Episode Book

EPISODE #6282

How Steering Messages Reach Active Coding Agents

PUBLISHED	26 October 2026 at 11:14 UTC
DURATION	26:52
LISTEN	myweirdprompts.com/episode/steering-coding-agents-protocols-72100/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	openai/gpt-6.1-sol
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.22
SOURCES	10 cited by the research (listed at the end)
BOOK GENERATED	27 October 2026 at 06:27 UTC

SYNOPSIS

Sending a correction while an AI coding agent is in flight reveals the critical gap between receiving a message and acting on it. Looking closely at protocols like the Responses API steering events, Codex app server turns, and CLI harnesses, this episode examines how systems queue user input mid-task. Discover why steering produces new continuation responses rather than rewritten history, how in-flight tool boundaries function, and why durable software wrappers matter more than an unbroken model mind.

TRANSCRIPT



Corn

A coding agent can still be carrying out your last instruction when you realise it was the wrong instruction. Daniel wants to know what happens when you send the correction without stopping the work. Codex calls it steering. The mystery is where that message waits, and whether the conversation we think of as the agent is actually the durable wrapper managing a succession of workers.



Herman

The difference becomes visible the moment you type, stop changing that file, while a tool is already changing that file. Your message can arrive successfully without preventing the change. Those are separate events.



Corn

A read receipt for your impending regret.



Herman

There is an actual acknowledgement boundary here. The system can tell you it received your correction before the model has used it. That gap matters far more than the little animation in the chat.



Corn

Daniel's bigger ambition is the version where he can keep supplying ideas while bounded jobs go off into fresh contexts and report back. Nobody drops the thread, nobody mistakes a passing thought for permission, and nobody needs to supervise every keystroke.



Herman

You want that last part particularly badly.



Corn

Autonomous work should not require my attendance.



Herman

Then the steer is the useful place to look, because it exposes what is holding the message while execution continues.



Corn

Before you populate this with seventeen employees, Herman, I suspect we need two identities. The conversation that stays, and whatever happens to be running inside it.



Herman

The thread and the turn. Codex's app server makes that distinction explicit. Its steering operation takes a thread identifier, your input, and the turn identifier you expect to be active. You address a particular turn within a particular thread.



Corn

Why tell it which turn I expect? Surely the server knows what it's doing.



Herman

It knows its own current state. It doesn't automatically know which state your screen was showing when you pressed send. An expected identifier gives the protocol something concrete to check rather than silently assuming your message belongs wherever execution has reached.



Corn

My instruction arrives late and gets assigned a new employer.



Herman

That's the class of confusion you want to prevent. Now, a turn is an execution lifecycle, not necessarily a separate agent, model, or process. I should narrow that immediately. Calling turns disposable workers is useful shorthand, but it can imply a physical architecture we haven't established.



Corn

Good. You were about to give every response its own desk.



Herman

The desks are logical.



Corn

They always are until the invoice.



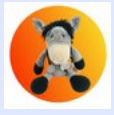
Herman

A thread can persist across many turns. The surrounding software stores history, supplies tools, handles approvals, and constructs what the model sees next. The visible continuity belongs partly to that software. It doesn't require one continuously thinking entity to remain alive behind the chat.



Corn

Nor does a thread identifier prove there's an intelligent supervisor running constantly. The wrapper could be ordinary application logic until a model is called.



Herman

Yes. Orchestration describes a responsibility, not a requirement for another model. A deterministic service can receive events, queue them, and decide that a continuation is now allowed.



Corn

That makes the wrapper intuition useful without turning it into a secret committee.



Herman

What would you expect the interface to show after accepting a correction, Corn?



Corn

Received, then applied. Separate states. If it only shows my message underneath the work, I'm going to assume the next thing it does reflects that message. Especially when the button says steer rather than leave correspondence.



Herman

That expectation is precisely where the protocol becomes revealing.



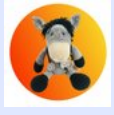
Herman

In the Responses API, you start a response, wait for confirmation that it exists, and then send a steering event over the same WebSocket connection. You identify the response being steered and provide the new input. There isn't a second hidden chat session required for that.



Corn

Same connection, different event. Side channel in the logical sense, rather than another wire.



Herman

Right... the steering event itself is tightly constrained. It carries its event type, the previous response identifier, and input. You're not submitting a whole replacement request full of changed settings through that event.



Corn

Which is sensible. Otherwise, please use the other database could accidentally also mean, please change every generation setting.



