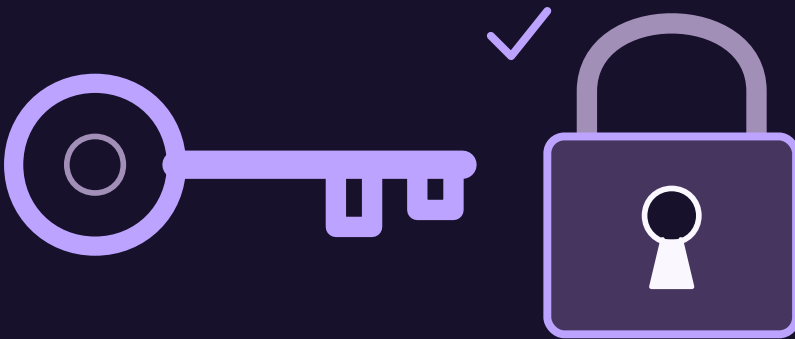




PRACTICAL AI

Design Reliable Agent Tool Contracts

Validate actions, bind approvals, and make retries understandable



Sanjaay Gowda T S

FREE SAMPLE / INTRODUCTION + CHAPTER 1 / EDITION 1.0.0

sanjaaygowdats.ai

About this free sample

Design Reliable Agent Tool Contracts

Sanjaay Gowda T S

First edition 1.0.0, 2026-10-13

Copyright 2026 Sanjaay Gowda T S. This is a labelled free sample, not the complete ebook. You may share this sample unchanged. It contains the cover, sample permissions, contents, introduction, Chapter 1 and a closing product link. The remaining chapters, worked solutions and companion files are not included. The full ebook text, diagrams and artwork have separate single-reader terms; this sample permission does not authorize sharing the full purchased PDF or EPUB.

The companion source code, tests and fictional data are separately MIT licensed. You may adapt or share that project with its included licence notice. MIT does not apply to the ebook text or book artwork. The books and companion projects have no technical activation requirement or buyer-specific watermark.

This first edition was prepared with AI assistance. The case studies are fictional teaching examples. Any measured outputs come from the supplied examples, not production customer data. The teaching scope, assumptions and limitations are explained in the chapters. Independent human technical, editorial and assistive-technology reader review is pending. Automated tests and accessibility checks do not establish certification or suitability for every use.

Run examples in a separate working folder using the supplied synthetic data. Review inputs and calculations before adapting the techniques to real operations. Model software, service access, subscriptions, ongoing updates and consulting are not included.

Author website: sanjaaygowdats.ai

Ebook terms: sanjaaygowdats.ai/ebooks/license

Inside this free sample

Start here.....	4
1. Begin with effects and invariants.....	5
Continue with the full guide	7

This free sample contains the introduction and Chapter 1 only. The closing link opens the complete guide. The companion project and worked solutions are supplied with the complete ebook.

Start here

An agent tool contract is the boundary between a proposed action and an effect that your application permits. The contract must help two callers: the model deciding what to request, and the program deciding whether to execute it. A sentence such as "reserve inventory carefully" serves neither caller well. It leaves the quantity range, authority, replay behavior, and failure meaning to interpretation.

This book builds a small, fictional inventory boundary. A caller can inspect stock or reserve between one and ten lamps. Reservations need a trusted approval record matching the subject, item, and quantity. Repeating an identical successful request returns a receipt instead of subtracting stock again. Reusing the same request identity for different content is rejected. These are deliberately narrow rules, which makes their consequences inspectable.

The companion project uses Python 3.10 or later and only the standard library. It makes no model calls, sends no network traffic, and changes no external inventory. Every subject, ticket, request, and stock quantity is synthetic. The project is an executable teaching fixture, not an authentication system or a production reservation service.

Run `python main.py --input example-input.json` from project, then run `python -m unittest discover -s tests -v`. Compare the parsed result with `expected-output.json`. Read the implementation alongside Chapters 3, 4, and 6. You will finish with a contract worksheet, a failure vocabulary, and tests that inspect behavior at the boundary.

The central question throughout is concrete: after this result, what may the caller safely do next? A well-designed contract makes that answer visible without requiring the caller to guess whether a side effect occurred.

1. Begin with effects and invariants

Name the effect before naming the tool

Start with a small inventory table: lamp has eight available units. A reservation reduces that number and creates an obligation to hold units for a particular buyer. Those two effects should be part of one conceptual operation. If your real system subtracts stock but fails to create the hold, a valid-looking response can conceal an inconsistent business state.

