

Bovnar Conformance Test Tool

Bovnar (BVNR) v1.0 documentation · Conformance Test Tool · 2026-06-01

Version: 1.0 **Protocol:** bvnr-conformance-v1 **Last updated:** 2026-06-01

Table of Contents

1. Purpose
2. Quick Start
3. Architecture
4. Building
5. Running
6. IUT Protocol
7. Writing a Compliant IUT Adapter
8. Test Case Corpus
9. Output Format (TAP)
10. Extending the Corpus
11. CMake Integration

1. Purpose

The Bovnar Conformance Test Tool (`bvnr_conformance`) verifies that any implementation of the Bovnar serialization format produces correct, spec- compliant behaviour. The reference implementation (this repository) is used both as the test driver and as the oracle against which candidate implementations are judged.

Two use modes are supported:

Mode	Description
Self-test (default)	Runs the corpus against the reference libbvnr directly
IUT test (<code>--iut</code>)	Invokes an external binary and compares its output to the reference

2. Quick Start

Self-test (verify the reference implementation)

```
cd build
cmake --build . --target bvnr_conformance
ctest -R bvnr_conformance_self --output-on-failure
```

Testing an external implementation

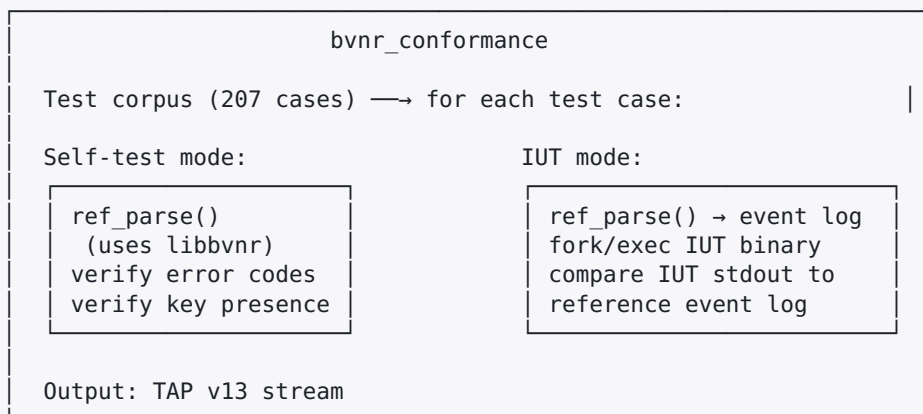
```
# Build your adapter (see Section 7)
cc -o my_impl_adapter my_adapter.c -lmy_bovnar

# Run the conformance suite against it
./tests/bvnr_conformance --iut ./my_impl_adapter
```

Filtering by group

```
./tests/bvnr_conformance --filter units
./tests/bvnr_conformance --iut ./my_impl_adapter --filter arrays
```

3. Architecture



The reference implementation is the single authoritative oracle. An implementation is conformant when its IUT adapter produces output byte-for-byte identical to the reference for every test case.

Validation tiers

Most cases exercise the **streaming reader** (`bvnr_read`): the lexer, validator, and the `on_verified` event stream the IUT protocol mirrors. A smaller set — the `homogeneity` group — exercises the **materialised-document (DOM) tier** (`bvn_dom_parse`), because the spec-1.0 array-homogeneity (§7.4), struct-shape, and duplicate-key (§8.1) rules are

enforced *above* the lexer and are therefore unreachable through the streaming `on_verified` callback. These DOM-tier cases run in **self-test mode only**; under `--iut` they are reported as `# SKIP`, since IUT protocol v1 is streaming-only and cannot express a DOM-tier check. An implementation that targets full spec-1.0 conformance must still enforce these rules in its document/tree API; the self-test cases pin the reference behaviour and its frozen error codes (39, 40, 41).

4. Building

The conformance tool is built automatically as part of the standard CMake build. Both the main driver and the reference IUT adapter are compiled:

```
mkdir build && cd build
cmake ..
cmake --build .
```

Targets produced:

Target	Binary	Purpose
<code>bvnr_conformance</code>	<code>tests/bvnr_conformance</code>	Main conformance driver
<code>bvnr_conformance_iut</code>	<code>tests/bvnr_conformance_iut</code>	Reference IUT adapter

The conformance tool links against `bvnr_static`. No external dependencies beyond `libc` and `POSIX` are required.