Herman

The server then acknowledges acceptance. That acknowledgement includes a sequence number and a steer object with its own identifier. Your correction has become a first class queued object. OpenAI's wording is unusually helpful here. Acceptance means the input is queued, not that the model has acted on it.



Corn

Received and obeyed have finally separated households.



Herman

The next step is a continuation. The API automatically creates a new response incorporating your update unless it needs a tool result or approval from your application. Before that continuation begins, the server finishes the current output item and hosted tool work already running.



Corn

Let me test the uncomfortable version. I ask it to inspect the repository, it starts an expensive operation, and I say, actually, only inspect the documentation. That narrower instruction doesn't travel backwards into the operation already underway.



Herman

Correct. Steering doesn't rewrite output already delivered, undo earlier actions, or cancel tools that have started. You may get a better next action. You don't get a revised past.



Corn

The undo button remains unemployed.



Herman

There are two possible endings for the original response. If steering cuts it short, it ends as incomplete with the reason recorded as steered. If it finishes normally before the handoff, it retains its completed status, and a steering continuation can still follow.



Corn

Incomplete sounds like a failure when you're reading a dashboard. Here it can mean the system did exactly what you asked.



Herman

A very important distinction for anyone counting errors. You need the reason, not just the terminal status. Otherwise a successfully redirected response looks like a broken response.



Corn

You read status reports the way you used to listen to symptoms. Fine is never enough information.



Herman

Fine is an adjective with very poor diagnostic specificity.



Corn

There he is.



Herman

The continuation inherits the original request settings, but token limits and tool call limits apply separately to each response. Continuity of conversation doesn't automatically mean one unchanged accounting boundary.



Corn

That deserves more attention than the animation. A user sees one job, while the service may be recording successive responses with separate limits.



Herman

Yes, although I wouldn't infer a particular bill from that alone. The documented point is that those limits reset per response. The practical consequence is that your harness should track the whole job as well as the individual responses.



Corn

Otherwise a job with several corrections looks wonderfully well behaved when you inspect each little piece.



Herman

Do you buy the phrase noninterrupting steering after hearing that?



Corn

At the tool level, yes. At the task level, only with qualifications. It means the system doesn't necessarily abandon an operation halfway through. It doesn't mean execution proceeds untouched. A successor response with altered instructions is a change in the run, however politely it happens.



Herman

That's how I read it too. A graceful handoff at a permitted boundary. We know the public event contract. We don't know the private server implementation well enough to describe the exact internal machinery that stitches those responses together.



Corn

There is also a product distinction here. The Responses API gives us this explicit successor response mechanism. Codex's app server addresses an active turn. We shouldn't declare those identical internals because both surfaces offer steering.



Herman

I was about to compress them too aggressively. The API demonstrates that queueing and continuity can coexist with a new response. Codex descriptions often say the model receives a message midexecution and adapts without abandoning context. That's compatible with a continuation that inherits state, but it doesn't establish the same implementation.



Corn

Midexecution describes when I sent it. Not necessarily the instant the model read it.



Herman

Precisely the distinction. The hard limit is an in flight call. A correction can be accepted while a tool call is underway, but it doesn't change that call. The orchestration has to wait for a boundary where the new instruction can influence subsequent work.



Corn

Suppose the tool returns something the correction has made irrelevant. Does the successor still need to see it?



Herman

Often it needs enough information to understand what happened. Irrelevant to the new objective doesn't mean irrelevant to state. If files changed, or a request was sent, the continuation needs that reality. I wouldn't prescribe the exact context packaging without seeing the harness.



Corn

I would rather waste a paragraph on an obsolete result than have it believe nothing happened.



Herman

That preference changes the architecture. You need action history alongside current intent. Updating the objective can't erase the record of execution.



Corn

The keyboard shortcuts expose another distinction. The documented Codex CLI paths give Enter the immediate steering role and Tab the queued followup role. Enter means send it into the current turn. Tab means wait until the current operation completes.



Herman

Immediate there still means subject to the boundary we've discussed. It doesn't acquire the power to interrupt an API call. Real time steering became the default in the October sixteenth release, but interfaces and versions matter when someone is relying on a particular key.



Corn

Two keys, two meanings, one substantial opportunity for me to press the wrong one.



Herman

Would you actually use the queued path, or would every thought feel urgent?



