

SPELL

Spec Programming & Execution Logic Language

Architect Manual

Version 2.0.0

Marcos Fiore

1. What is SPELL

SPELL is a technical specification language with a compiler. You write the system spec in structured syntax; spellcomp validates, audits, and generates a set of .md files that the executor agent follows as a contract — no inferences, no ambiguities, no architectural decisions delegated to AI.

The complete SPELL workflow is:

Architect writes	→	spellcomp compiles	→	Agent implements
schedule.spell		Structural validation		-data.md
		Semantic audit		-rules.md
		Gap detection		-endpoints.md
		.md generation per module		-screens.md

What sets SPELL apart from any manually written specification document is spellcomp itself: a compiler that detects contradictions, dead references, and logical gaps that human review cannot reliably catch. The spec is not just written — it is proven.

2. SPELL, BMAD, and Spec-Driven Development

The problem all these tools try to solve

Asking AI for code without a prior specification is the leading cause of rework in AI-assisted development. The AI infers what was left unsaid, makes architectural decisions it has no authority to make, and produces code that needs to be corrected — sometimes from scratch.

SPELL eliminates this problem at the root.

BMAD: when the process also needs structure

The BMAD Method organizes the AI development process. It defines roles (Analyst, Architect, Developer), work sequences, and specialized agents that collaborate in a chain to carry an idea all the way to code.

BMAD does not specify systems — it orchestrates who does what and in what order.

Teams that use BMAD to organize the process and SPELL to specify technically get the best of both: structured process with precise specification. But BMAD does not replace SPELL — without a formal technical specification, BMAD agents work on natural-language documents, subject to the same ambiguities that Spec-Driven Development was created to eliminate.

SPELL can be adopted independently of BMAD. The reverse does not offer the same guarantees.

OpenSpec and Spec Kit: same philosophy, less depth

OpenSpec and Spec Kit implement Spec-Driven Development (SDD) — the same philosophy as SPELL: specification before code. They structure the developer's intent, break the spec into smaller tasks, and guide the agent to execute in a controlled manner.

That is the right approach. But there are fundamental differences compared to SPELL:

	OpenSpec / Spec Kit	SPELL
Format	Structured documents and templates	Formal language with its own syntax
Validation	Human review	Compiler with structural validation
Semantic audit	Not available	Detects gaps, contradictions, and dead references via AI
Output	Tasks and user stories	Domain .md files, optimized for agents
Target audience	Developers	System architects
Domain	Generic	Specific to software architecture

The central difference lies in semantic auditing. OpenSpec and Spec Kit turn a spec into tasks — but they do not question whether the spec is logically consistent. spellcomp does. An entity with a status field and no declared transition rule, an endpoint with no error response for a resource that might not exist, a screen that navigates to an undeclared screen — all of this is detected before any agent starts working.

Teams already using OpenSpec or Spec Kit can adopt SPELL as a prior layer: the spec compiled and audited by spellcomp feeds those tools' task flows as an immutable source of truth. Teams that adopt SPELL from scratch need nothing more to have a reliable, traceable AI-assisted development process.

3. Syntax: numbered hierarchy

SPELL uses numbered hierarchy to infer structure. You can write in Word and save as plain text (.spell).

```
# Hierarchy
1.          → root block (fixed name — see table below)
1.1.        → sub-block inside the root block
1.1.1.      → sub-item inside the sub-block

# Attributes
field: value          → structured attribute
field: recommend      → delegate decision to AI

# Free-form instructions
- text                → implicit instructions

# Comments
// text               → ignored by the parser
```

```
# Inline modifiers
field: type(N), modifier1, modifier2
```

Parser rules:

- The parser recognizes root blocks by name, not by number.
- Numbers do not need to be sequential — 2. can come before 1. without error.
- Lines starting with - inside any block are treated as implicit instructions.
- The word adapt alone on a line inside Wireframes tells the AI it may adapt the layout for mobile or accessibility.

4. Root blocks

Number	Fixed name	Required
1.	Project	Yes
2.	Database	Recommended
3.	Business Rules	Recommended
4.	Endpoints	Recommended
5.	Screens	Recommended
6.	Security	Optional
7.	Services	Optional

5. Project block

```
1. Project
  name: "EasyScheduler"
  version: "1.0"
  description: "System for clinics to schedule appointments"
```

1.1. Stack

```
1.1. Stack
  frontend: React
  backend: Node.js
  database: PostgreSQL
  authentication: accounts with login and password
  agent: claude-code
```

The agent field controls the name of the generated entry file:

Value	Generated file
claude-code (default)	CLAUDE.md
cursor	.cursor/rules
aider	CONVENTIONS.md
copilot	.github/copilot-instructions.md
none	(no entry file generated)

1.2. Conventions

Declares organizational constraints — not technical conventions the AI already knows.

```
1.2. Conventions
  code-language: english
  - All services must have the Service suffix
  - Search methods must start with find
```

Field	Description
code-language	Language of identifiers in generated code
- text	Organizational convention in natural language

1.3. Skills

Declares capabilities expected from the executor agent. spellcomp also detects additional skills automatically.

```
1.3. Skills
  - pdf-reading
  - xlsx
  - charts
```

Available skills in the catalog: pdf-reading, xlsx, charts, docx, frontend-design, oauth-integration, email, storage, websocket, queue.