5. Running

Command-line options

```
bvnr_conformance [OPTIONS]
```

Options:

```
--iut <binary>  Path to the IUT binary to test
--filter <group> Run only cases in the specified group
--list           List all test case IDs and descriptions
--verbose       Print additional diagnostic information
--help          Show this help and exit
```

Examples

```
# Run self-test (reference vs. reference)
./tests/bvnr_conformance

# Run with verbose output
./tests/bvnr_conformance --verbose

# List all test cases
./tests/bvnr_conformance --list

# Run only unit-related tests
./tests/bvnr_conformance --filter units

# Test an external IUT adapter
./tests/bvnr_conformance --iut ./my_adapter

# Test external adapter, units group only
./tests/bvnr_conformance --iut ./my_adapter --filter units
```

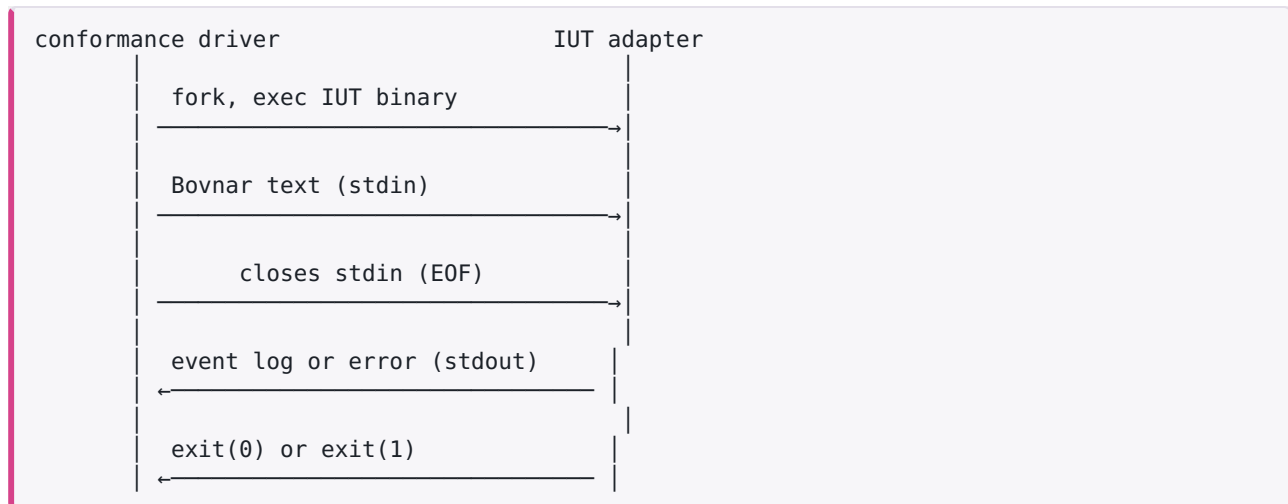
Test groups

Group	Description
encoding	UTF-8, BOM handling, byte class enforcement
identifiers	Key syntax, length limits
strings	Escape sequences, concatenation, length limits
numbers	Integer and float literals, scientific notation
types	All type families, widths, bases
default_synthesis	Automatic type annotation inference
symbols	Bare-word values
references	&.path syntax
null_values	Null scalars and null array elements
structs	Nesting, empty structs, struct arrays
arrays	1D, 2D, nested, typed, null elements
octet_streams	Binary chunk protocol
units	SI, IEC, compound, inline suffix, mismatch errors
special_numbers	nan, inf, ninf
roundtrip	Multi-assignment sequences
recovery	Error-resync behaviour
comments	Comment parsing
whitespace	Whitespace tolerance
homogeneity	DOM-tier: array homogeneity, struct shape, key uniqueness (self-test only)

6. IUT Protocol

The **IUT (Implementation Under Test) Protocol** version 1 (`bvnr-conformance-v1`) defines the interface between the conformance driver and a candidate implementation's adapter binary.

Communication model



Success response

The IUT must:

1. Exit with code **0**.
2. Write the conformance event log to stdout with **no trailing garbage**.

The event log is a sequence of lines, one event per line, in the order the events were received from the `on_verified` callback.

Error response

The IUT must:

1. Exit with code **non-zero** (typically 1).
2. Write exactly one line to stdout: `ERROR <code_name>` followed by `\n`.

Where `<code_name>` is the value returned by `bvn_error_to_string()` for the first error encountered, e.g. `value_out_of_range`.

7. Writing a Compliant IUT Adapter

The file `tests/bvnr_conformance_iut.c` is the reference IUT adapter. It uses the reference libbvnr and is intended both for self-testing and as a template for third-party implementors.

A minimal conforming adapter must:

1. **Read all of stdin** into a buffer.
2. **Parse the buffer** using the implementation under test.
3. **Collect events** from the `on_verified` callback.
4. **Emit the event log** to stdout on success.
5. **Emit** `ERROR <code_name>\n` to stdout and exit non-zero on failure.

