

Spine Feature Seed

Spine Evaluation Edition 0.7.4 | Feature Seed form

© 2026 Big Intelligence AI, LLC. You may copy this form and fill it in to use with Spine.

What this form is for. A Feature Seed describes one feature in your own words: what it must let people do, who uses it, and what could go wrong. Your AI assistant builds the four-layer Spine model from it, and Spine's gate checks every step. The model can only be as complete as this form, so answer every required field, even briefly.

How to use it.

1. Fill it in, replacing each _____. Plain language is fine; you do not need modelling terms.
2. Save it as a Markdown (`.md`) file in your Spine project folder, for example `seeds/feature-seed-<name>.md`.
3. Open your AI assistant in that folder and ask it to build the model from the file, for example: *"Build the model from seeds/feature-seed-door-opener.md."*

If the feature includes a hardware or software component you procure or build as a unit, also fill in a **Component Seed** for it. A worked example of both forms, for an electric power steering system, ships with Spine.

Section 1 — Feature Identity

Feature Name *(required)*

Feature Description *(required)* — an executive summary, 2-3 sentences. This is the elevator pitch, **not** the scope statement — the binding statement of what the feature must do is **Section 2A**, and that is what Spine conducts from.

Feature Type *(required)*

- ☐ New feature — concept only; the model is built from scratch
- ☐ Existing feature, still in development — it can still be reworked
- ☐ Existing feature, in production — the model records it and reports gaps

For an existing feature: you may skip Section 4 (Brand Impact Assessment), and you must complete Section 5 (Existing-Feature Details) instead.

Target Vehicle / Product Line *(required)*

Target Market(s) *(required)*

Section 2 — Problem Statement

What problem does this feature solve? *(required)*

Who is affected? *(required)*

How is this handled today? *(required)*

What is out of scope? *(required)*

Section 2A — Feature Capabilities & Invocation *(required)*

This is the binding scope statement. Spine projects the operational model directly from this section: each capability below becomes a capability in the model, and each invocation method becomes a path into it. A capability that is not listed here does not get built — and a capability listed here that Spine fails to build is a **defect in Spine**, not in your seed. Everything in Sections 1, 2, 2B and 2C is context that informs this table; only this table is conducted from.

Write it in your own operational language. Do **not** describe a solution, an architecture, or a component — say what the system must let someone do.

Capability table *(required — one row per capability)*

#	Capability — what the system must let someone do	Actor(s)	How it is invoked	Enable/disable?
C1	___	___	___	___
C2	___	___	___	___
C3	___	___	___	___

Column rules:

- **Capability** — one verb + object per row, stated as an operational outcome ("open the garage door", "report door status to the driver"). If a row needs the word "and", it is probably two capabilities.
- **Actor(s)** — who initiates or receives it. Include non-human initiators (a backend service, a timer, a location service) — an automatic behaviour still has an initiator.
- **How it is invoked** — **every** distinct way this capability can be triggered: physical control, companion app, voice, in-vehicle HMI, automatic/environmental trigger, gesture, key fob, scheduled, remote/backend. **List them per capability, not once for the feature** — two capabilities of the same feature often have different invocation sets, and that difference is exactly what gets lost when it is stated once globally.

- **Enable/disable?** — can the owner (or an integrator) turn this capability on and off? If yes, state the **default state, who has authority**, and whether the setting **persists** across power cycles. Then complete the inverse-pair rule below for the disabled branch.

Per-capability properties *(required — one block per capability listed above)*

Three more things Spine needs about each capability. Fixed/build-calibratable/ runtime-configurable is a build-time question, not a rating — answer what you know today.

C1

- **Non-functional bound (optional):** *If omitted, Spine may propose one; you approve or reject it.*
- **Configurability:** fixed | build-calibratable | runtime-configurable
- **Can it be switched off?** yes | no — *if yes, describe the "off" behaviour*
- **Human-facing control or indicator?** none | phone_app | ivi_touchscreen | physical_control | voice

C2

- **Non-functional bound (optional):** *If omitted, Spine may propose one; you approve or reject it.*
- **Configurability:** fixed | build-calibratable | runtime-configurable
- **Can it be switched off?** yes | no — *if yes, describe the "off" behaviour*
- **Human-facing control or indicator?** none | phone_app | ivi_touchscreen | physical_control | voice

C3

- **Non-functional bound (optional):** *If omitted, Spine may propose one; you approve or reject it.*
- **Configurability:** fixed | build-calibratable | runtime-configurable
- **Can it be switched off?** yes | no — *if yes, describe the "off" behaviour*
- **Human-facing control or indicator?** none | phone_app | ivi_touchscreen | physical_control | voice

Add one more block per row if your capability table has more than three rows.

