULD include:

```
qbin.version.major = 1
qbin.version.minor = 0
```

---

11. Validation rules

A file is valid if:

1. Magic, versions, and header_crc32 are correct.
2. Section Table ranges are non-overlapping, 8-byte aligned, in-bounds.
3. INST section is present exactly once.
4. If QUBS/BITS are present, indices in INST are within declared counts.
5. If GATE is present and CALLG used, gate_id exists and operand counts match.
6. Angle and parameter references are well-typed.
7. Checksums (if present) validate.
8. Compression wrappers (if any) decode to declared raw_size.

Recommended additional checks:

- No NaN angles (or normalize to finite by policy).
- Reasonable nesting depth for IF/ENDIF.
- Optional max sizes (guard rails) to prevent resource abuse.

---

12. Error codes

Implementations SHOULD surface canonical errors:

```
0x01 ERR_MAGIC_OR_VERSION
0x02 ERR_HEADER_CRC
0x03 ERR_SECTION_TABLE_RANGE
0x04 ERR_MISSING_INST
0x05 ERR_MULTIPLE_INST
0x06 ERR_SECTION_CHECKSUM
0x07 ERR_DECOMPRESSION
```

```
0x08 ERR_TRUNCATED_SECTION
0x09 ERR_UNSUPPORTED_OPCODE
0x0A ERR_BAD_OPERAND_MASK
0x0B ERR_QUBIT_OOB
0x0C ERR_BIT_OOB
0x0D ERR_GATE_ID_OOB
0x0E ERR_PARAM_ID_OOB
0x0F ERR_GUARD_NESTING
0x10 ERR_TYPE_MISMATCH
0x11 ERR_META_FORMAT
```

---

13. Round-trip mapping to OpenQASM

13.1 QASM 3.0 example

QASM:

```
OPENQASM 3.0;
qubit[2] q;
bit[2] c;

h q[0];
cx q[0], q[1];
c[1] = measure q[1];
if (c[1] == 1) { x q[0]; }
```

QBIN INST (pseudo):

```
magic="INST"
count=6
1) H       a=0
2) CX      a=0, b=1
3) MEASURE a=1, aux(bit_index=1)
4) IF_EQ   aux(bit_index=1), value=1
5) X       a=0
6) ENDIF
```

13.2 Parameters

QASM:

```
cx q[0], q[1];
rz(pi/4) q[0];
rx(theta) q[1];
```

QBIN:

- RZ with angle_0 literal f32 = 0.785398163...

- RX with angle_0 param_ref to PARS entry ”theta”.

---

14. Debug info (DEBG)

Optional per-instruction mapping to original source:

```
u32   magic = "DEBG"
varint file_count
repeat file_count:
  varint path_str_id
varint mapping_count
repeat mapping_count:
  varint instr_index
  varint file_id
  varint line
  varint column
```

---

15. Extensions (EXTS) [for v1.1+]

If present, registers:

- Additional opcodes with operand schemas
- Expression DAGs for PARS (symbolic math)
- Vector operands (qubit lists) and multi-control gates

EXTS is a TLV catalog. Tools MAY ignore unknown tags.

---

16. Security considerations

- Treat all inputs as untrusted; cap sizes and counts.
- Decompression bombs: enforce raw_size and compression ratio limits.
- Signature verification: rely on standard CMS/COSE stacks.
- Avoid executing embedded GATE bodies unless explicitly allowed.

---

17. Reference values

17.1 CRC32C (Castagnoli)

- Polynomial: 0x1EDC6F41
- Init: 0xFFFFFFFF
- XorOut: 0xFFFFFFFF

17.2 Angle normalization

- Writers SHOULD normalize angles to [-pi, pi) to aid deduplication.
- Readers MUST accept any finite float32.

---

18. Test vectors (informative)

Bell state minimal:

- META:
  - qasm.version = "3.0"
  - generator = "qbin-compiler 0.1"
- QUBS: qubit_count = 2
- INST:
  1) H a=0
  2) CX a=0, b=1

Expected sizes (indicative):

- QASM text: ~40-60 bytes
- QBIN (uncompressed): ~24-48 bytes
- QBIN (zstd): often <= 24 bytes

---

19. Conformance

A conforming encoder:

- Produces exactly one INST section.
- Emits a valid header and table.
- Ensures indices and counts are in range.
- Optionally emits STRS/META for better decompile fidelity.

A conforming decoder:

- Validates header and table.
- Supports all v1 core opcodes and operand encodings.
- Skips unknown sections and reports unknown opcodes.

---

20. License