Event log format reference

Each event maps to one line. Text fields use `\xNN` escaping for any byte outside the printable ASCII range `0x20–0x7E` or for the backslash character `0x5C`.

```
STREAM_START
ASSIGNMENT_START <key>
TYPE_ANN_START <family>
TYPE_FAMILY <family>
TYPE_PARAM_WIDTH <N>
TYPE_PARAM_BASE <N>
TYPE_PARAM_Q <N>
TYPE_PARAM_UNIT <unit>
TYPE_ANN_END <family>
DATA <token_type> <value>
STRUCT_START
STRUCT_END
ARRAY_ROW_START
ARRAY_ROW_END
ARRAY_DIM_START
OCTET_STREAM_START
OCTET_STREAM_END
```

Field details

Line	Fields	Notes
<code>STREAM_START</code>	—	Always first
<code>ASSIGNMENT_START <key></code>	key: raw key bytes, safe-escaped	
<code>TYPE_ANN_START <family></code>	family: <code>uint</code> , <code>sint</code> , <code>float</code> , <code>float_fix</code> , <code>float_dec</code> , <code>utf8</code> , <code>bool</code>	
<code>TYPE_FAMILY <family></code>	Same as <code>TYPE_ANN_START</code>	
<code>TYPE_PARAM_WIDTH <N></code>	N: effective width (0 → 64)	Only for numeric types
<code>TYPE_PARAM_BASE <N></code>	N: effective base (0 → 10)	Only for numeric types
<code>TYPE_PARAM_Q <N></code>	N: Q parameter	Only for <code>float_fix</code>
<code>TYPE_PARAM_UNIT <unit></code>	unit: unit string, safe-escaped	Only for numeric types

Line	Fields	Notes
TYPE_ANN_END <family>	Same as TYPE_ANN_START	
DATA <token_type> <value>	token_type: see below; value: safe-escaped	
STRUCT_START	—	
STRUCT_END	—	
ARRAY_ROW_START	—	
ARRAY_ROW_END	—	
ARRAY_DIM_START	—	Emitted between /-separated rows
OCTET_STREAM_START	—	
OCTET_STREAM_END	—	

TOKEN_TYPE values for DATA lines

Token type	String
token_is_number	number
token_is_string	string
token_is_symbol	symbol
token_is_reference	reference
token_is_array_number	array_number
token_is_array_string	array_string
token_is_null_value	null
token_is_octet_stream	octets
token_is_bool	bool

For `octets` token type, the value field is `<N> bytes` (decimal byte count, then a space, then the literal string `bytes`), not the raw binary data.

Effective width and base

- **Effective width:** if the stored width is 0, emit `64`. Use `bvn_effective_width(value_type)`.
- **Effective base:** if the stored base is 0, emit `10`. Use `bvn_effective_base(value_type)`.

These rules ensure that untyped values synthesised to `uint:64, 10` produce `TYPE_PARAM_WIDTH 64` and `TYPE_PARAM_BASE 10`, matching the reference output exactly.

Example traces

Input: `.x = 42;`

```

STREAM_START
ASSIGNMENT_START x
TYPE_ANN_START uint
TYPE_FAMILY uint
TYPE_PARAM_WIDTH 64
TYPE_PARAM_BASE 10
TYPE_PARAM_UNIT no_unit
TYPE_ANN_END uint
DATA number 42

```

Input: `.s = "hello";`

```

STREAM_START
ASSIGNMENT_START s
TYPE_ANN_START utf8
TYPE_FAMILY utf8
TYPE_ANN_END utf8
DATA string hello

```

Input: `.v = <float:64,m/s> 9.81;`

```

STREAM_START
ASSIGNMENT_START v
TYPE_ANN_START float
TYPE_FAMILY float
TYPE_PARAM_WIDTH 64
TYPE_PARAM_BASE 10
TYPE_PARAM_UNIT m/s
TYPE_ANN_END float
DATA number 9.81

```

Input: `.x = ok;` (symbol)

```

STREAM_START
ASSIGNMENT_START x
DATA symbol ok

```

Input: `.a = [1, 2];`

```

STREAM_START
ASSIGNMENT_START a
ARRAY_ROW_START
TYPE_ANN_START uint
TYPE_FAMILY uint
TYPE_PARAM_WIDTH 64
TYPE_PARAM_BASE 10
TYPE_PARAM_UNIT no_unit
TYPE_ANN_END uint
DATA array_number 1
TYPE_ANN_START uint
TYPE_FAMILY uint
TYPE_PARAM_WIDTH 64
TYPE_PARAM_BASE 10
TYPE_PARAM_UNIT no_unit
TYPE_ANN_END uint
DATA array_number 2
ARRAY_ROW_END

```


Input: `.x = <uint:8> 999;` (error — value out of range)

ERROR value_out_of_range

(exit code 1)

8. Test Case Corpus

The corpus is embedded in `tests/bvnr_conformance.c`. Each case specifies:

