# SODL Language Documentation v0.3

## Overview

SODL is a Python-like, indentation-based Domain-Specific Language (DSL) for controlled AI-driven code generation. It allows developers to describe intent, architecture, constraints, and generation plans in a concise, structured, and reproducible way.

### *Key Characteristics*

- **Python-like syntax**: Familiar to software engineers
- **Indentation-based blocks**: Uses `:` and indentation
- **Declarative**: Describes what, not how
- **Explicit constraints**: Over implicit assumptions
- **Human-writable, machine-compilable**: Easy to write and deterministic to parse
- **AI-agent targeted**: Generates instructions for AI coding agents, not application code

### *Core Pipeline*

```
Intent → Plan → Prompt Chunks → Coding Agent
```

- **Intent**: Defines what must be built and why
- **Plan**: Defines how generation is staged and constrained
- **Prompt Chunks**: Deterministic instructions consumed by an AI coding agent

## Language Constructs

### *Top-Level Constructs*

| Construct | Purpose |
|---|---|
| `template` | Reusable base specification |
| `system` | Concrete system to be generated |
| `interface` | Required functionality contract |

| `policy` | Global rules and constraints |

| `module` | Unit of responsibility and generation |

| `pipeline` | Controlled generation process |

| `step` | Atomic generation phase |

## System Definition

A `system` is the root executable specification that defines a complete application or component.

### *Basic System Syntax*

```
system "MySystem":
  version = "1.0.0"
  stack:
    language = "Python 3.12"
    web = "Flask"
  intent:
    primary = "Main application goal"
```

### *System Components*

#### stack

Defines technological context and dependencies.

```
stack:
  language = "Python 3.12"
  web = "FastAPI"
  templating = "Jinja2"
  ui_framework = "Material Components Web"
  database = "PostgreSQL"
  testing = ["pytest", "pytest-cov"]
```

#### intent

Defines goals, outcomes, and scope boundaries.

```
intent:
  primary = "Build a todo list application"
  outcomes = [
```

```
    "Users can create, read, update, delete todos",
    "Todos have categories and due dates",
    "RESTful API with JSON responses"
  ]
  out_of_scope = ["User authentication", "Real-time sync", "Mobile app"]
```

### *Complete System Example*

```
system "BlogPlatform":
  version = "2.0.0"
  stack:
    language = "Python 3.12"
    web = "FastAPI"
    database = "PostgreSQL"
    orm = "SQLAlchemy"
  intent:
    primary = "Modern blogging platform with rich text editor"
    outcomes = [
      "Users can create and publish blog posts",
      "Rich text editing with markdown support",
      "Comment system with moderation",
      "Tag-based categorization"
    ]
    out_of_scope = ["User authentication", "Analytics dashboard"]
```

## Templates and Inheritance

### *Template Definition*

Templates are reusable base specifications that can be extended by systems.

```
template "BasePythonWebApp":
  stack:
    language = "Python 3.12"
    testing = ["pytest"]
  policy Security:
    rule "No secrets in repository" severity=critical
    rule "Validate all inputs" severity=high
```

### *Inheritance with extends*

Systems can extend templates to inherit their configuration.

```
system "MyApp" extends "BasePythonWebApp":
  intent:
    primary = "My specific application"
  stack:
    web = "Flask"
```

**Inheritance rules:**

- Single inheritance only (MVP)
- Merge order: parent → child
- Child values override parent values

### *Override Operations*

| Operation | Syntax | Meaning |
|---|---|---|
| Replace | `override path = value` | Replace scalar or block |
| Append | `append path += value` | Append to list |
| Remove | `remove path -= value` | Remove from list |
| Replace block | `replace block Name:` | Replace entire named block |

```
system "CustomApp" extends "BasePythonWebApp":
  override stack.language = "Python 3.13"
  append stack.testing += "pytest-asyncio"
  remove stack.testing -= "pytest-cov"
```

## Interfaces

Interfaces describe required functionality contracts without implementation details.

