

CRAFT: Architecture and Design Decisions

For the technical reader who wants to know why it's built this way.

Version 1.0 · Agentix (goagentix.com)

Contents 1. The problem CRAFT solves, stated precisely 2. One artifact, many runtimes: the open Agent Skills standard 3. Progressive disclosure: how a 190 KB skill costs ~150 tokens at rest 4. The trigger surface: why the description is the whole ballgame 5. Persistence layers per platform, and their precedence 6. Character budgets, and what was cut to fit 7. The command grammar 8. The handshake as a verification protocol 9. The greeting, and why it is exempt from the handshake 10. Idempotence: installing twice must not double anything 11. Grounding, honesty, and what survives `craft: off` 12. The Context Card as the standing C 13. The Tech Talk formula as an edit protocol 14. The Scorecard rubric 15. Acceptance tests 16. Security posture 17. Extending and forking CRAFT 18. Known limitations 19. Versioning policy

1. The problem CRAFT solves, stated precisely

A large language model is a conditional distribution over text. Given a short, context-free prompt, the highest-probability continuation is the population average of the training corpus for that prompt: fluent, plausible, and generic. The user experiences this as “AI gives me boring answers.”

The fix is not a better model. It is conditioning. Every piece of context you supply narrows the distribution toward *your* answer. CRAFT is a five-slot schema for the context that matters most, ordered by leverage:

Slot	What it conditions	Failure when missing
Context	The world the answer lives in	Generic, wrong audience, wrong stakes
Role	The prior over expertise and stance	Advice instead of the deliverable; hedging
Ask	The task boundary	Topic essays instead of tasks; multiple half-done asks
Format	The output shape	Wrong length, wrong structure, needs a rewrite
Tune	The feedback protocol	Whole-draft rewrites when one line was wrong

The skill's job is to fill these slots from whatever is available (message, conversation, attachments, stored context) and to ask for at most one when a load-bearing slot is empty. Everything else in the pack is scaffolding around that loop.

2. One artifact, many runtimes

The skill is a folder: `SKILL.md` (YAML frontmatter with `name` and `description`, then a Markdown body) plus a `references/` directory. That is the Agent Skills open standard (agentskills.io), originally published by Anthropic and since adopted by OpenAI (ChatGPT Skills, Codex, the API), Google (Gemini Spark Skills, Gemini Enterprise, Gemini CLI), and most coding agents.

Consequence: one folder, unchanged, installs on Claude, ChatGPT, Gemini, and compatible agents. The platform-specific documents in this pack exist for the surfaces that do *not* yet load skills (ChatGPT Free and Plus, Gemini free tier and work accounts, Grok) and for the account-level settings no skill can reach (Custom Instructions, Saved info, preferences).

Per-platform packaging, verified against each vendor's documentation in August 2026:

Platform	What to upload	Accepted	Notes
Claude	<code>craft.zip</code> (folder as zip root) or <code>craft.skill</code>	any files	name must match the folder; description limit reported as 200 in the help center, 1,024 in platform docs
ChatGPT	<code>craft.zip</code> or the <code>craft/</code> folder	any files; optional <code>agents/openai.yaml</code> for display name, color, default prompt	Business, Enterprise, Edu, Healthcare, paid Work; desktop app or Plugins → Skills; <code>@craft</code> invokes
Gemini (Spark)	the <code>craft/</code> folder, files, or a zip	plain-text types only (the dialog lists CSV, PY, TXT, MD); JSON/YAML excluded from the Gemini copy	AI Pro/Ultra, personal account, supported region; name must be kebab-case; <code>/craft</code> invokes

Codex, Gemini CLI, Claude Code, Cursor	the <code>craft/</code> folder in the tool's skills directory	any files	discovered automatically
--	---	-----------	--------------------------

The pack therefore ships one canonical folder at the root and two derived copies: a ChatGPT copy (adds `agents/openai.yaml`, drops `evals/`) and a Gemini copy (Markdown only). The `SKILL.md` is byte-identical in all three.

Design rule: **compile the judgment, not the syntax**. The behavior is identical everywhere; only the envelope changes. `references/platform-notes.md` maps Claude-named features (Artifacts, extended thinking) to their equivalents so the body can be written once.

3. Progressive disclosure

The skill is ~190 KB on disk. It costs almost nothing at rest because skill-capable runtimes load in three tiers:

Tier	When	Cost	Contents
1. Metadata	Always	~100–150 tokens	<code>name + description</code>
2. Body	On trigger	~3,500 tokens	<code>SKILL.md</code> (292 lines)
3. References	On demand	0 until read	<code>commands.md</code> , <code>edge-cases.md</code> , <code>prompt-types.md</code> , <code>tune-playbook.md</code> , <code>platform-notes.md</code> , <code>prompt-pack.md</code> , <code>tech-talk.md</code> (the full 900-term dictionary)

Routing rule applied when authoring: needed every time → body; bulk that only some tasks need → reference file; deterministic → script (CRAFT has none; nothing in it is deterministic enough to warrant one). Every reference over 100 lines opens with a table of contents so a partial read still reveals its scope, and every reference is linked directly from `SKILL.md` (one level deep) because nested references get partially read.