6. Database block

2.0. Configuration

Special sub-block with fixed number 2.0. for global database configuration.

```
2. Database
```

```

2.0. Configuration
  multitenancy: schema-per-tenant
  scope: hybrid
  migration: goose
  - Global tables live in the public schema
  - Tenant tables use the prefix tenant_{id}

```

Field	Possible values
multitenancy	schema-per-tenant, database-per-tenant, tenant-column, none
scope	global, per-tenant, hybrid
migration	flyway, liquibase, goose, etc.

2.N. Table TableName

```

2.1. Table Doctor
  id: integer, primary-key, auto-increment
  name: text(100), required
  license-number: text(20), unique, required
  active: boolean, default: yes

```

Available types:

integer, decimal, decimal(P,S), text, text(N), boolean, date, datetime, time, file, json

Modifiers:

primary-key, auto-increment, unique, required, optional, default: value, private, indexed

Delegating structure to the AI:

```

2.2. Table Patient
  structure: recommend
  - Patient must have name, date of birth, and phone number
  - SSN is optional but must be unique if provided

```

2.N.1. Relationships

Used when naming conventions are not enough (two fields pointing to the same table).

```

2.4. Table Message
  sender_id: integer, required
  recipient_id: integer, required

```

```

2.4.1. Relationships
  sender_id -> User.id
  recipient_id -> User.id

```

7. Business Rules block

Business rules are the heart of the specification. The AI does not invent rules — anything not declared here will be ignored.

3.1. Rule SlotAvailable

```
context: Appointment scheduling
condition: Doctor already has an appointment at the same time
action: Reject the booking with a slot-unavailable message
reason: Clinic policy to prevent scheduling conflicts
```

Field	Required	Description
context	Yes	Which operation or moment the rule applies to
condition	Yes	Exactly when the rule triggers
action	Yes	What must happen
exception	No	Case in which the rule does not apply
reason	No	Why this rule exists

8. Endpoints block

4.1. Endpoint ScheduleAppointment

```
route: POST /appointments
authentication: required
rules: SlotAvailable
```

4.1.1. Input Data

```
doctor_id: integer, required
patient_id: integer, required
datetime: datetime, required
```

4.1.2. Responses

```
201: Appointment scheduled successfully
400: Time slot unavailable for this doctor
404: Doctor or patient not found
```

Field	Description
route	HTTP method + path (e.g. POST /appointments)
authentication	required or not-required
rules	Name(s) of rules declared in block 3. Business Rules

9. Screens block

```
5.1. Screen DoctorSchedule
  route: /schedule/{doctor_id}
  authentication: required
  layout: recommend
  - Show weekly calendar with occupied and free slots
  - Clicking a free slot must open the booking form

5.1.1. Wireframes
  mobile: wireframes/schedule-mobile.png
  desktop: wireframes/schedule-desktop.png
  adapt
```

The word `adapt` alone in the Wireframes sub-block tells the AI it may adapt the layout for mobile or accessibility.

10. Security block

The Security block is written in business-intent language — not technical language. The architect declares what needs to be protected. The AI decides how to implement it.

```
6. Security
  access: users must create an account with a username and password
  - Login must be protected against automated password attempts
  - Registration must be protected against mass account creation
```

11. Services block

```
7.1. Service SessionCleanup
  frequency: 24 hours
  time: 03:00
  - Remove user sessions expired more than 30 days ago
  - Send a cleanup report by email to the administrator
```

12. spellcomp commands

```
spellcomp new      <name>          # Creates a .spell file from template
spellcomp compile <file.spell>      # Compiles and generates files per module
spellcomp check   <file.spell>      # Validates without generating anything
spellcomp info    <file.spell>      # Displays a summary of the file
spellcomp --version
spellcomp --help
```


compile options:

```
--output <folder/> # Output folder (default: ./<project-name>/)
--semantic          # Runs semantic audit via the Claude API
```

Diagnostic symbols:

Symbol	Meaning
✖ ERROR	Prevents correct generation — must be fixed
⚠ WARNING	Ambiguous — architect should review
i INFO	Inference applied — verify if correct
✓ OK	Everything in order

13. Semantic audit

The semantic auditor is the most powerful layer of spellcomp. After building the AST, it sends the spec to the Claude API and receives a diagnostic report.