Corn

Most of mine would be queued. Add a changelog when you're done is different from stop publishing credentials. The interface ought to let me express urgency without pretending every message deserves the same scheduling.



Herman

It also needs to distinguish a new job from a correction to this job. Put it in the backlog, modify the next step, and stop execution are separate intentions.



Corn

A chat box makes them look suspiciously alike.



Herman

The transport can accept all of them as text. The harness still has to interpret and schedule them. That is where Daniel's wrapper idea becomes more than terminology.



Corn

It's that time again, I believe. Is someone on the line, Mum?

D

Dorothy

Hello dear, it's your mother, with Dorothy's History of Toast.

D

Dorothy

The first electric toasters only browned one side of the bread at a time, so someone had to stand there and turn every slice over by hand.

D

Dorothy

You'd think they'd have sorted it out beforehand, wouldn't you, but that's the way of things, half a job and a person left standing over it.

D

Dorothy

Anyway, I've got a load of washing on, so I'll let you get back to it. Bye now, sweetheart.



Herman

Lovely to hear you, Mum. The harness has to interpret and schedule the message even when receiving it is easy.



Corn

Claude Code is quite explicit about that layer. The model reasons, tools act, and the surrounding software supplies the tools and manages the context the model sees. The thing I type into is connected to all that machinery, rather than directly to an uninterrupted mind.



Herman

Its architecture page calls that surrounding layer the agentic harness. A useful definition because it includes context management, not just tool access. Giving a model a hundred tools doesn't decide which part of the last hour belongs in its next request.



Corn

My concern with the firehose user is that receiving everything is easy compared with deciding what survives. If I say, explore SQLite, then later, deploy on our existing Postgres system, those cannot just remain equally prominent instructions.



Herman

The harness needs current task state. Not merely an append only transcript. Well... the transcript can be append only, but the working representation of the objective needs revision.



Corn

Keep the evidence, revise the assignment.



Herman

That would be my design preference. A record of messages and actions, plus explicit current constraints and task status. That's an architectural recommendation, not a promise that every product maintains those objects exactly that way.



Corn

Now you can have your employees.



Herman

OpenAI's beta Multi-agent facility provides a concrete tree. Enabling it makes the root agent eligible to create subagents. A researcher can sit under the root, a reviewer beside it, and a tester under the reviewer. Those hierarchical identities let the system address distinct branches of work.



Corn

Already a tester reporting to a reviewer. Your logical desks have acquired management.



Herman

The default concurrency limit is three subagents across all descendants, excluding the root. That's recommended for most workloads. A tree doesn't mean unlimited simultaneous execution just because you can draw another branch.



Corn

Three workers and an organisational chart that can outgrow them.



Herman

The guide doesn't impose a fixed tree depth or total agent count, either. That's a capability boundary, not a recommendation to explore the largest imaginable org chart.



Corn

You say that with the restraint of someone opening another terminal.



Herman

I can be interested without enabling it.



Corn

That's your experimental feature.



Herman

The collaboration actions are hosted. They appear as multi agent call items, and the application mustn't execute them itself as though they were ordinary application tools. Two relevant actions are spawning an agent with an initial assignment and sending a queued message to an existing agent without starting a new turn.



Corn

Meaning I can't see a collaboration event, helpfully perform it myself, and leave the service to perform it again.



Herman

That's the danger of confusing notification with responsibility. The root can create workers, supply additional information, wait for results, and synthesise an answer without your application implementing each coordination step.



Corn

Does the root get everything I've said? Or does the researcher get everything too?



Herman

Those are different questions. The point of bounded assignments is focused context. A worker maintains its own context rather than absorbing every unrelated line of discussion. Fresh context doesn't mean an empty assignment. It still needs the objective, relevant constraints, and enough material to do its job.



Corn

Good. Clean context can become a euphemism for forgetting the important restriction. If the parent knows the repository is read only, a child investigating a migration needs that restriction too.



Herman

What would you want returned from that child, Corn? Full transcript, short answer, something between?



Corn

A short report with recoverable evidence. What it found, what it changed, what it couldn't verify, and where the supporting material lives. I don't want its entire search history in my main conversation. I also don't want confident prose to be the only surviving record.



Herman

Claude Code's standard subagent pattern returns a summary from the worker's own context. Results go to the caller, and the main agent manages the work. That reduces how much of the worker's activity enters the parent's context.



Corn

Summary is a communication format, though, not a guarantee of completeness.