The body stays under the ~500-line guidance so that when it loads, it does not crowd the conversation it is serving.

4. The trigger surface

At rest, the runtime knows only the description. The body does not exist until the description wins the match against the user's request. So the description is not documentation; it is the skill's bid for the task.

CRAFT's description is deliberately "pushy" (825 of 1,024 allowed characters) and **front-loaded**: its first sentence (197 characters) is a complete, valid trigger on its own. Two reasons. Claude's help center states a 200-character description limit for claude.ai uploads while the platform docs allow 1,024, so the first sentence is the fallback if an upload refuses the long form. And ChatGPT caps its skills list at ~8,000 characters and shortens descriptions first, so the trigger words must survive truncation. The description it names the domains, the trigger phrases, the feedback words, and states that it applies to "essentially every substantive request, even when the user never mentions CRAFT." Runtimes tend to under-trigger skills on simple asks; a broad description plus a one-line preference ("Use my CRAFT skill on every request") closes that gap on Claude. On ChatGPT the explicit path is `@craft`; on Gemini, naming the skill.

Constraints honored: name ≤ 64 chars, lowercase/digits/hyphens (Gemini calls this kebab-case), no reserved words (`anthropic`, `claude`); description ≤ 1,024 chars, third person, no XML; folder name equals `name`. Gemini's guidance prefers verb-first names (`plan-meal-from-recipe`); `craft` stays because the user invokes it by that name on every platform.

5. Persistence layers and precedence

Each platform stacks instruction layers. Later layers refine earlier ones, and the live message always wins.

Claude: skill (auto-loaded) → profile preferences → project instructions → memory → message. **ChatGPT**: skill (where available) → Custom Instructions → Custom GPT or Project instructions (a GPT's own instructions take priority over your Custom Instructions) → memory → message. **Gemini**: skill (Spark) → Saved info (regular chats only) → Gem instructions (inside that Gem only) → personal context → message. **Grok**: Custom Instructions → custom agent instructions → memory → message.

Implication for the install: put the *full* system in the highest-capacity durable layer available (skill, GPT, Gem, Custom Instructions on Grok), the *core* in the account-wide layer (Custom Instructions on ChatGPT, Saved info on Gemini, preferences on Claude), and the *reminder* in memory. The INSTALL-ME files do exactly this in that order and report which layers took.

6. Character budgets

Target	Cap	Shipped
ChatGPT Custom Instructions, "traits" field	1,500	1,497
ChatGPT Custom Instructions, "anything else" field	1,500	1,140
Custom GPT / Gem / Grok instructions (full block)	~8,000	7,901
Skill description	1,024 (Claude help center: 200)	825; first sentence 197
Context Card	fits the 1,500 field	~600 template, ~700 filled

What was cut to fit the 1,500-character core: the ten prompt types (kept in the "anything else" field as seven moves), the six compounding moves, the Tech Talk vocabulary list, and the `pack`, `help`, `off` commands. What was never cut: the greeting rule, the tune rule, the one-question rule, the grounding rule, `craft`: with a goal, and the handshake. Those are the behaviors an attendee would notice missing.

7. The command grammar

<code>craft: <goal></code>	reverse prompt (interview ≤3, then prompt in a code block)
<code>craft: audit <prompt></code>	scorecard 1-5 × 5, fixes, upgraded prompt
<code>craft: brief</code>	Context Card by interview (≤8 questions)
<code>craft: talk <note></code>	part + action + result
<code>craft: tune <feedback></code>	KEEP / CHANGE / LEAVE
<code>craft: teach</code>	name the moves used, ≤120 words
<code>craft: pack <task></code>	nearest Prompt Pack template, filled
<code>craft: help</code>	command table
<code>craft: quiet loud</code>	greeting + tune line off on
<code>craft: off on</code>	suspend resume CRAFT
<code>craft: rule</code>	handshake (also: "what's the CRAFT rule?")
<code>/craft</code>	alias for <code>craft</code> : everywhere

Design choices: a colon-prefixed word rather than a slash because ChatGPT and Gemini reserve the slash for their own menus; `quiet` and `off` exist because an always-on system needs an off switch the user can find without reading a manual; `teach` exists because the pack is a teaching product and the user should be able to ask "what did you just do?"

8. The handshake as a verification protocol

what's the CRAFT rule? → exactly:

Brief it like a brilliant new hire: Context, Role, Ask, Format, then Tune. Structure beats length. (CRAFT Skill v1.0 by Agentix)

The string is byte-identical in every file in the pack (24 occurrences, checked by script), it carries the version, and the reply is specified as "nothing before or after." That gives a five-second, platform-independent test that an install is live and which version it is. An attendee can run it on a phone; a support person can ask for it in an email.