The teaching project models only the quantity change and its receipt. Write that limitation beside the example. It prevents a reader from assuming that a return value proves a warehouse commitment. In a real application, the effect may also involve expiry, payment authorization, shipment allocation, and notification. Adding those responsibilities expands the contract and the failure space.

Choose tool names that reveal scope. `lookup_stock` describes a read. `reserve_stock` describes a mutation. A broad tool named `manage_inventory` invites loosely specified commands and makes permission boundaries harder to review. Two narrow operations let you give the read and the mutation different authority rules.

Turn promises into invariants

An invariant is a condition that must remain true across permitted transitions. For this example, stock remains a nonnegative integer. A rejected request leaves stock unchanged. A successfully replayed reservation does not consume additional units. An approval for two lamps cannot authorize three lamps. Each promise can be demonstrated with a before state, request, and after state.

Invariant	Example probe	Observable evidence
Stock never becomes negative	Reserve two when one remains	Rejection and unchanged stock
Authority matches action	Use a two-unit approval for three	Approval mismatch error
Replay has one effect	Submit the same successful request twice	Second result marks replay
Unknown fields have no meaning	Add <code>admin: true</code> to arguments	Field rejection

Do not put every product wish into the invariant list. "Helpful" is too broad for an executor. "Quantity must be an integer from one to ten" is actionable. User experience goals still matter, but they belong beside, rather than inside, the executor's structural checks.

Draw the trust boundary

The model proposes a tool name and arguments. A trusted application supplies the authenticated subject and retrieves approval evidence. The executor checks both. In the fixture, the subject and approval table arrive in the same input file because there is no identity service. That convenience is safe only within the fictional demonstration. Copying the shape into an application without moving subject resolution to trusted code would let a caller impersonate another buyer.

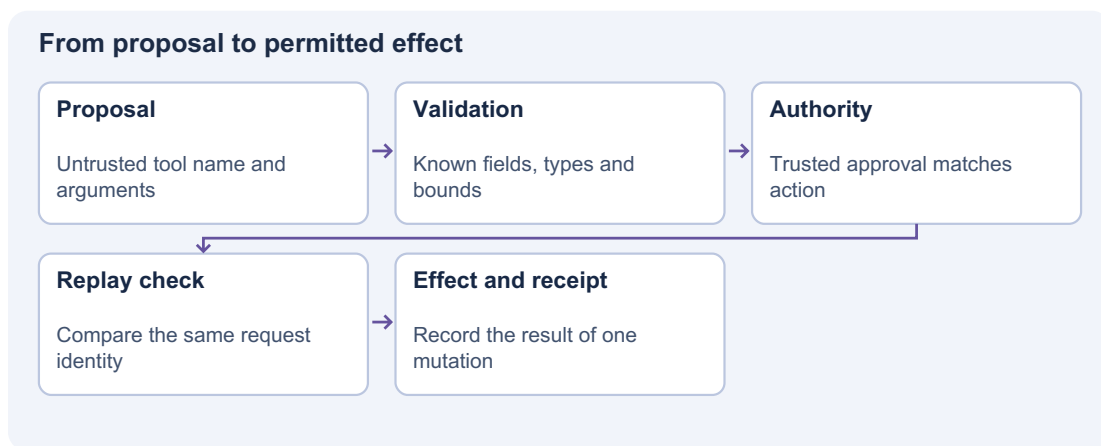


Figure 1. A proposed action passes structural validation, authority matching, and replay checking before inventory changes and a receipt is returned.

Exercise 1

list three invariants for a tool that schedules a maintenance visit. Identify which input values must come from trusted application context rather than from the model's arguments.

Continue with the full guide

This free sample ends after the introduction and Chapter 1. The complete guide contains nine teaching chapters, worked exercise solutions, primary references and a separate Python companion project. The full PDF, reflowable EPUB and project files are supplied together.

Use the complete guide to work through the remaining exercises and inspect the supplied examples. The companion code and fictional data are separately MIT licensed; the complete ebook has the single-reader terms described on its product page.

The examples are teaching fixtures, not production customer results. Independent human technical, editorial and assistive-technology reader review remains pending. Software access, subscriptions, consulting and guaranteed results are not included.

[View the complete guide and purchase options.](#)

You may share this labelled free sample unchanged. Keep the full purchased PDF and EPUB private. Thank you for reading.