The inverse-pair rule *(required)*

Any capability with a natural inverse must have **both** directions stated explicitly, each as its own row — or the inverse must be explicitly declared out of scope **with a reason**. Silence is not a scoping decision.

Common inverse pairs: open / close · enable / disable · grant / revoke · pair / unpair · lock / unlock · start / stop · subscribe / unsubscribe · provision / decommission · mute / unmute · arm / disarm.

Why this rule exists. A seed that says the feature lets the driver "open, close, and monitor" the door can easily produce a model with an activity for *open* and none for *close*. One direction of a pair going missing is the most common way a capability disappears without anyone noticing, because the feature still looks like it works.

Inverse-pair declarations *(required — one line per pair)*

e.g. "C1 open ↔ C2 close — both in scope." / "C4 grant access ↔ revoke access — revoke is OUT of scope for this build because delegated third-party access is thrifted (see Section 2)."

Out-of-scope capabilities *(required)*

Capabilities a reader might reasonably expect this feature to have, and which it deliberately will **not** — each with a one-line reason. This is what stops Spine inventing scope, and it is what lets the coverage ledger tell "deliberately excluded" apart from "accidentally dropped".

Section 2B — Operational Context

This section describes your feature's real-world environment: who interacts with it, what happens when things go wrong, and how it behaves across its whole lifecycle, from first activation to vehicle resale.

If you're not sure about an answer, write your best guess and say it is a guess. Spine's checks will point out what is still missing.

Who else interacts with this feature beyond the primary user? *(required)*

List all roles that touch this feature in any way: service technician, fleet manager, passengers, bystanders, dealer personnel, OEM backend systems, emergency responders, regulatory bodies, third-party service providers, etc.

What happens when this feature fails or gets bad input? *(required)*

Describe 1–2 failure scenarios. For each: What does the user experience? What is the fallback? Is there a safety concern? Think about connectivity loss, sensor failure, corrupted data, power loss, and user error.

How is this feature set up, configured, and eventually retired? *(required)*

Think through the full lifecycle:

- **First use:** How does the owner activate or register the feature for the first time?
 - **Configuration:** What settings or preferences can the user adjust?
 - **Monitoring:** Does the feature produce notifications, status displays, or logs?
 - **End of life:** What happens when the vehicle is sold, the feature is deactivated, or the vehicle is decommissioned? Is there data to transfer or erase?
-

What existing vehicle systems does this feature interact with? *(required)*

List the vehicle systems this feature sends data to or receives data from. Examples: infotainment/head unit, telematics control unit, ADAS, powertrain controller, body electronics module, cloud services, mobile app, home automation systems, dealer diagnostic tools.

Section 2C — Data Analytics & Observability Baseline

The five kinds of runtime data the product should record about this feature. Answer each one, or write **N/A** with a one-line reason; a reasoned N/A is recorded as a decision, not dropped. Without this section, nothing in the model says what the product reports about itself in use.

1. Utilization (Type 1) — required *(required)*

What are the feature's operating modes? Spine instruments one usage counter per mode.

2. Method of Invocation (Type 2) — required unless single-method *(required)*

Scope for invocation methods now lives in Section 2A, per capability. This field is the **telemetry** view of the same facts: which invocation paths need a usage counter so the product can report how customers actually trigger the feature.

Confirm the set here matches Section 2A, and note any path that must NOT be instrumented (and why). If only one method exists across all capabilities, state it — the single method is still recorded, and Type 2 is skipped with that rationale.

Do not use this field to introduce a capability. Anything appearing only here and not in Section 2A is telemetry for a capability that does not exist, and will be flagged as a seed defect rather than built. *(For example, an automatic location-based trigger stated only here gets a usage counter for its setting, and the trigger itself is never built.)*

3. Performance / Prognostics (Type 3) — required if anything wears out or degrades

What performance or degradation aspects should be monitored to predict wear-out or loss of function (e.g., battery life, actuation count, signal drift, thermal margin)? Include any that apply. If nothing in the feature wears out or degrades, write N/A with the reason.

4. Driver Distraction / Notifications (Type 4) — required for any system-initiated notification *(required)*

List every notification the feature issues to the driver that the driver did not directly request (alerts, warnings, status pushes).

5. Fault Reporting / Diagnostics (Type 5) — required *(required)*

What faults should the feature detect and report as diagnostic trouble codes? State the diagnostic intent and any known fault conditions. A later failure analysis may add to this list; it does not need to be complete here.

Section 3 — Design Constraints

Include connectivity protocols, compute platforms, architecture mandates, regulatory pre-certifications. If your organization has standard architecture decisions (e.g., CAN-FD backbone, BLE for short-range), include them here — the model treats these as fixed constraints. This is background for the `non_functional` design category in Section 6, not a category answer on its own.

