Messages and delivery

Command Queue

We serialize inbound auto-reply runs (all channels) through a tiny in-process queue to prevent multiple agent runs from colliding, while still allowing safe parallelism across sessions.

Why

Auto-reply runs can be expensive (LLM calls) and can collide when multiple inbound messages arrive close together.

Serializing avoids competing for shared resources (session files, logs, CLI stdin) and reduces the chance of upstream rate limits.

How it works

A lane-aware FIFO queue drains each lane with a configurable concurrency cap (default 1 for unconfigured lanes; main defaults to 4, subagent to 8).

`runEmbeddedPiAgent` enqueues by **session key** (lane `session:<key>`) to guarantee only one active run per session.

Each session run is then queued into a **global lane** (`main` by default) so overall parallelism is capped by `agents.defaults.maxConcurrent`.

When verbose logging is enabled, queued runs emit a short notice if they waited more than ~2s before starting.

Typing indicators still fire immediately on enqueue (when supported by the channel) so user experience is unchanged while we wait our turn.

›

Queue modes (per channel)

Inbound messages can steer the current run, wait for a followup turn, or do both:

- `steer` : inject immediately into the current run (cancels pending tool calls after the next tool boundary). If not streaming, falls back to followup.
- `followup` : enqueue for the next agent turn after the current run ends.
- `collect` : coalesce all queued messages into a **single** followup turn (default). If messages target different channels/threads, they drain individually to preserve routing.
- `steer-backlog` (aka `steer+backlog`): steer now **and** preserve the message for a followup turn.
- `interrupt` (legacy): abort the active run for that session, then run the newest message.
- `queue` (legacy alias): same as `steer` .

Steer-backlog means you can get a followup response after the steered run, so streaming surfaces can look like duplicates. Prefer `collect` / `steer` if you want one response per inbound message. Send `/queue collect` as a standalone command (per-session) or set `messages.queue.byChannel.discord: "collect"` .

Defaults (when unset in config):

- All surfaces → `collect`

Configure globally or per channel via `messages.queue` :

```
{
  messages: {
    queue: {
      mode: "collect",
      debounceMs: 1000,
      cap: 20,
      drop: "summarize",
      byChannel: { discord: "collect" },
    },
  },
}
```

Queue options

Options apply to `followup`, `collect`, and `steer-backlog` (and to `steer` when it falls back to followup):

- `debounceMs`: wait for quiet before starting a followup turn (prevents “continue, continue”).
- `cap`: max queued messages per session.
- `drop`: overflow policy (`old`, `new`, `summarize`).

Summarize keeps a short bullet list of dropped messages and injects it as a synthetic followup prompt. Defaults: `debounceMs: 1000`, `cap: 20`, `drop: summarize`.

Per-session overrides

- Send `/queue <mode>` as a standalone command to store the mode for the current session.
- Options can be combined: `/queue collect debounce:2s cap:25 drop:summarize`
- `/queue default` or `/queue reset` clears the session override.

Scope and guarantees

- Applies to auto-reply agent runs across all inbound channels that use the gateway reply pipeline (WhatsApp web, Telegram, Slack, Discord, Signal, iMessage, webchat, etc.).
- Default lane (`main`) is process-wide for inbound + main heartbeats; set `agents.defaults.maxConcurrent` to allow multiple sessions in parallel.
- Additional lanes may exist (e.g. `cron` , `subagent`) so background jobs can run in parallel without blocking inbound replies.
- Per-session lanes guarantee that only one agent run touches a given session at a time.
- No external dependencies or background worker threads; pure TypeScript + promises.

Troubleshooting

- If commands seem stuck, enable verbose logs and look for “queued for …ms” lines to confirm the queue is draining.
- If you need queue depth, enable verbose logs and watch for queue timing lines.

‹ Retry Policy