STUNIR Semantic IR Specification Format

Version: 1.0.0
Last Updated: 2026-01-30

Overview

This document defines the specification format for STUNIR Semantic IR. Specifications are written in JSON or YAML and describe the structure and behavior of programs across 24 target categories.

Base Schema

Root Object

```
{
  "schema": "https://stunir.dev/schemas/semantic_ir_v1_schema.json",
  "metadata": { ... },
  "functions": [ ... ],
  "types": [ ... ],
  "constants": [ ... ],
  "imports": [ ... ]
}
```

Metadata Object

Required Fields:
- `category`: Target category (see Supported Categories)

Optional Fields:
- `version`: Specification version (semver)
- `description`: Human-readable description
- `author`: Specification author
- `license`: License identifier
- `tags`: Array of tags

Category-Specific Fields: Each category may define additional metadata fields (see Category Extensions).

Function Object

```
{
  "name": "function_name",
  "parameters": [
    {
      "name": "param1",
      "type": "type_spec",
      "location": { ... }
    }
  ],
  "return_type": "type_spec",
  "body": [ ... ],
  "location": { ... },
  "is_inline": false,
  "is_static": false
}
```

Type Specification

Types can be specified as:

1. **Simple String**: `"i32"`, `"f64"`, `"bool"`
2. **Full Object**:

```
{
  "name": "i32",
  "is_primitive": true,
  "is_pointer": false,
  "is_array": false
}
```

1. **Pointer Type**:

```
{
  "name": "ptr",
  "is_pointer": true,
  "element_type": "i32"
}
```

1. **Array Type**:

```
{
  "name": "array",
  "is_array": true,
  "element_type": "i32",
  "array_size": 10
}
```

1. **Struct Type**:

```
{
  "name": "Point",
  "is_primitive": false,
  "fields": {
    "x": "i32",
    "y": "i32"
  }
}
```

Statement Object

```
{
  "kind": "statement_kind",
  "expressions": [ ... ],
  "statements": [ ... ],
  "location": { ... }
}
```

Statement Kinds:
- `assignment` : Variable assignment
- `return` : Return statement
- `if` : Conditional statement
- `while` : While loop
- `for` : For loop
- `call` : Function call
- `break` : Break statement
- `continue` : Continue statement

Expression Object

```
{
  "kind": "expression_kind",
  "value": "...",
  "type": "type_spec",
  "location": { ... }
}
```

Expression Kinds:
- `literal` : Literal value (number, string, bool)
- `variable` : Variable reference
- `binary_op` : Binary operation (+, -, *, /, etc.)
- `unary_op` : Unary operation (-, !, ~)
- `call` : Function call
- `array_access` : Array element access
- `field_access` : Struct field access

Source Location Object

```json
{
  "file": "spec.json",
  "line": 10,
  "column": 5,
  "length": 20
}
```

Primitive Types

Integer Types

- `i8` : 8-bit signed integer
- `i16` : 16-bit signed integer
- `i32` : 32-bit signed integer
- `i64` : 64-bit signed integer
- `u8` : 8-bit unsigned integer
- `u16` : 16-bit unsigned integer
- `u32` : 32-bit unsigned integer
- `u64` : 64-bit unsigned integer

Floating Point Types

- `f32` : 32-bit floating point
- `f64` : 64-bit floating point

Other Types

- `bool` : Boolean
- `char` : Character
- `string` : String
- `void` : Void (no value)

Category Extensions

Embedded Category

Additional Metadata:

```json
"metadata": {
  "category": "embedded",
  "target_arch": "arm|avr|risc-v|mips|...",
  "memory": {
    "ram_size": 65536,
    "flash_size": 262144,
    "stack_size": 4096
  }
}
```

Additional Root Fields:

```
"interrupts": [
  {
    "name": "Timer0_IRQ",
    "priority": 1,
    "handler": "timer0_handler"
  }
],
"peripherals": [
  {
    "name": "GPIO",
    "base_address": "0x40020000"
  }
]
```

GPU Category

Additional Metadata:

```
"metadata": {
  "category": "gpu",
  "gpu_platform": "cuda|rocm|opencl|metal|vulkan"
}
```

Additional Root Fields:

```
"kernels": [
  {
    "name": "vector_add",
    "grid_size": [256, 1, 1],
    "block_size": [256, 1, 1],
    "shared_memory": 0
  }
]
```

Lisp Category

Additional Metadata:

```
"metadata": {
  "category": "lisp",
  "dialect": "common-lisp|scheme|clojure|racket|emacs-lisp|guile|hy|janet"
}
```

Additional Root Fields:

```
"forms": [
  {
    "name": "add",
    "params": ["a", "b"],
    "body": "(+ a b)"
  }
],
"macros": [
  {
    "name": "when",
    "params": ["condition", "body"],
    "expansion": "..."
  }
],
"packages": [
  {
    "name": "my-package",
    "use": ["COMMON-LISP"]
  }
]
```

Prolog Category

Additional Metadata:

```
"metadata": {
  "category": "prolog",
  "dialect": "swi-prolog|gnu-prolog|sicstus|yap|xsb|ciao|b-prolog|eclipse"
}
```

Additional Root Fields:

```
"facts": [
  {
    "predicate": "parent",
    "args": ["john", "mary"]
  }
],
"rules": [
  {
    "head": "grandparent(X, Z)",
    "body": "parent(X, Y), parent(Y, Z)"
  }
],
"queries": [
  {
    "query": "grandparent(john, ann)",
    "expected": true
  }
]
```

WebAssembly Category

Additional Metadata:

```json
"metadata": {
  "category": "wasm",
  "wasm_version": "mvp|simd|threads"
}
```

Additional Root Fields:

```json
"exports": [
  {
    "name": "add",
    "type": "function"
  }
],
"imports": [
  {
    "module": "env",
    "name": "print",
    "type": "function"
  }
]
```

Complete Examples

See `examples/specifications/` for complete examples of all 24 categories.

Minimal Example

```json
{
  "schema": "https://stunir.dev/schemas/semantic_ir_v1_schema.json",
  "metadata": {
    "category": "polyglot",
    "version": "1.0.0"
  },
  "functions": [
    {
      "name": "add",
      "parameters": [
        {"name": "a", "type": "i32"},
        {"name": "b", "type": "i32"}
      ],
      "return_type": "i32",
      "body": [
        {
          "kind": "return",
          "expressions": [
            {
              "kind": "binary_op",
              "value": "+",
              "operands": [
                {"kind": "variable", "value": "a"},
                {"kind": "variable", "value": "b"}
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Validation Rules

1. **Schema Field**: Must be present and match semantic IR schema URL
2. **Category**: Must be one of 24 supported categories
3. **Function Names**: Must be unique within specification
4. **Type References**: All referenced types must be defined
5. **Parameter Names**: Must be unique within function
6. **Expression Types**: Must be compatible with operations

Best Practices

1. **Always include source locations** for better error messages
2. **Use type inference** when possible to reduce verbosity
3. **Group related functions** using comments or metadata
4. **Validate specifications** before committing
5. **Document complex expressions** using comments in metadata

See Also

- Parser Guide (SEMANTIC_IR_PARSER_GUIDE.md)
- API Documentation (SEMANTIC_IR_PARSER_API.md)
- Examples (../examples/specifications/)