### *Interface Declaration*

```
interface ImageStore:
  doc = "Stores PNG images and returns public URLs"
  method save_png(pngBytes: bytes) -> SavedImage
  method get_url(filename: str) -> str
  invariants:
    invariant "Filename is generated server-side"
```

```
    invariant "URLs are publicly accessible"
```

### *Interface Methods*

Method syntax: `method name(param: type) -> return_type`

```
interface TodoStore:
  doc = "Persistent storage for todo items"
  method create(todo: TodoInput) -> TodoItem
  method get_all() -> List[TodoItem]
  method get_by_id(id: UUID) -> Optional[TodoItem]
  method update(id: UUID, updates: TodoUpdate) -> TodoItem
  method delete(id: UUID) -> bool
```

### *Interface Inheritance*

Interfaces can extend other interfaces.

```
interface Storage:
  method save(data: bytes) -> str
  method retrieve(key: str) -> bytes

interface ImageStorage extends Storage:
  override method save(data: bytes) -> SavedImage
  method get_thumbnail(key: str, size: tuple) -> bytes
```

### *Interface Usage in Modules*

```
module ImageAPI:
  requires = [ImageStore]  # Depends on interface

module StorageLocal:
  implements = [ImageStore]  # Must implement all methods
  exports = [ImageStore]     # Provides interface to others
```

**Semantics:**

- `requires`: Module depends only on interface contract
- `implements`: Module must implement all interface methods
- `exports`: Module provides interface to other modules

## Policies

Policies define global rules and constraints with severity levels.

### *Policy Syntax*

```
policy Security:
  rule "No secrets in repository" severity=critical
  rule "Validate all user input" severity=high
  rule "Use HTTPS for external APIs" severity=high
  rule "Log security events" severity=medium
```

### *Severity Levels*

| Severity | Meaning | Enforcement |

|----------|---------|-------------|

| `critical` | MUST NOT violate | Hard constraint, blocks generation |

| `high` | MUST follow | Required constraint |

| `medium` | SHOULD follow | Strong recommendation |

| `low` | MAY follow | Suggestion |

### *Multiple Policies*

```
system "SecureApp":
  policy Security:
    rule "Encrypt sensitive data at rest" severity=critical
    rule "Use parameterized queries" severity=critical
    rule "Implement rate limiting" severity=high

  policy Performance:
    rule "Cache frequently accessed data" severity=medium
    rule "Use database indexes" severity=high
    rule "Lazy load large resources" severity=medium

  policy CodeQuality:
    rule "Test coverage above 80%" severity=high
    rule "No code duplication" severity=medium
    rule "Document public APIs" severity=medium
```

## Modules

Modules are the primary unit of generation and responsibility.

### *Module Structure*

```
module ImageAPI:
  owns = ["Image upload endpoint", "Image retrieval logic"]
  requires = [ImageStore]
  api:
    endpoint "POST /api/images" -> JsonSavedImage
    endpoint "GET /api/images/{id}" -> JsonImage
  invariants:
    invariant "Accept only PNG and JPEG formats"
    invariant "Validate file size < 10MB"
  acceptance:
    test "valid image uploads successfully"
    test "oversized image returns 413"
    test "invalid format returns 400"
  artifacts = ["app/api.py", "app/routes/images.py"]
```

### *Module Sections*

| Section | Purpose | Example |
|---|---|---|
| `owns` | Domain ownership and responsibilities | `owns = ["User authentication"]` |
| `requires` | Required interfaces (dependencies) | `requires = [TodoStore, Logger]` |
| `implements` | Implemented interfaces | `implements = [TodoStore]` |
| `exports` | Provided interfaces to others | `exports = [TodoStore]` |
| `api` | External API definition | `endpoint "GET /api/todos"` |
| `invariants` | Must-hold constraints | `invariant "IDs are unique"` |
| `acceptance` | Definition of Done (tests) | `test "creates todo successfully"` |
| `artifacts` | Allowed file scope | `artifacts = ["app.py"]` |
| `config` | Module configuration | `config: timeout = 30` |

