

# STUNIR CLI Standardization Guide v1.0

**Purpose:** Define consistent command-line interfaces across all STUNIR tool implementations (SPARK, Python, Rust, Haskell)

**Status:** Production Ready

**Last Updated:** January 31, 2026

## Table of Contents

1. Overview
2. Core Principles
3. Command Structure
4. Tool Specifications
5. Implementation Status
6. Examples
7. Migration Guide

## 1. Overview

STUNIR provides deterministic code generation through a two-stage pipeline:

```
Spec File [spec_to_ir] IR JSON [ir_to_code] Target Code
```

Each stage has a standardized CLI for consistency across language implementations.

## 2. Core Principles

### 2.1 Consistency

- All implementations should accept the same core arguments
- Exit codes, output formats, and error messages should be uniform
- Help text format should be consistent

### 2.2 Simplicity

- Minimal required arguments
- Sensible defaults
- Clear, actionable error messages

### 2.3 Composability

- Standard input/output for piping
- JSON output for machine parsing
- Exit codes for scripting

## 2.4 Verbosity Control

- Quiet mode for CI/CD ( `-q/--quiet` )
- Verbose mode for debugging ( `-v/--verbose` )
- Standard error for logging (never pollute stdout)

## 3. Command Structure

### 3.1 General Format

```
stunir_<tool>_<lang> [OPTIONS] <REQUIRED_ARGS>
```

#### Naming Convention:

- `stunir_spec_to_ir_<lang>` - Spec to IR converter
- `stunir_ir_to_code_<lang>` - IR to code emitter
- `<lang>` = `spark`, `python`, `rust`, `haskell`

### 3.2 Standard Options (All Tools)

Option	Long Form	Description	Default
<code>-h</code>	<code>--help</code>	Show help message	N/A
<code>-V</code>	<code>--version</code>	Show version info	N/A
<code>-q</code>	<code>--quiet</code>	Suppress non-essential output	false
<code>-v</code>	<code>--verbose</code>	Enable debug logging	false
<code>-o FILE</code>	<code>--output FILE</code>	Output file path	stdout

### 3.3 Exit Codes

Code	Meaning	Usage
0	Success	Operation completed without errors
1	General error	Input validation, parsing, or execution failure
2	Invalid arguments	Missing required args or invalid options
3	File I/O error	Cannot read input or write output
4	Schema validation error	IR does not conform to stunir_ir_v1
130	SIGINT (Ctrl+C)	User interrupted execution

## 4. Tool Specifications

### 4.1 spec\_to\_ir - Spec to IR Converter

#### Purpose

Convert specification files (JSON, YAML, etc.) into canonical STUNIR IR (JSON).

#### Standard CLI (Recommended)

```
stunir_spec_to_ir [OPTIONS] --spec-root <DIR>
```

#### Required Arguments:

- --spec-root <DIR> - Directory containing specification files

#### Optional Arguments:

- -o, --output <FILE> - Output IR file (default: stdout)
- --lockfile <FILE> - Toolchain lockfile path (default: local\_toolchain.lock.json )
- --validate - Validate IR against schema after generation
- -q, --quiet - Suppress info messages
- -v, --verbose - Enable debug logging

#### Output Format:

- JSON file conforming to schemas/stunir\_ir\_v1.schema.json
- Written to -o/--output file or stdout
- Logs written to stderr (never stdout)

#### Example Usage:

```

# SPARK implementation
./tools/spark/bin/stunir_spec_to_ir_main --spec-root spec/ardupilot_test --out ir.json

# Python implementation
python3 tools/spec_to_ir.py --spec-root spec/ardupilot_test --out ir.json

# Rust implementation (current - non-standard)
./tools/rust/target/release/stunir_spec_to_ir spec/ardupilot_test/test_spec.json -o ir.json

# Rust implementation (proposed - standardized)
./tools/rust/target/release/stunir_spec_to_ir --spec-root spec/ardupilot_test --out ir.json

```

## Implementation Comparison

Feature	SPARK	Python	Rust (Current)	Rust (Proposed)
--spec-root	✓ Yes	✓ Yes	✗ No (uses positional)	✓ Yes
--out	✓ Yes	✓ Yes	✓ Yes (-o)	✓ Yes
--lockfile	✓ Yes	✓ Yes	✗ No	✓ Yes
--validate	✗ No	✗ No	✗ No	⌚ Planned
Exit codes	✓ Standard	✓ Standard	⚠ Partial	✓ Standard

## 4.2 ir\_to\_code - IR to Code Emitter

### Purpose

Generate target language code from STUNIR IR JSON.

