Part 1 — How Ringmaster works

← Contents · Next: Part 2A — Setup

Read this page before the endpoints. Ringmaster is the **engine** of an intelligent call centre; your service is a **client app** built on it. This page explains the engine's objects, how a call runs, and what the client app must do at each step.

1.1 Who's who

Who	What they are	Example
Ringmaster owner	Runs the engine; creates client apps, buys phone numbers, gives numbers to client apps (`ringmaster-admin`, see Appendix)	QBIT SPARK ops
Client app	A product that uses the engine through this API. Has one API key, a webhook URL and a set of features	a messaging or business platform
Account	One business inside a client app. Everything (numbers, flows, calls, agents) belongs to an account	PMK Chemicals, Kliniki ya Afya Bora
Number	A phone number the account answers on and calls from	`+255677000010`
Call flow (script)	What a caller hears and can do, as wired blocks. Stored in numbered, unchangeable **versions**	`orders` v3
Agent	A person who answers calls in the browser (or on a mobile). Ringmaster keeps only a ref and a name	`asha`
Department	A group of agents a flow can send callers to	Mapokezi, Uhasibu
Caller / person called	Anyone on a phone. Ringmaster keeps only the number and what happened on the call	`+255745051250`

```
Ringmaster owner
  └── Client app  (API key, webhook URL, features)
       ├── Account: PMK Chemicals  (hours, quiet hours, recording, limits)
```

```
│   ├── Numbers — each plays one pinned flow version
│   ├── Call flows — versions v1, v2, v3 …
│   ├── Departments — agents
│   ├── Media (audio files), knowledge base (for the AI), opt-outs
│   └── Calls, call-outs, reports
└── Account: Kliniki ya Afya Bora …
```

A client app maps its own objects to these: one of its businesses = an account (`externalRef` = its business id), one of its users who answers calls = an agent (`ref` = its user id).

1.2 Getting access

The Ringmaster owner creates the client app once and gives it:

Item	What it is	Where it's used
API key	`vk_live_…`, shown once	Every request: `Authorization: Bearer <key>`
Webhook URL	The client app's HTTPS endpoint, e.g. `https://app.example.co.tz/hooks/ringmaster`	The engine POSTs events there (Part 3)
Webhook secret	Shown once; signs every webhook	The client app checks each event's signature
Features	What the app may do	See below
Rate limit	Requests per second for the app (fixed one-second window)	Over it → `429 RATE_LIMITED`

Feature	Allows
`INBOUND`	Answering incoming calls. Without it every incoming call is refused (`REJECTED`, `FEATURE_OFF`)
`CALLOUTS`	Calling people: quick calls and call-outs (`403 FEATURE_OFF` without it); callers booking a callback (refused without it)
`OTP`	Voice OTP calls (`POST /v1/otp`); without it → `403 FEATURE_OFF`
`AI`	AI Agent blocks and "smart" (AI) speech matching in Menu, Poll and Pick; a flow using them is refused (`403 FEATURE_OFF`)
`NUMBERS`	Adding and removing numbers through the API; without it → `403 FEATURE_OFF`

Features are checked when something is made **and again when it runs**: if the owner takes a feature away later, running call-outs pause (`FEATURE_OFF`), AI Agent blocks take their `err` exit, and

smart speech matching falls back to the options' own words. Agent click-to-call (`POST /v1/calls/agent`) needs no feature.

A client app only ever sees its own accounts, numbers and calls. Another app's id gives `404 NOT_FOUND`.