### *API Endpoints*

```
module TodoAPI:
  api:
    endpoint "GET /api/todos" -> List[TodoResponse]
    endpoint "POST /api/todos" -> TodoResponse (201 CREATED)
    endpoint "PUT /api/todos/{id}" -> TodoResponse
    endpoint "DELETE /api/todos/{id}" -> Empty (204 NO CONTENT)
    endpoint "GET /api/todos/{id}" -> TodoResponse
```

### *API Models*

```
module StorageModels:
  owns = ["Data models for todo items"]
  api:
    model TodoItem:
      field id: UUID
      field title: str
      field description: Optional[str]
      field category: Optional[str]
      field due_date: Optional[datetime]
      field priority: int (1-3)
      field completed: bool

    model TodoInput:
      field title: str
      field description: Optional[str]
      field category: Optional[str]
      field due_date: Optional[str]
      field priority: int (1-3)
```

### *Invariants*

Invariants define constraints that must always hold true.

```
module UserService:
  invariants:
    invariant "Email addresses are unique"
    invariant "Passwords are hashed with bcrypt"
    invariant "User IDs are UUIDs"
    invariant "All timestamps use UTC"
```

### *Acceptance Tests*

Define the criteria for successful module implementation.

```
module WebUI:
  acceptance:
    test "renders empty state when no todos exist"
    test "submits valid todo and displays it"
    test "shows error on invalid due date"
    test "toggles completion status correctly"
    test "deletes todo and updates UI"
```

### *Complete Module Example*

```
module InMemoryTodoStore:
  implements = [TodoStore]
  exports = [TodoStore]
  config:
    persistence = "in-memory (ephemeral)"
    max_items = 1000
  invariants:
    invariant "Thread-safe access to todo list"
    invariant "All operations maintain data consistency"
    invariant "UUIDs are unique across all todos"
  acceptance:
    test "persists todo across GET calls within same runtime"
    test "correctly creates new todos with unique IDs"
    test "properly updates existing todos"
    test "successfully deletes todos"
    test "handles concurrent access safely"
  artifacts = ["app/storage.py"]
```

## Pipelines and Steps

Pipelines define the controlled generation process.

### *Pipeline Syntax*

```
pipeline "Development":
  step Design:
    output = design
    require = "Produce architecture diagram and data model"

  step Implement:
    modules = ["StorageModels", "TodoStore", "TodoAPI"]
    output = code
    gate = "All acceptance tests pass"
```

```
  step Review:
    output = diff
    require = "Code review checklist completed"
```

### *Step Definition*

Steps are atomic generation phases with specific outputs and constraints.

```
step Implement "Backend":
  modules = ["ImageAPI", "StorageLocal"]
  output = code
  require = "Generate code incrementally"
  gate = "pytest passes with 80% coverage"
```

### *Step Components*

| Component | Purpose | Example |

|-----------|---------|---------|

| `modules` | Which modules to generate | `modules = ["TodoAPI"]` |

| `output` | Type of output to produce | `output = code` |

| `require` | Additional requirements | `require = "Follow style guide"` |

| `gate` | Exit criteria | `gate = "All tests pass"` |

### *Allowed Output Types*

- `design`: Architecture diagrams, data models
- `code`: Application code
- `tests`: Test code
- `diff`: Code changes/review
- `docs`: Documentation

### *Complete Pipeline Example*

```
pipeline "Cursor":
  step Design:
    output = design
    require = "Create data model and API specification"
  step ImplementModels:
```

```
    modules = ["StorageModels", "APISchemas"]
    output = code
    require = "Use Pydantic for validation"
    gate = "Models compile without errors"

  step ImplementStorage:
    modules = ["InMemoryTodoStore"]
    output = code
    require = "Thread-safe implementation"
    gate = "Storage tests pass"

  step ImplementAPI:
    modules = ["TodoAPI"]
    output = code
    require = "RESTful conventions"
    gate = "API tests pass"

  step ImplementUI:
    modules = ["WebUI"]
    output = code
    require = "Material Design components"
    gate = "UI tests pass"

  step FinalReview:
    output = diff
    require = "All acceptance criteria met"
    gate = "Full integration test suite passes"
```