### Standard CLI (Recommended)

```
stunir_ir_to_code [OPTIONS] --ir <FILE> --target <TARGET>
```

#### Required Arguments:

- --ir <FILE> - Input IR JSON file (or - for stdin)
- --target <TARGET> - Target language/platform (e.g., c99, rust, python )

#### Optional Arguments:

- -o, --output <FILE> - Output code file (default: stdout)
- --template <DIR> - Custom template directory
- --validate - Validate IR schema before emission
- -q, --quiet - Suppress info messages
- -v, --verbose - Enable debug logging

**Output Format:**

- Generated code in target language
- Written to `-o/--output` file or stdout
- Logs written to stderr (never stdout)

**Example Usage:**

```
# SPARK implementation
./tools/spark/bin/stunir_ir_to_code_main --ir ir.json --target c99 --out output.c

# Python implementation
python3 tools/ir_to_code.py --ir ir.json --target c99 --out output.c

# Rust implementation (current)
./tools/rust/target/release/stunir_ir_to_code --ir ir.json --target c99 -o output.c

# Rust implementation (standardized)
./tools/rust/target/release/stunir_ir_to_code --ir ir.json --target c99 --out output.c
```

**Supported Targets****Polyglot:**

- `c89` - ANSI C (1989)
- `c99` - ISO C99
- `rust` - Rust (2021 edition)
- `python` - Python 3.9+
- `javascript` - ES6+
- `typescript` - TypeScript

**Assembly:**

- `x86` - x86 assembly (Intel syntax)
- `arm` - ARM assembly
- `riscv` - RISC-V assembly

**Lisp:**

- `common-lisp` - Common Lisp
- `scheme` - R7RS Scheme
- `clojure` - Clojure
- `racket` - Racket

**Specialized:**

- `cuda` - NVIDIA CUDA
- `wasm` - WebAssembly
- `embedded-arm` - ARM Cortex-M (bare metal)
- `matlab` - MATLAB/Octave
- `swift` - Swift 5+

## 5. Implementation Status

### 5.1 SPARK (Ada) - COMPLIANT

**Binary:** `tools/spark/bin/stunir_spec_to_ir_main`

```
./tools/spark/bin/stunir_spec_to_ir_main --spec-root <DIR> --out <FILE> --lockfile
<FILE>
```

**Status:** Fully standardized, DO-178C Level A compliant

**Strengths:**

- Formal verification
- Safety-critical certified
- Reference implementation for spec\_to\_ir

**Gaps:** None

---

## 5.2 Python - COMPLIANT

**Binary:** tools/spec\_to\_ir.py , tools/ir\_to\_code.py

```
python3 tools/spec_to_ir.py --spec-root <DIR> --out <FILE>
python3 tools/ir_to_code.py --ir <FILE> --target <TARGET> --out <FILE>
```

**Status:** Fully standardized, reference implementation

**Strengths:**

- Most comprehensive emitter coverage
- Active development
- Easiest to extend

**Gaps:** None

---

## 5.3 Rust - PARTIAL COMPLIANCE

**Binary:** tools/rust/target/release/stunir\_spec\_to\_ir , stunir\_ir\_to\_code

**Current CLI:**

```
./tools/rust/target/release/stunir_spec_to_ir <SPEC_FILE> -o <FILE>
./tools/rust/target/release/stunir_ir_to_code --ir <FILE> --target <TARGET> -o <FILE>
```

**Issues:**

1. spec\_to\_ir uses positional <SPEC\_FILE> instead of --spec-root <DIR>
2. Uses -o instead of --out (minor inconsistency)
3. Missing --lockfile support

**Recommended Fixes** (Priority: Medium):

```
// Update tools/rust/src/spec_to_ir.rs
#[derive(Parser, Debug)]
#[command(name = "stunir_spec_to_ir")]
struct Args {
    /// Specification root directory
    #[arg(long = "spec-root")]
    spec_root: PathBuf,

    /// Output IR file
    #[arg(short = 'o', long = "out")]
    output: Option<PathBuf>,

    /// Toolchain lockfile
    #[arg(long = "lockfile", default_value = "local_toolchain.lock.json")]
    lockfile: PathBuf,
}
```

**Status:** ! Partially compliant, requires minor CLI refactoring

## 5.4 Haskell - ! UNTESTED

**Binary:** tools/haskell/dist-newstyle/build/.../stunir-spec-to-ir, stunir-ir-to-code

**Expected CLI:**

```
stunir-spec-to-ir --spec-root <DIR> --out <FILE>
stunir-ir-to-code --ir <FILE> --target <TARGET> --out <FILE>
```

**Status:** ! Not validated (requires Haskell toolchain)

**Recommendation:** Test after cabal install to verify compliance

## 6. Examples

### 6.1 Basic Workflow

```
# Step 1: Generate IR from spec
./tools/spark/bin/stunir_spec_to_ir_main \
--spec-root spec/ardupilot_test \
--out ir.json \
--lockfile local_toolchain.lock.json

# Step 2: Emit C99 code
python3 tools/ir_to_code.py \
--ir ir.json \
--target c99 \
--out output.c

# Step 3: Compile (if applicable)
gcc -std=c99 -o output output.c
```

## 6.2 Pipeline with stdin/stdout

```
# Generate IR to stdout, pipe to emitter
./tools/spark/bin/stunir_spec_to_ir_main --spec-root spec/ardupilot_test | \
    python3 tools/ir_to_code.py --ir - --target python
```