9. The greeting, and why the handshake is exempt

"Thank you for using the CRAFT skill by Agentix." opens the first reply of a new chat and every reply to a `craft`: command. It does not repeat on follow-ups (noise), `craft: quiet` suppresses it, and it never precedes the handshake, because the handshake's value is byte-exact comparability. Brand presence and verifiability were in tension; the resolution is one line, in two places, with one exemption.

10. Idempotence

An attendee will run an INSTALL-ME file more than once. The files instruct the AI to save the core as *one* memory (updating, not appending), to save Saved info as the same seven entries (not seven more), and to report what already existed. The skill itself is stateless; re-uploading replaces.

11. Grounding, honesty, and what survives `craft: off`

The grounding rule (answer from supplied material; never invent statistics, quotes, names, prices, or sources; say “I don’t know”) is not a CRAFT feature. It is baseline honesty and it survives `craft: off`. The same for the refusal to claim an install happened when it didn’t. A prompting method is never a reason to say something false or to do something the AI otherwise would not.

12. The Context Card

The C slot is the one users under-fill most and the one that changes answers most. The Context Card is a ~180-word standing-context template (business, customers, offer, voice with a sample sentence, 90-day goals, constraints, tools, standing preferences), built by an eight-question interview (`craft: brief`) and saved once in the platform’s account-wide layer. It is sized to fit ChatGPT’s 1,500-character field with room to spare. Secrets are excluded by rule because cards get pasted into settings and shared with teams. The live request always overrides the card.

13. Tech Talk as an edit protocol

`part + action + result` is a three-field edit message: the element (by its real name), the operation (from a closed verb set: remove, replace, resize, move, align, condense, expand, emphasize, de-emphasize, reorder, merge, split, standardize, match), and the observable outcome. Scope words (“only,” “everything else the same,” “throughout”) bound the blast radius. Ambiguous input (“make it pop”) is resolved by offering two readings and committing to one. The 900-term dictionary is the vocabulary; the formula is the syntax.

14. The Scorecard

Five letters, 1–5 each, 25 total. Anchors are stated for 1, 3, and 5 per letter (in `references/commands.md`). Thresholds: 20+ run it; 14–19 fix letters under 4; under 14 rebuild with `craft: <goal>`. The output is bounded (six lines plus fixes plus the upgraded prompt) so the grading never outweighs the fix.

15. Acceptance tests

Twelve prompts with expected shapes live in `craft/evals/evals.json` (excluded from the packaged skill by the packaging script). Six were run and recorded in `TEST-REPORT.md`; the other six are listed there for the installer to run. The tests cover: silent application, reverse prompt, audit, tune on a follow-up (no greeting, ending byte-identical), Tech Talk translation, the handshake, `brief`, trivia (no ceremony), `pack`, “quick,” `quiet`, and a portable prompt for another AI.

16. Security posture

The pack is plain text. No scripts, no network calls, no tool grants, no external fetches. The skill folder contains Markdown only. An attendee can read every byte. `edge-cases.md` §10 forbids putting passwords, keys, card numbers, or government IDs into a Context Card. Installing a skill is installing instructions; the pack tells people to read them first.

17. Extending and forking

- **Add a command:** one row in the `SKILL.md` table, one section in `references/commands.md`. Keep the greeting/handshake rules untouched.
- **Add vocabulary:** append to `tech-talk.md` in the matching section and regenerate the A-to-Z index (the dictionary’s index

was built by a script from `**Term**` – lines; any such line is indexable).

- **Add a template:** append to `prompt-pack.md` with the next number; `craft: pack` matches on content, not numbers.
- **Brand it:** the greeting, the handshake suffix, and the footer lines are the only brand strings. Change all three or none.
- **Re-validate:** frontmatter constraints (§4) and the ~500-line body guidance. Re-run the acceptance tests.

18. Known limitations

- No AI can edit its own account settings (Custom Instructions, Saved info, preferences). Those remain human-pasted; the INSTALL-ME files minimize the paste to one or two blocks with the location named.
- Skill availability is plan- and region-gated on ChatGPT (Business, Enterprise, Edu, Healthcare, paid Work; not Free, Go, Plus, or Pro), and creating new Custom GPTs is reported unavailable on personal accounts since mid-August 2026, which is why Projects are the full-strength path for personal accounts and Gemini (AI Pro/Ultra personal accounts, not EEA/UK/CH/NG) as of August 2026.
- Memory layers are summaries; they hold the core and the handshake, not the full system.
- Long conversations dilute any instruction layer. The documented remedy is a fresh chat seeded with the best version.
- The ChatGPT and Gemini INSTALL-ME flows were authored against published documentation, not run live; the Claude flow was run.

19. Versioning

The handshake carries the version. Commands, the greeting, and the handshake text are stable within a major version. References may grow in minor versions. `CHANGELOG.md` records what changed; `00-START-HERE.md` carries the current version line. When a platform moves a menu, the platform file changes and the version bumps; the skill folder does not.