## Complete Examples

### *Example 1: Image Drawing Web App*

```
system "MouseDrawWebApp":
  stack:
    language = "Python 3.12"
    web = "Flask"

  intent:
    primary = "Draw with mouse and save image"
    outcomes = [
      "Browser-based canvas for mouse drawing",
      "Save drawings as PNG files",
      "Retrieve saved images by ID"
    ]

  interface ImageStore:
    method save_png(pngBytes: bytes) -> SavedImage
```

```
    method get_png(id: str) -> bytes

  module ImageAPI:
    requires = [ImageStore]
    api:
      endpoint "POST /api/save" -> JsonSavedImage
      endpoint "GET /api/image/{id}" -> PNG
    invariants:
      invariant "Accept only valid PNG data URLs"
      invariant "Generate unique IDs server-side"
    acceptance:
      test "saves valid PNG successfully"
      test "returns saved image by ID"

  module StorageLocal:
    implements = [ImageStore]
    exports = [ImageStore]
    config:
      storage_path = "./images"
    artifacts = ["app/storage.py"]

  pipeline "Cursor":
    step Implement:
      modules = ["ImageAPI", "StorageLocal"]
      output = code
      gate = "All tests pass"
```

### *Example 2: Advanced Todo Application*

```
system "AdvancedTodoApp":
  version = "1.0.0"
  stack:
    language = "Python 3.12"
    web = "FastAPI"
    templating = "Jinja2"
    ui_framework = "Material Components Web"

  intent:
    primary = "Full-featured web-based Todo list with rich UI"
    outcomes = [
      "Create, read, update, and delete todos",
      "Support categories, due dates, priority levels",
      "Responsive HTML frontend with Material Design",
      "RESTful JSON API alongside server-rendered pages"
    ]
    out_of_scope = ["User authentication", "Real-time sync", "Mobile app"]

  interface TodoStore:
    doc = "Persistent storage for advanced todo items"
```

```
  method create(todo: TodoInput) -> TodoItem
  method get_all() -> List[TodoItem]
  method get_by_id(id: UUID) -> Optional[TodoItem]
  method update(id: UUID, updates: TodoUpdate) -> TodoItem
  method delete(id: UUID) -> bool
  invariants:
    invariant "All todos have unique UUIDs"
    invariant "Due date may be null"

module WebUI:
  owns = ["Todo listing page", "Todo creation form", "Task interface"]
  requires = [TodoStore]
  api:
    endpoint "GET /" -> HTML (via Jinja2)
    endpoint "POST /todos" -> Redirect to "/"
    endpoint "POST /todos/{id}/toggle" -> Redirect to "/"
    endpoint "POST /todos/{id}/delete" -> Redirect to "/"
  invariants:
    invariant "Forms validate client and server-side"
    invariant "Material Design components properly initialized"
  acceptance:
    test "renders empty state when no todos exist"
    test "submits valid todo and displays it"
    test "shows error on invalid due date"
    test "toggles completion status correctly"
    test "deletes todo and updates UI"
  artifacts = ["app/main.py", "app/templates/*.html", "app/static/css/*.css"]

module TodoAPI:
  owns = ["REST API for todos"]
  requires = [TodoStore]
  api:
    endpoint "GET /api/todos" -> List[TodoResponse]
    endpoint "POST /api/todos" -> TodoResponse (201 CREATED)
    endpoint "PUT /api/todos/{id}" -> TodoResponse
    endpoint "DELETE /api/todos/{id}" -> Empty (204 NO CONTENT)
  invariants:
    invariant "All API responses use consistent JSON schema"
    invariant "Invalid inputs return 422 with error details"
    invariant "RESTful conventions followed"
  acceptance:
    test "creates todo via POST with valid payload"
    test "returns 404 for non-existent ID"
    test "updates todo via PUT"
    test "deletes todo via DELETE"
  artifacts = ["app/api.py"]

module InMemoryTodoStore:
  implements = [TodoStore]
  exports = [TodoStore]
  config:
    persistence = "in-memory (ephemeral)"
  invariants:
```

```
      invariant "Thread-safe access to todo list"
      invariant "All operations maintain data consistency"
    acceptance:
      test "persists todo across GET calls"
      test "creates new todos with unique IDs"
      test "properly updates existing todos"
      test "successfully deletes todos"
    artifacts = ["app/storage.py"]

  module StorageModels:
    owns = ["Data models for todo items"]
    api:
      model TodoItem:
        field id: UUID
        field title: str
        field description: Optional[str]
        field category: Optional[str]
        field due_date: Optional[datetime]
        field priority: int (1-3)
        field completed: bool
      model TodoInput:
        field title: str
        field description: Optional[str]
        field due_date: Optional[str]
        field priority: int (1-3)
    artifacts = ["app/storage.py"]

  pipeline "Development":
    step Design:
      output = design
      require = "Architecture diagram and data model"

    step Implement:
      modules = ["StorageModels", "InMemoryTodoStore", "TodoAPI", "WebUI"]
      output = code
      gate = "All acceptance tests pass"

    step Review:
      output = diff
      require = "Material Design components properly initialized"
```

