STUNIR Rust Testing Guide

This document describes the test structure, conventions, and best practices for testing the `stunir-native` Rust crate.

Test Structure

```
tests/
├── common/
│   └── mod.rs          # Shared test utilities
├── crypto_test.rs      # Cryptographic function tests
├── ir_test.rs          # IR parsing and validation tests
├── receipt_test.rs     # Receipt generation tests
├── canonical_test.rs   # JSON canonicalization tests
├── serialization_test.rs # Serialization tests
└── integration_test.rs # End-to-end integration tests
```

Running Tests

Run All Tests

```
cargo test
```

Run Specific Test File

```
cargo test --test crypto_test
cargo test --test ir_test
cargo test --test integration_test
```

Run Single Test

```
cargo test test_hash_file_basic
```

Run Tests with Output

```
cargo test -- --nocapture
```

Run Tests with Verbose Output

```
cargo test --verbose
```

Test Categories

1. Crypto Tests (`crypto_test.rs`)

Tests for cryptographic utilities including:
- File hashing (SHA-256)
- Directory hashing (Merkle trees)
- Hash determinism
- Error handling for invalid inputs

Key Tests:
- `test_hash_file_basic` - Basic file hashing
- `test_hash_path_directory` - Directory hashing
- `test_hash_directory_order_independent` - Determinism verification

2. IR Tests (`ir_test.rs`)

Tests for Intermediate Representation:
- Spec JSON parsing
- IR JSON parsing
- Structure validation
- Serialization roundtrips

Key Tests:
- `test_parse_spec_json` - Parse valid spec
- `test_ir_serialization_roundtrip` - Roundtrip consistency
- `test_invalid_spec_json` - Error handling

3. Receipt Tests (`receipt_test.rs`)

Tests for receipt handling:
- Receipt JSON parsing
- Serialization
- Tool info structure

4. Canonical Tests (`canonical_test.rs`)

Tests for JSON canonicalization:
- Whitespace normalization
- Key sorting
- Deterministic output
- Unicode handling

5. Serialization Tests (`serialization_test.rs`)

Tests for general serialization:
- JSON determinism
- Roundtrip consistency
- Special character handling
- Hash consistency

6. Integration Tests (`integration_test.rs`)

End-to-end workflow tests:
- Spec → IR pipeline

- File hash → Manifest generation
- Full workflow simulation

Test Utilities

The `common/mod.rs` module provides:

Functions

```rust
// Create temporary directory
let temp = create_temp_dir();

// Create test file with content
let path = create_test_file(temp.path(), "name.txt", "content");

// Create nested test directory structure
let files = create_test_tree(base);

// Assert file contains text
assert_file_contains(&path, "expected");

// Validate JSON
let value = assert_valid_json(json_str);

// Compute SHA-256
let hash = sha256_str("content");
```

Constants

```
SAMPLE_SPEC_JSON   // Valid spec JSON
SAMPLE_IR_JSON     // Valid IR JSON
SAMPLE_RECEIPT_JSON // Valid receipt JSON
```

Writing New Tests

1. Basic Test Structure

```rust
#[test]
fn test_feature_behavior() {
    // Arrange
    let input = "test data";

    // Act
    let result = function_under_test(input);

    // Assert
    assert!(result.is_ok());
    assert_eq!(result.unwrap(), expected);
}
```

2. Using Test Utilities

```rust
mod common;

use common::{create_temp_dir, create_test_file};

#[test]
fn test_with_temp_files() {
    let temp = create_temp_dir();
    let path = create_test_file(temp.path(), "test.txt", "content");

    // Test with temporary file
    let result = process_file(&path);
    assert!(result.is_ok());
}
```

3. Error Case Testing

```rust
#[test]
fn test_invalid_input_fails() {
    let result = parse_json("invalid");
    assert!(result.is_err(), "Should fail on invalid input");
}
```

4. Determinism Testing

```rust
#[test]
fn test_output_is_deterministic() {
    let input = "test";

    let result1 = process(input);
    let result2 = process(input);

    assert_eq!(result1, result2, "Output should be deterministic");
}
```

Best Practices

Do

- ✅ Use descriptive test names: `test_<module>_<scenario>_<expected>`
- ✅ Test both success and failure cases
- ✅ Verify determinism for any hashing/serialization
- ✅ Use temporary directories for file operations
- ✅ Clean assertions with clear messages
- ✅ Keep tests focused and independent

Don't

- ❌ Don't rely on external state
- ❌ Don't skip error case testing
- ❌ Don't write tests that depend on order
- ❌ Don't use hardcoded paths outside temp dirs
- ❌ Don't ignore test failures

Coverage Goals

Module	Target Coverage
crypto	80%+
ir_v1	80%+
canonical	90%+
spec_to_ir	70%+
receipt	70%+

CI Integration

Tests are automatically run by GitHub Actions on:
- Push to main/devsite
- Pull requests

See `.github/workflows/ci.yml` for workflow configuration.

Troubleshooting

Tests fail to find module

Ensure `src/lib.rs` exports the module:

```rust
pub mod module_name;
```

Temporary files not cleaned up

Use `tempfile::TempDir` which auto-cleans on drop.

Hash tests produce different values

Verify file content has no trailing whitespace/newlines.