Section 4 — Brand Impact Assessment

These are brand attributes: how this feature affects what customers value about the product. They are **not** the `safety` and `cybersecurity` design categories, which live in Section 6.

For each of the 10 brand attributes below, indicate the impact this feature has and provide the requested detail. If an attribute is not affected, select "No Impact" and move on.

4.1 — Quality & Reliability

"Will it hold up?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.2 — Performance & Driving Dynamics

"How does it drive?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.3 — Affordability & Value

"Is it worth the money?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.4 — Safety

"Will it protect my family?" — brand perception only. **Not** the `safety` design category (Section 6).

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.5 — Security

"Is it protected from theft/hacking?" — brand perception only. **Not** the `cybersecurity` design category (Section 6).

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.6 — Technology & Innovation

"Is it modern and future-proof?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.7 — Comfort & Convenience

"Is it easy and pleasant to live with?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.8 — Design & Emotional Appeal

"Does it make me feel something?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.9 — Productivity

"Can it help me get work done?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

4.10 — Sustainability & Efficiency

"Is it responsible / cheap to run?"

Impact: ☐ No Impact / ☐ Status Quo / ☐ Improves / ☐ Degrades

If Status Quo — Brief description of what this feature delivers for this attribute:

If Improves:

Current state — How does this work today?

Future state — How should this work with the feature?

Key differentiator — What makes your approach better than alternatives?

If Degrades:

What is the tradeoff?

What benefit in other attributes justifies this cost?

Section 5 — Existing-Feature Details

Complete this section only for an existing feature (still in development, or in production). For a new feature, write "N/A — new feature".

System Architecture / Components *(required for an existing feature)*

Known Interfaces & Protocols *(required for an existing feature)*

Known Field Issues / Failure Modes *(required for an existing feature)*

Safety/Security Measures Already in Place *(required for an existing feature)*

Existing Regulatory Certifications *(optional)*

Section 6 — Design Categories

Thirteen categories. Every one must carry a **disposition**, `answered` or `not_applicable`. There is no third option: a blank category is a gap, not a light answer.

Which categories Spine treats as required:

- **Always:** `functional`, `non_functional`, `interface`, `context_boundary`, `environmental`, `regulatory`, `vv_criteria`, `robustness`.
- **At ASIL A or above:** `safety`, `cybersecurity`, `diagnostic`, `responsibility_split`. Answer them anyway if the feature has any exposure: if no ASIL is known yet, Spine reports them as "unknown", not as "not needed".
- **Recommended:** `variants`.

`not_applicable` **needs a reason** — one line is enough, e.g. *"not applicable — this feature has no external interface, because it exposes nothing outside the vehicle boundary."* Your AI assistant is told to respect a reasoned `not_applicable` and to raise any disagreement with you rather than overrule it. In this edition, Spine's seed checks count only the requirements in the model, so a required category you mark `not_applicable` is still listed as having none. Your stated reason is the record of why.

`functional` — What should it do?

- **Disposition:** `answered` | `not_applicable`
- **If `not_applicable`, why:**
- **Answer:**

`non_functional` — How well, how fast, how often?

See Section 3 for any standing platform/protocol constraints that bound this answer.

- **Disposition:** `answered` | `not_applicable`
- **If `not_applicable`, why:**
- **Answer:**

`interface` — What does it exchange with, outside itself?

- **Disposition:** `answered` | `not_applicable`
- **If `not_applicable`, why:**
- **Answer:**

`context_boundary` — Where does it operate, and what is explicitly not yours?

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

safety — Could this hurt someone, and how?

Describe the exposure in your own words — for example "commands physical motion near people" or "can leave the driver without steering assist". **Do not supply an ASIL.** The analysis determines that, and a QM result still traces back to what you wrote here.

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

cybersecurity — What would an attacker want from it?

Describe the exposure in your own words — for example "accepts commands from outside the vehicle" or "stores a credential or personal data". **Do not supply a CAL.** The analysis determines that, and a low result still traces back to what you wrote here.

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

environmental — Where does it live — heat, water, vibration, dirt?

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

diagnostic — When it breaks, how does anyone find out?

Section 2C, Type 5 already asks for the DTC-facing detail — a short answer here is enough; point back to that section if it is easier.

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

regulatory — What rules apply, or which markets is it sold in?

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

vv_criteria — How will you know it works?

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

responsibility_split — Who builds which part, and who is accountable?

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

variants — Does it differ by market, trim, or model?

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**

robustness — Does it keep its promises when something is interrupted?

Covers state that must survive a power cycle, recovery after an interrupted command, behaviour on a degraded or absent link, and what the product may legitimately forget.

- **Disposition:** answered | not_applicable
- **If not_applicable, why:**
- **Answer:**