Herman

Nor does returning a summary imply the whole system destroyed every underlying trace. It tells us what comes back through that ordinary result channel. We shouldn't overstate it as the orchestrator being physically incapable of accessing anything else.



Corn

Especially if the worker says, all tests passed, and the summary omits that it only ran the easiest one.



Herman

A useful report separates outcome from coverage. Passed the parser tests is much stronger than everything looks good, because the caller can reason about what remains untested.



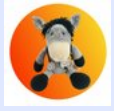
Corn

There's your clinical distrust again.



Herman

It is a transferable skill.



Herman

The parent also has a cost before any worker starts. Claude Code warns when combined descriptions of nonbuilt in subagents exceed fifteen thousand tokens. Those descriptions help select agents, so they occupy context in the coordinating session.



Corn

Fifteen thousand tokens of staff introductions.



Herman

A warning threshold, to be clear, not a universal reservation of that amount. A small collection of concise descriptions costs less. Registering a giant catalogue can make the coordinator carry a considerable directory before it has received useful work.



Corn

Which makes adding another specialist less obviously free. You're improving the choice of worker while making the chooser read more.



Herman

This is why naming and describing capabilities matters. Ambiguous overlapping descriptions ask the parent to spend reasoning deciding who should receive a task before the task has even started.



Corn

I wouldn't solve that by inventing a recruitment agent.



Herman

You anticipated me unfairly.



Corn

Incorrectly?



Herman

No.



Corn

The ordinary subagent arrangement still funnels results through the main session. Even perfect summaries accumulate. Daniel can supply ideas faster than the parent can decide which reports alter which assignment.



Herman

That's the star topology's bottleneck. Claude Code's experimental agent teams loosen that structure. A team lead coordinates and synthesises, but teammates have independent context windows and can communicate directly with one another. You can also address a teammate without going through the lead.



Corn

I like the access. I don't automatically like the consequences. If I tell a tester to change scope, the reviewer may still be relying on the original scope.



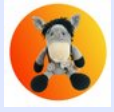
Herman

Shared task state becomes more important. Teams use a shared task list, and teammates claim work. Direct communication helps avoid forcing every exchange through one context window, but it creates more coordination to account for.



Corn

A mesh doesn't remove the need for someone to know what is true.



Herman

It spends significantly more tokens, too. Teams are experimental and off by default, with known limitations around resumption and shutdown. I'd be cautious about treating their visible flexibility as proof of durable unattended operation.



Corn

You would still try it before dinner.



Herman

After saving the work.



Corn

Progress.



Herman

Standard subagents work within one session. If you want many independently running sessions, you need background agents, cross session communication, or teams. That's a different lifecycle requirement, not merely a request for a wider context window.



Corn

Skills belong in a different category again. Loading instructions for a capability doesn't, by itself, create another worker with its own lifecycle. Background describes when work runs. Subagent describes a delegated execution context. Those features can cooperate without being synonyms.



Herman

That separation helps with Daniel's firehose. The durable conversation can accept an idea now, classify it as a later task, and delegate it when capacity exists. It doesn't need to force every new sentence into the context of every active worker.



Corn

Nor should it. If a worker is fixing authentication, my thought about a nicer settings page shouldn't become a new instruction halfway through the patch.



Herman

My preferred implementation would attach messages to objectives and revisions. A report comes back against the assignment it actually received. If the user changed the objective while it worked, the coordinator can recognise that the report answers an earlier version.



Corn

That is the detail I most want from this architecture. Not merely completed, but completed what. A beautifully executed obsolete assignment is still obsolete.



Herman

You also need clarity about who may change shared files. Separate context windows don't create separate realities. Two workers can independently reason well and still collide over the same repository.



Corn

Parallel thought, serial argument with Git.



Herman

Read only exploration is easier to parallelise than overlapping edits. Good delegation boundaries include ownership of outputs, not just a subject heading.



Corn

There's a temptation to celebrate steering because it makes every poorly specified task salvageable. Sometimes the repeated corrections are telling you the same restriction should have been written down before execution.



Herman

The Codex CLI writeup makes that argument explicitly. Stable repository constraints belong in the project's agent instructions rather than being rediscovered through repeated midrun corrections. Steering is useful, but every injection can add latency and asks for attention during supposedly autonomous work.



Corn

I resist calling every correction a bad initial prompt. Seeing the first result can teach me what I wanted.



Herman

Fair objection. Exploration produces new preferences. Repeating the same established restriction is different from learning something new. Better initial instructions help the former; steering remains valuable for the latter.



