Messages and delivery

Streaming and Chunking

OpenClaw has two separate “streaming” layers:

- **Block streaming (channels):** emit completed **blocks** as the assistant writes. These are normal channel messages (not token deltas).
- **Token-ish streaming (Telegram only):** update a temporary **preview message** with partial text while generating.

There is **no true token-delta streaming** to channel messages today. Telegram preview streaming is the only partial-stream surface.

Block streaming (channel messages)

Block streaming sends assistant output in coarse chunks as it becomes available.

```
Model output
  └─ text_delta/events
       ├─ (blockStreamingBreak=text_end)
       │    └─ chunker emits blocks as buffer grows
       └─ (blockStreamingBreak=message_end)
            └─ chunker flushes at message_end
                 └─ channel send (block replies)
```

Legend:

- `text_delta/events` : model stream events (may be sparse for non-streaming models).
- `chunker` : `EmbeddedBlockChunker` applying min/max bounds + break preference.
- `channel send` : actual outbound messages (block replies).

Controls:

- `agents.defaults.blockStreamingDefault` : `"on"` / `"off"` (default off).
- Channel overrides: `*.blockStreaming` (and per-account variants) to force `"on"` / `"off"` per channel.
- `agents.defaults.blockStreamingBreak` : `"text_end"` or `"message_end"` .
- `agents.defaults.blockStreamingChunk` : `{ minChars, maxChars, breakPreference? }` .
- `agents.defaults.blockStreamingCoalesce` : `{ minChars?, maxChars?, idleMs? }` (merge streamed blocks before send).
- Channel hard cap: `*.textChunkLimit` (e.g., `channels.whatsapp.textChunkLimit`).
- Channel chunk mode: `*.chunkMode` (`length` default, `newline` splits on blank lines (paragraph boundaries) before length chunking).
- Discord soft cap: `channels.discord.maxLinesPerMessage` (default 17) splits tall replies to avoid UI clipping.

Boundary semantics:

- `text_end` : stream blocks as soon as chunker emits; flush on each `text_end` .
- `message_end` : wait until assistant message finishes, then flush buffered output.

`message_end` still uses the chunker if the buffered text exceeds `maxChars` , so it can emit multiple chunks at the end.

Chunking algorithm (low/high bounds)

Block chunking is implemented by `EmbeddedBlockChunker`:

- **Low bound:** don't emit until buffer ≥ `minChars` (unless forced).
- **High bound:** prefer splits before `maxChars`; if forced, split at `maxChars`.
- **Break preference:** `paragraph` → `newline` → `sentence` → `whitespace` → hard break.
- **Code fences:** never split inside fences; when forced at `maxChars`, close + reopen the fence to keep Markdown valid.

`maxChars` is clamped to the channel `textChunkLimit`, so you can't exceed per-channel caps.

Coalescing (merge streamed blocks)

When block streaming is enabled, OpenClaw can **merge consecutive block chunks** before sending them out. This reduces "single-line spam" while still providing progressive output.

- Coalescing waits for **idle gaps** (`idleMs`) before flushing.
- Buffers are capped by `maxChars` and will flush if they exceed it.
- `minChars` prevents tiny fragments from sending until enough text accumulates (final flush always sends remaining text).
- Joiner is derived from `blockStreamingChunk.breakPreference` (`paragraph` → `\n\n`, `newline` → `\n`, `sentence` → space).
- Channel overrides are available via `*.blockStreamingCoalesce` (including per-account configs).
- Default coalesce `minChars` is bumped to 1500 for Signal/Slack/Discord unless overridden.

Human-like pacing between blocks

When block streaming is enabled, you can add a **randomized pause** between block replies (after the first block). This makes multi-bubble responses feel more natural.

- Config: `agents.defaults.humanDelay` (override per agent via `agents.list[].humanDelay`).
- Modes: `off` (default), `natural` (800–2500ms), `custom` (`minMs` / `maxMs`).
- Applies only to **block replies**, not final replies or tool summaries.

“Stream chunks or everything”

This maps to:

- **Stream chunks:** `blockStreamingDefault: "on"` + `blockStreamingBreak: "text_end"` (emit as you go). Non-Telegram channels also need `*.blockStreaming: true`.
- **Stream everything at end:** `blockStreamingBreak: "message_end"` (flush once, possibly multiple chunks if very long).
- **No block streaming:** `blockStreamingDefault: "off"` (only final reply).

Channel note: For non-Telegram channels, block streaming is **off unless** `*.blockStreaming` is explicitly set to `true`. Telegram can stream a live preview (`channels.telegram.streamMode`) without block replies.

Config location reminder: the `blockStreaming*` defaults live under `agents.defaults`, not the root config.

Telegram preview streaming (token-ish)

Telegram is the only channel with live preview streaming:

Uses Bot API `sendMessage` (first update) + `editMessageText` (subsequent updates).

`channels.telegram.streamMode: "partial" | "block" | "off"` .

- `partial` : preview updates with latest stream text.
- `block` : preview updates in chunked blocks (same chunker rules).
- `off` : no preview streaming.

Preview chunk config (only for `streamMode: "block"`): `channels.telegram.draftChunk` (defaults: `minChars: 200` , `maxChars: 800`).

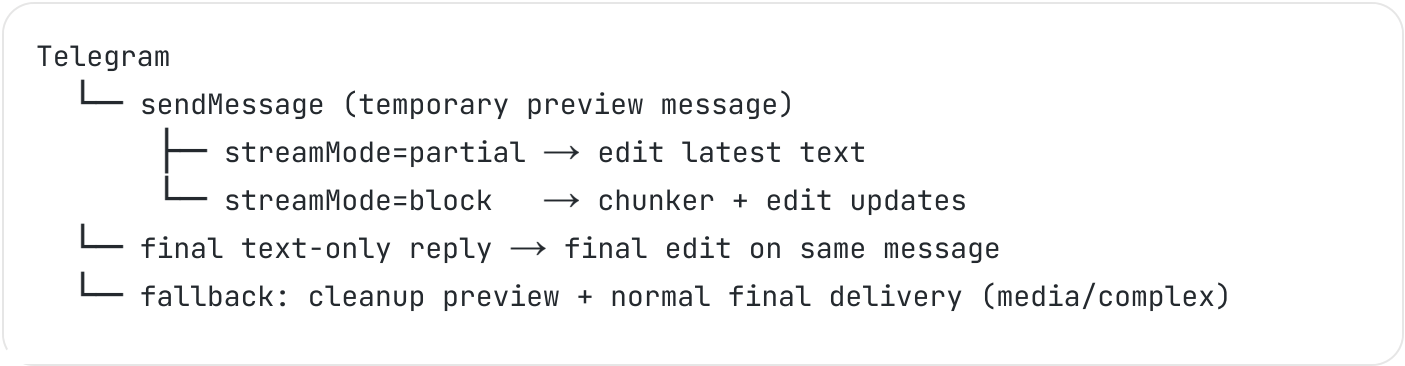
Preview streaming is separate from block streaming.

When Telegram block streaming is explicitly enabled, preview streaming is skipped to avoid double-streaming.

Text-only finals are applied by editing the preview message in place.

Non-text/complex finals fall back to normal final message delivery.

`/reasoning stream` writes reasoning into the live preview (Telegram only).



Legend:

- `preview message` : temporary Telegram message updated during generation.
- `final edit` : in-place edit on the same preview message (text-only).

‹ Messages Retry Policy ›