What the auditor detects:

- Gaps: entity with a status field but no rule governing its transitions
- Contradictions: two rules with conflicting behavior for the same context
- Dead references: endpoints referencing undeclared rules; screens navigating to undeclared screens
- Orphan entities: declared but with no endpoint that writes to them
- Error coverage: endpoint that fetches by ID with no 404 response declared

Activating the audit:

```
spellcomp compile schedule.spell --semantic
```

Semantic auditing is off by default. Use `--semantic` when you want full semantic validation of the spec before generating output files.

14. API authentication

spellcomp never asks you to save credentials inside project files. Authentication follows a cascade:

1. ANTHROPIC_API_KEY environment variable
↓
2. Claude Code credentials (`~/.claude/.credentials.json`)

```
↓
3. Key saved by spellcomp (~/.spell_api_key)
↓
Interactive prompt → offers to save for future use
```

Simplest setup (shell profile):

```
export ANTHROPIC_API_KEY=sk-ant-...
```

15. Modular output

spellcomp compiles and generates a folder with separate files per domain:

```
easyschedule/
├─ CLAUDE.md                ← agent entry file
├─ easyschedule-project.md  ← stack, conventions, skills
├─ easyschedule-data.md     ← database
├─ easyschedule-rules.md    ← business rules
├─ easyschedule-endpoints.md ← endpoints
├─ easyschedule-screens.md  ← screens
├─ easyschedule-services.md ← services (if declared)
└─ easyschedule-security.md ← security (if declared)
```

The entry file (CLAUDE.md or equivalent) contains:

- Index of all generated files and what each one contains
- Required and suggested skills
- Module dependency instructions
- Compiler warnings and semantic report (if any)

At the end of every compilation, spellcomp displays a reference table with the entry-file names expected by each agent — useful for teams using different agents.

16. Full example

```
1. Project
  name: "EasyScheduler"
  version: "1.0"
  description: "System for clinics to schedule appointments"

1.1. Stack
  frontend: React
  backend: Node.js
  database: PostgreSQL
  authentication: accounts with login and password
```

agent: claude-code

1.2. Conventions

code-language: english

- All services must have the Service suffix

1.3. Skills

- xlsx

2. Database

2.0. Configuration

multitenancy: none

scope: global

2.1. Table Doctor

id: integer, primary-key, auto-increment

name: text(100), required

specialty: text(100), required

license-number: text(20), unique, required

active: boolean, default: yes

2.2. Table Patient

structure: recommend

- Patient must have name, date of birth, and phone number

- SSN is optional but must be unique if provided

2.3. Table Appointment

id: integer, primary-key, auto-increment

doctor_id: integer, required

patient_id: integer, required

datetime: datetime, required

status: text, default: "Scheduled"

notes: text, optional

3. Business Rules

3.1. Rule SlotAvailable

context: Appointment scheduling

condition: Doctor already has an appointment at the same time

action: Reject the booking with a slot-unavailable message

3.2. Rule CancellationAdvanceNotice

context: Appointment cancellation

condition: Cancellation requested less than 24 hours in advance

action: Show cancellation policy warning and ask for confirmation

reason: Clinic policy to reduce last-minute no-shows

4. Endpoints

4.1. Endpoint ScheduleAppointment

route: POST /appointments
authentication: required
rules: SlotAvailable

4.1.1. Input Data

doctor_id: integer, required
patient_id: integer, required
datetime: datetime, required
notes: text, optional

4.1.2. Responses

201: Appointment scheduled successfully
400: Time slot unavailable for this doctor
404: Doctor or patient not found

5. Screens

5.1. Screen DoctorSchedule

route: /schedule/{doctor_id}
authentication: required

- Show weekly calendar with occupied and free slots
- Clicking a free slot must open the booking form
- Cancelled appointments must appear struck through, not removed

5.1.1. Wireframes

mobile: wireframes/schedule-mobile.png
desktop: wireframes/schedule-desktop.png
adapt

6. Security

access: users must create an account with a username and password

- Login must be protected against automated password attempts

7. Services

7.1. Service SessionCleanup

frequency: 24 hours
time: 03:00

- Remove user sessions expired more than 30 days ago
- Send a cleanup report by email to the administrator

About the author

Serial entrepreneur with over 30 years of experience in technology. He founded and led one of the first email marketing and digital channel automation platforms in Brazil, built from scratch in 2005 and sold to one of the country's largest Customer Experience players in 2022. Today he leads a technology group through which he develops multiple projects in digital transformation, cloud computing, and process automation, while also acting as an investor in technology businesses. Throughout his career, he has helped more than 5,000 Brazilian companies communicate better with their customers through technology..