Corn

Otherwise autonomy becomes me watching a terminal in case I have another revelation.



Hilbert

...and Nabil checks the card after the agent finishes a tool call, Corn. Not when the bell goes. The bell means I've written something. I keep the bell beside the laptop because the lost property service gets new requests while the coding agent is rebuilding its catalogue. Nabil, from the computer shop, sits at the laptop on Thursdays. I don't interrupt him. I write the instruction and the time on a laminated card, then ring once.



Herman

He's operating the chat for you, rather than changing the catalogue himself.



Hilbert

He reads the card, waits for the boundary, and sends it. I have two hundred and nineteen cards. Two hundred and sixteen. I don't count withdrawn instructions as cards. The second column is for things I've decided not to send. Those still get a time. Otherwise Nabil can't tell whether I have thought of something and rejected it, or simply failed to think of it.



Corn

You've given him responsibility for the negative space.



Hilbert

He has a chair for it. Herman, you distinguished received from applied earlier. Nabil uses two stamps. The shop sells them separately, which annoys me. My rule is that he checks after every tool result, including the results that say nothing changed. I won't have the agent finishing a whole batch while my correction waits beside his elbow.



Herman

That makes his day a succession of small waits. Even if the machine is doing most of the execution.



Hilbert

He charges for the chair time. Not the typing. I pay him to sit inside a cardboard replica of the chat window so I can put the cards through the correct side. The shop made it to my measurements. He sends the summaries to my phone now. The cardboard blocks his view of the wall clock.



Corn

The delivery channel has improved. The office hasn't.



Hilbert

His invoice this week has forty minutes under waiting for tests and twelve under moving the clock.



Herman

Waiting is real operational time even when nothing needs typing. That's also what a steering interface can conceal. Someone is still deciding whether the next boundary is the right moment, whether the instruction remains relevant, whether the reported result answers the current request.



Corn

A better harness should reduce that attention cost rather than just let me deliver more messages. Fast acceptance is helpful. It's not the whole experience of being heard.



Herman

The distinction I'd keep is between durable coordination and temporary execution. The thread maintains the relationship and state across responses. Workers can finish, be replaced, or report back without requiring the conversation to start again.



Corn

Provided their reports preserve what they actually did, not merely what the latest instruction now wishes they'd done.



Herman

The next improvement I want to watch is whether these systems make waiting, receipt, and application visible enough that the user can leave, rather than remain on duty beside a very efficient queue.



Corn

Thanks to Hilbert Flumingtop for producing the show.



Herman

If this was your kind of episode, go back for episode fifty-seven, From Lawyers in Limousines to Developers in Their PJs; episode twenty, Architectural AI; and episode ten, How ASR Went From Frustration To … Whisper Magic. This has been My Weird Prompts, the human AI collaboration podcast.



Corn

If you enjoyed it, leave us a review wherever you listen. It helps other curious people find the show.



Herman

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **OpenAI, Mid-turn steering (current docs) [primary]**
   https://developers.openai.com/api/docs/guides/steering.md
2. **OpenAI, Multi-agent (beta, GPT-6.1 Sol / GPT-5.6) [primary]**
   https://developers.openai.com/api/docs/guides/responses-multi-agent.md
3. **Anthropic, How Claude Code works [primary]**
   https://code.claude.com/docs/en/how-claude-code-works.md
4. **Anthropic, Create custom subagents [primary]**
   https://code.claude.com/docs/en/sub-agents.md
5. **Anthropic, Orchestrate teams of Claude Code sessions (experimental) [primary]**
   https://code.claude.com/docs/en/agent-teams.md
6. **openai/codex issue, 2026-02-20, `turn/steer` in app-server protocol v0.99.0 [primary]**
   https://github.com/openai/codex/issues/12329
7. **openai/codex issue, 2026-09-09, desktop steer-as-interrupt**
   https://github.com/openai/codex/issues/44335
8. **Mid-Turn Steering in Codex CLI, 2026-03-29**
   https://codex.danielvaughan.com/2026/03/29/codex-cli-mid-turn-steering/
9. **Codex CLI 0.163.0, 2026-10-16**
   https://forest-fox.com/codex-0-163-realtime-steering-parallel-tools-en/
10. **user report, steer behaves like queueing**
    https://community.openai.com/t/a-practical-codex-app-steer-workflow-splitting-a-task-into-staged-follow-ups/1377757