Field	Description
<code>id</code>	Unique identifier, e.g. <code>TYP-019</code>
<code>group</code>	Group name for <code>--filter</code>
<code>description</code>	Human-readable description
<code>input</code>	Bovnar text (or binary) to parse
<code>expect</code>	<code>CF_VALID</code> or <code>CF_ERROR</code>
<code>expected_error</code>	Error code for <code>CF_ERROR</code> cases
<code>continue_on_error</code>	Whether the reader should resync
<code>max_*</code>	Limit overrides (0 = use defaults)
<code>expect_key</code>	Optional: key name expected in event log

Coverage summary

Group	Cases	What is tested
<code>encoding</code>	9	UTF-8 validity, BOM placement, byte classes
<code>identifiers</code>	11	Syntax, body characters, length limits
<code>strings</code>	17	Escapes, concatenation, UTF-8, limits
<code>numbers</code>	16	Integer, float, scientific, special numbers
<code>types</code>	44	All seven type families, widths, bases, errors
<code>default_synthesis</code>	8	Auto-type inference rules
<code>symbols</code>	6	Bare-word values and limits
<code>references</code>	4	Dotted paths and limits
<code>null_values</code>	5	Null in all positions
<code>structs</code>	7	Nesting, empty, unmatched braces
<code>arrays</code>	19	1D, 2D, nested, typed, null, limits, /-row size consistency
<code>octet_streams</code>	4	Single/multi-chunk, sync errors
<code>units</code>	23	SI/IEC prefixes, compound, inline, errors
<code>special_numbers</code>	5	<code>nan</code> , <code>inf</code> , <code>ninf</code>

Group	Cases	What is tested
roundtrip	5	Multi-assignment correctness
recovery	2	Error-resync: valid data after error
comments	6	Comment styles
whitespace	4	Whitespace tolerance
homogeneity	12	DOM-tier: array homogeneity (§7.4), struct shape, key uniqueness (§8.1) — self-test only
Total	207	

9. Output Format (TAP)

The tool emits **TAP version 13** (Test Anything Protocol), which is consumed natively by CTest and many CI systems.

```
TAP version 13
1..207
ok 1 - [ENC-001] empty stream
ok 2 - [ENC-002] UTF-8 BOM at byte 0
not ok 3 - [ENC-003] UTF-8 BOM after first comment
---
message: expected error invalid_byte_order_mark but got none
...
```

Exit code is **0** when all tests pass, **1** when any test fails.

10. Extending the Corpus

To add a new conformance test, add an entry to the `g_cases[]` array in `tests/bvnr_conformance.c` using one of the provided macros:

```

/* Valid input – no expected error */
VALID("GRP-NNN", "group_name", "description of the test",
      ".input = value;"),

/* Valid input – verify a specific key and value appear */
VALID_KEY("GRP-NNN", "group_name", "description",
          ".key = value;", "key", "value"),

/* Error – no special limits */
ERROR_CASE("GRP-NNN", "group_name", "description",
           ".bad = <uint:8> 999;",
           error_value_out_of_range),

/* Error – with continue_on_error (resync mode) */
ERROR_CONT("GRP-NNN", "group_name", "description",
           ".a = 1; .bad = <uint:8> 999; .b = 2;",
           error_value_out_of_range),

/* Error – with custom limits (max_id, max_str, max_num,
                               max_struct_nesting, max_array_nesting,
                               max_array_items) */
ERROR_LIM("GRP-NNN", "group_name", "description",
          ".long_name = 1;",
          error_identifier_too_long,
          4 /*max_id*/, 0, 0, 0, 0, 0),

/* Materialised-document (DOM) tier – validated via bvn_dom_parse instead of
   the streaming reader; runs in self-test mode only (skipped under --iut).
   Use for the spec-1.0 homogeneity / struct-shape / duplicate-key rules. */
DOM_VALID("GRP-NNN", "group_name", "description",
          ".a = [1, 2.5, 3];"),
DOM_ERROR("GRP-NNN", "group_name", "description",
          ".a = [1, \"two\"];",
          error_array_element_type_mismatch),

```

After editing, rebuild:

```
cmake --build build --target bvnr_conformance
```

11. CMake Integration

Two CTest tests are registered:

CTest name	What it runs
bvnr_conformance_self	Conformance suite against the reference implementation
bvnr_conformance_iut_self	Conformance suite using the IUT adapter as the external binary

Both are in the `conformance` label group and can be run with:

```
ctest -L conformance --output-on-failure
```

The `bvnr_conformance_iut_self` test validates the IUT adapter itself: it must produce bit-for-bit identical output to the internal reference path for every valid test case.

To run all tests including conformance:

```
ctest --output-on-failure
```