STUNIR Documentation Standards

This guide defines the documentation standards for the STUNIR project across all supported languages.

Python Documentation

Module Docstrings

Every Python module should have a module-level docstring:

```
"""STUNIR IR Emitter Module.

This module provides functionality for converting STUNIR specifications
into deterministic Intermediate Representation (IR) format.

Example:
    >>> from tools.ir_emitter import emit_ir
    >>> ir = emit_ir.spec_to_ir(spec_data)
    >>> print(ir['module'])
    'my_module'

Attributes:
    CANONICAL_VERSION: The canonical JSON version used for output.
    IR_SCHEMA_VERSION: Current IR schema version.
"""
```

Function Docstrings

Use Google-style docstrings for all public functions:

```
def compute_sha256(data: Union[bytes, str]) -> str:
    """Compute SHA-256 hash of the given data.

    Computes a deterministic SHA-256 hash suitable for use in
    manifests and receipts.

    Args:
        data: The input data to hash. Can be bytes or a string.
            If a string is provided, it will be encoded as UTF-8.

    Returns:
        A lowercase hexadecimal string representation of the hash.

    Raises:
        TypeError: If data is neither bytes nor str.
        ValueError: If data is empty.

    Example:
        >>> compute_sha256("hello")
        '2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824'
    """
```

Class Docstrings

```python
class ManifestGenerator:
    """Generate deterministic manifests for STUNIR artifacts.

    This class provides methods for scanning directories, computing
    hashes, and generating canonical JSON manifests.

    Attributes:
        manifest_type: Type of manifest (e.g., 'ir', 'receipts').
        schema_version: Schema version string.
        output_path: Path where manifest will be written.

    Example:
        >>> gen = ManifestGenerator('ir', 'stunir.manifest.ir.v1')
        >>> gen.generate('asm/ir/', 'receipts/ir_manifest.json')
    """
```

Inline Comments

- Use sparingly for complex logic
- Explain “why” not “what”
- Keep comments up to date

```python
# Sort keys alphabetically for deterministic JSON output (RFC 8785)
data = dict(sorted(data.items()))

# NOTE: We use a placeholder for dCBOR until full implementation
# TODO(#1205): Replace with actual dCBOR encoder
```

Rust Documentation

Module Documentation

```rust
//! STUNIR Native Rust Toolchain
//!
//! This crate provides native Rust implementations of STUNIR
//! core functionality including hashing, serialization, and verification.
//!
//! # Features
//!
//! - Deterministic JSON canonicalization
//! - SHA-256 hashing for manifests
//! - Receipt generation and verification
//!
//! # Example
//!
//! ```rust
//! use stunir_native::canonical::canonical_json;
//!
//! let data = serde_json::json!({"b": 2, "a": 1});
//! let canonical = canonical_json(&data);
//! assert_eq!(canonical, r#"{"a":1,"b":2}"#);
//! ```
```

Function Documentation

```rust
/// Compute SHA-256 hash of the given data.
///
/// # Arguments
///
/// * `data` - Byte slice to hash
///
/// # Returns
///
/// A lowercase hexadecimal string of the SHA-256 hash.
///
/// # Examples
///
/// ```rust
/// let hash = compute_sha256(b"hello");
/// assert_eq!(hash, "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1-
fa7425e73043362938b9824");
/// ```
///
/// # Panics
///
/// This function does not panic.
pub fn compute_sha256(data: &[u8]) -> String {
    // Implementation
}
```

Struct Documentation

```rust
/// Configuration for manifest generation.
///
/// This struct holds all parameters needed to generate a STUNIR manifest,
/// including paths, schemas, and output options.
///
/// # Fields
///
/// * `manifest_type` - The type of manifest (ir, receipts, etc.)
/// * `schema` - Schema version string
/// * `source_dir` - Directory to scan for artifacts
#[derive(Debug, Clone)]
pub struct ManifestConfig {
    /// Type of manifest being generated
    pub manifest_type: String,
    /// Schema version identifier
    pub schema: String,
    /// Source directory path
    pub source_dir: PathBuf,
}
```

Haskell Documentation

Module Documentation (Haddock)

```
{-|
Module      : Stunir.Manifest
Description : Deterministic IR bundle manifest generation
Copyright   : (c) STUNIR Authors, 2024-2026
License     : MIT
Maintainer  : maintainer@example.com
Stability   : stable
Portability : POSIX

This module provides functionality for generating deterministic
IR bundle manifests by scanning directories and computing hashes.

Example usage:

@
import Stunir.Manifest

main :: IO ()
main = do
    manifest <- generateIrManifest "asm/ir"
    writeIrManifest "receipts/ir_manifest.json" manifest
@
-}
```

Function Documentation

```
-- | Compute SHA-256 hash of file contents.
--
-- Reads the file at the given path and computes its SHA-256 hash.
--
-- ==== Arguments
--
-- * @path@ - Path to the file to hash
--
-- ==== Returns
--
-- A lowercase hexadecimal string of the hash.
--
-- ==== Example
--
-- >>> computeFileHash "test.txt"
-- "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
computeFileHash :: FilePath -> IO String
```

Type Documentation

```haskell
-- | Represents a manifest entry for a single IR artifact.
--
-- Each entry contains metadata about one file in the IR bundle.
data ManifestEntry = ManifestEntry
    { entryName :: !Text      -- ^ Filename without path
    , entryHash :: !Text      -- ^ SHA-256 hash (lowercase hex)
    , entrySize :: !Int       -- ^ File size in bytes
    } deriving (Show, Eq, Generic)
```

Documentation Checklist

Required Documentation

- [ ] All public functions have docstrings
- [ ] All public classes/structs have docstrings
- [ ] All modules have module-level documentation
- [ ] Complex algorithms have inline comments
- [ ] All parameters are documented
- [ ] Return values are documented
- [ ] Exceptions/errors are documented
- [ ] Examples are provided for complex functions

Documentation Quality

- [ ] Docstrings follow the project style (Google for Python)
- [ ] Examples are runnable and correct
- [ ] Comments are up to date with code
- [ ] Technical terms are explained
- [ ] Links to related functions are provided

TODO/FIXME Tracking

Use standardized TODO and FIXME comments:

```
# TODO(#issue): Description of what needs to be done
# FIXME(#issue): Description of bug to fix
# HACK: Temporary workaround, explain why
# NOTE: Important information for maintainers
# DEPRECATED: Feature will be removed, use alternative
```

Documentation Linting

The project uses the following tools for documentation quality:

- **Python**: `pydocstyle` (via ruff D rules)
- **Rust**: `cargo doc --document-private-items`
- **Haskell**: `haddock` with warnings enabled

Run documentation checks:

```
# Python
ruff check --select=D tools/ manifests/

# Rust
cd tools/native/rust/stunir-native && cargo doc --no-deps

# Haskell
cd tools/native/haskell/stunir-native && cabal haddock
```