## 6.3 CI/CD Integration

```
#!/bin/bash
set -euo pipefail

# Quiet mode for CI
./tools/spark/bin/stunir_spec_to_ir_main \
    --spec-root spec/ardupilot_test \
    --out ir.json \
    --quiet

# Validate IR
if ! jq -e '.schema == "stunir_ir_v1"' ir.json > /dev/null; then
    echo "ERROR: Invalid IR schema" >&2
    exit 4
fi

# Emit code
python3 tools/ir_to_code.py \
    --ir ir.json \
    --target c99 \
    --out output.c \
    --quiet

echo "✓ Pipeline complete"
```

## 6.4 Multi-Target Emission

```
# Generate IR once
./tools/spark/bin/stunir_spec_to_ir_main --spec-root spec/ardupilot_test --out ir.json

# Emit to multiple targets
for target in c99 rust python; do
    python3 tools/ir_to_code.py \
        --ir ir.json \
        --target $target \
        --out output.$target
done
```

## 7. Migration Guide

### 7.1 For Rust Tool Users

**Before (Current):**

```
./tools/rust/target/release/stunir_spec_to_ir spec/ardupilot_test/test_spec.json -o
ir.json
```

### After (Proposed v1.1):

```
./tools/rust/target/release/stunir_spec_to_ir --spec-root spec/ardupilot_test --out ir.json
```

### Migration Steps:

1. Update scripts to use `--spec-root` instead of positional argument
2. Change `-o` to `--out` for consistency (or add `--out` alias)
3. Add `--lockfile` support

**Timeline:** Recommended for v1.1 release (non-breaking if `-o` alias maintained)

## 7.2 For Tool Implementers

When implementing new STUNIR tools:

1. **Use `clap` (Rust) or `argparse` (Python)** for argument parsing
2. **Follow naming conventions:** `stunir_<tool>_<lang>`
3. **Implement standard options:** `-h`, `--version`, `-o`, `--quiet`, `--verbose`
4. **Write to stderr for logs:** Never pollute stdout with non-output messages
5. **Return standard exit codes:** 0 (success), 1 (error), 2 (args), 3 (I/O), 4 (schema)
6. **Validate IR schema:** Use JSON Schema validators before/after operations

## 8. Schema Validation

### 8.1 Validating Generated IR

All `spec_to_ir` tools must generate IR conforming to:

- **Schema:** `schemas/stunir_ir_v1.schema.json`
- **Required fields:** `schema`, `ir_version`, `module_name`, `types`, `functions`

**Python validation example:**

```
import json
import jsonschema

with open("schemas/stunir_ir_v1.schema.json") as f:
    schema = json.load(f)

with open("ir.json") as f:
    ir_data = json.load(f)

jsonschema.validate(instance=ir_data, schema=schema)
print("\u2713 IR is valid")
```

**Command-line validation:**

```
# Using ajv-cli (npm install -g ajv-cli)
ajv validate -s schemas/stunir_ir_v1.schema.json -d ir.json

# Using Python (one-liner)
python3 -c "import json, jsonschema; \
    jsonschema.validate(json.load(open('ir.json')), \
    json.load(open('schemas/stunir_ir_v1.schema.json')))"
```

## 9. Future Enhancements

### 9.1 Planned for v1.1

- [ ] Add `--validate` flag to all tools
- [ ] Standardize Rust CLI to match SPARK/Python
- [ ] Add JSON output for machine-readable logs (`--json-output`)
- [ ] Support multiple spec formats (YAML, TOML, etc.)

### 9.2 Planned for v1.2

- [ ] Add `--dry-run` mode for testing without file writes
- [ ] Support incremental IR generation (`--incremental`)
- [ ] Add progress bars for large specs (`--progress`)
- [ ] Configuration file support (`~/.stunirrc`, `.stunir.toml`)

### 9.3 Planned for v2.0

- [ ] Interactive mode (`--interactive`)
- [ ] Built-in IR diff tool (`stunir diff ir1.json ir2.json`)
- [ ] Plugin system for custom emitters
- [ ] Web-based IR visualizer

## 10. Summary

### Current State

- ✓ SPARK and Python implementations are fully standardized
- ⚠ Rust implementation requires minor CLI updates for full compliance
- ⚠ Haskell implementation untested (requires toolchain)

### Recommendations

- 1. Immediate (v1.0):** Document current state, add migration notes for Rust users
- 2. Short-term (v1.1):** Refactor Rust CLI to match SPARK/Python conventions
- 3. Long-term (v1.2+):** Add enhanced features (validation, dry-run, progress bars)

### Compliance Checklist

- ✓ Standard option naming (`--help`, `--version`, `--output`)
- ✓ Standard exit codes (0, 1, 2, 3, 4)
- ✓ Stderr for logs (never pollute stdout)

- JSON output conforming to stunir\_ir\_v1
  - Rust CLI refactoring needed
- 

**Document Version:** 1.0.0

**Last Reviewed:** January 31, 2026

**Next Review:** March 2026 (v1.1 release)