### *Example 3: E-Commerce Product Catalog*

```
system "ProductCatalog":
  version = "2.0.0"
  stack:
    language = "Python 3.12"
    web = "FastAPI"
    database = "PostgreSQL"
```

```
  orm = "SQLAlchemy"
  cache = "Redis"

intent:
  primary = "Scalable product catalog with search and filtering"
  outcomes = [
    "Browse products by category and price range",
    "Full-text search across product names and descriptions",
    "Filter by multiple attributes simultaneously",
    "Paginated results for performance",
    "Product images with CDN URLs"
  ]
  out_of_scope = ["Shopping cart", "Checkout", "Payment processing"]

policy Performance:
  rule "Cache product listings for 5 minutes" severity=high
  rule "Index all searchable fields" severity=critical
  rule "Paginate results with max 50 items" severity=high

policy DataQuality:
  rule "All products have valid SKUs" severity=critical
  rule "Prices are positive decimals" severity=critical
  rule "Images are validated URLs" severity=high

interface ProductRepository:
  method find_all(filters: ProductFilters, page: int) -> PagedProducts
  method find_by_id(id: UUID) -> Optional[Product]
  method search(query: str, page: int) -> PagedProducts
  method create(product: ProductInput) -> Product
  method update(id: UUID, updates: ProductUpdate) -> Product
  invariants:
    invariant "SKUs are unique across all products"
    invariant "Category hierarchy is validated"

interface CacheService:
  method get(key: str) -> Optional[bytes]
  method set(key: str, value: bytes, ttl: int) -> bool
  method invalidate(pattern: str) -> int

module ProductAPI:
  requires = [ProductRepository, CacheService]
  api:
    endpoint "GET /api/products" -> PagedProductResponse
    endpoint "GET /api/products/{id}" -> ProductResponse
    endpoint "POST /api/products" -> ProductResponse (201)
    endpoint "PUT /api/products/{id}" -> ProductResponse
    endpoint "GET /api/products/search" -> PagedProductResponse
  invariants:
    invariant "Cache results for 5 minutes"
    invariant "Return 304 Not Modified when appropriate"
  artifacts = ["app/api/products.py"]

module PostgresProductRepository:
```

```
  implements = [ProductRepository]
  exports = [ProductRepository]
  config:
    connection_pool_size = 20
    search_index = "gin"
  invariants:
    invariant "Use prepared statements for all queries"
    invariant "Implement full-text search with tsvector"
  artifacts = ["app/repositories/products.py"]

module RedisCache:
  implements = [CacheService]
  exports = [CacheService]
  config:
    max_connections = 50
    default_ttl = 300
  artifacts = ["app/cache.py"]

pipeline "Production":
  step Design:
    output = design
    require = "Database schema with indexes"

  step ImplementModels:
    modules = ["ProductModels"]
    output = code
    gate = "Models validated"

  step ImplementRepository:
    modules = ["PostgresProductRepository"]
    output = code
    gate = "Repository tests pass"

  step ImplementCache:
    modules = ["RedisCache"]
    output = code
    gate = "Cache tests pass"

  step ImplementAPI:
    modules = ["ProductAPI"]
    output = code
    gate = "API integration tests pass"
```

## Design Principles

1. **Explicit over implicit**: State everything clearly, avoid assumptions
2. **Constraints over suggestions**: Use enforceable rules, not soft guidelines
3. **Architecture before code**: Define structure before implementation

4. **Deterministic prompts**: Same spec always produces same instructions
5. **Developer-controlled AI**: Human maintains control over generation process

## Best Practices

### *1. Module Organization*

- Keep modules focused on single responsibility
- Use interfaces to define contracts between modules
- Clearly specify dependencies with `requires`
- Document ownership boundaries with `owns`

### *2. Interface Design*

- Design interfaces before implementing modules
- Keep interfaces stable; change implementations instead
- Use semantic type annotations
- Document invariants for interface contracts

### *3. Pipeline Structure*

- Order steps by logical dependency
- Use gates to enforce quality at each step
- Make steps atomic and independently verifiable
- Specify clear output types for each step

### *4. Policy Definition*

- Use appropriate severity levels
- Make policies specific and measurable
- Group related rules into named policies
- Prioritize critical security and data integrity rules

### *5. Testing Strategy*

- Define acceptance criteria for every module

- Include positive and negative test cases
- Test interface boundaries
- Verify invariants through tests

## Compiler Output

A SODL compiler produces structured output for AI agents:

```
.sodl/
■■■ global.md              # System-level context
■■■ modules/
■   ■■■ ImageAPI.md       # Module-specific instructions
■   ■■■ StorageLocal.md
■■■ steps/
■   ■■■ Implement__ImageAPI.md  # Step-by-step instructions
■■■ manifest.json          # Metadata and structure
```

These files are consumed by AI coding agents (Cursor, Claude, GPT) to generate application code.

## Non-Goals

SODL is **not**:

- A programming language for writing application logic
- A UML replacement or visual modeling tool
- A test framework or test runner
- A code generator (it generates *instructions* for AI agents)
- A runtime system or execution environment

## Summary

SODL provides a structured, extensible way to control AI coding agents using familiar software engineering concepts:

- **Inheritance**: Reuse specifications through templates
- **Interfaces**: Define contracts between components
- **Modules**: Organize code by responsibility
- **Pipelines**: Control generation process flow
- **Constraints**: Enforce quality and correctness

It transforms prompt engineering into specification engineering, enabling reproducible, reviewable, and maintainable AI-driven development.

# Language Reference

## *Keywords*

- `system`: Define a concrete system
- `template`: Define a reusable template
- `extends`: Inherit from template
- `interface`: Define functionality contract
- `implements`: Implement an interface
- `exports`: Provide interface to others
- `requires`: Depend on interface
- `module`: Define generation unit
- `policy`: Define rules and constraints
- `pipeline`: Define generation process
- `step`: Define generation phase
- `override`: Replace value
- `append`: Add to list
- `remove`: Remove from list
- `replace`: Replace block

## *Sections*

- `stack`: Technology stack definition
- `intent`: Goals and scope
- `api`: API definition (endpoints, models)
- `owns`: Ownership declaration
- `invariants`: Constraints that must hold
- `acceptance`: Definition of done
- `artifacts`: File scope
- `config`: Configuration values

## *Severity Levels*

- `critical`: Must not violate
- `high`: Must follow
- `medium`: Should follow

- `low`: May follow

### *Output Types*

- `design`: Architecture and planning
- `code`: Application code
- `tests`: Test code
- `diff`: Code changes
- `docs`: Documentation

## Version History

### *v0.3 (Current)*

- Stable syntax for production use
- Complete interface system
- Pipeline and step definitions
- Template inheritance
- Policy severity levels

---

**SODL**: Turning prompt engineering into specification engineering.