1.3 Setting up a business, in order

```
1. POST  /v1/accounts                          {externalRef: "pmk-chemicals"}       →
accountId
2. PATCH /v1/accounts/{accountId}              {businessName, hours, quietHours, …}
3. GET   /v1/numbers?status=ALLOCATED           free numbers the owner gave the client app
   POST  /v1/numbers/{numberId}/assign          {accountId, label}
4. PUT   /v1/accounts/{accountId}/scripts/orders            {name, direction: "INCOMING"}
   POST  /v1/accounts/{accountId}/scripts/orders/versions   {version: 1, definition, runnable:
true}  → versionId
5. PUT   /v1/numbers/{numberId}/script          {scriptVersionId}
```

From step 5 on, calls to the number play the flow. Optional: departments and agents (to connect callers to people), media (recorded greetings, music), knowledge base (for AI agents), App actions (SMS, payments done by the client app).

1.4 An incoming call, step by step

```
Caller dials +255677000010
   │  phone network → SIP trunk → LiveKit room → voice worker joins
   ▼
Engine: which account and which flow version is pinned on the number?   (none → call refused,
NO_SCRIPT)
   │  checks: app active, account active, INBOUND on, caller under the per-caller limit
   ▼
call.started webhook ──────────────────────────────────────────────► client app
   │
   ▼  blocks run one after another, starting at the flow's start block
   say "Karibu PMK Chemicals…" → menu "Bonyeza 1 kuagiza…" → (caller presses 1 or says "oda")
   │      every step is written to the call's timeline (BLOCK_ENTER, SAY, KEYS, SPEECH, …)
   │      call.block webhooks for the steps clients care about (menu choice, App action, …)
```

```
    ▼
    … collect → pick → confirm → goodbye
    ▼
Call ends ─► call.ended webhook (full summary: result, durations, keys, outcomes, cost)
         └─► recording.ready / summary.ready later, if anything was recorded
```

What the caller hears is played **in order, block after block**: a block's sound finishes before the next block's sound starts. Blocks that make something happen for a *person* wait for that too: a Connect block rings agents only after the caller has heard everything before the waiting music.

Statuses of a call (`status`): `QUEUED` → `DIALING` → `RINGING` (outgoing only) → `IN_SCRIPT` → `WAITING` (waiting room) → `CONNECTING` → `LIVE` (with an agent) → `ENDED`. After it ends, `result` says how it went (`HANDLED_BY_SCRIPT`, `HANDLED_BY_AGENT`, `HANDLED_BY_AI`, `VOICEMAIL`, `MISSED`, `NO_ANSWER`, `BUSY`, `DECLINED`…), `endReason` why it ended and `endedBy` who ended it. All values: Part 4.

Guards that protect every account (account `limits`, changeable with `PATCH /v1/accounts/{id}`):

Limit	Default	What happens
`maxCallMinutes`	15	The call is ended (`MAX_CALL_LENGTH`)
`maxWaitSeconds`	300	Longest wait in a waiting room, whatever the flow says
`callerCallsPer10Min`	5	The same caller calling more often is refused (`REJECTED`, `CALLER_LIMIT`)
`otpPerDestinationPer10Min`	3	More OTP calls to one number → refused
`voicemailMaxSeconds`	120	Longest voicemail
`maxBlockVisits`	25	Blocks run without any input from the caller before the call is ended (`LOOP_GUARD`) — stops endless loops

---

1.5 Calling people (call-outs)

The engine places calls for an account and plays an **Outgoing** flow when the person answers.

Way	Endpoint	Use
Quick call	`POST /v1/calls`	"Your order is ready" to 1–1000 people, now. Created and started in one request

Way	Endpoint	Use
Broadcast	`POST /v1/callouts` `kind: BROADCAST`	The same, but built in steps (create → add people → start), optionally at `startsAt`
Campaign	`POST /v1/callouts` `kind: CAMPAIGN`	Calling a list over days: a daily window (`windowFrom`/`windowTo`), week days, pace per minute, up to 5 tries, a fallback App action (e.g. SMS) after the last failed try
Scheduled callback	made by a caller during a call (Schedule call block or AI `schedule_call`)	"Nipigie baada ya saa moja": a one-person call-out with ref `schedule:<flow>:<number>`; the client app sees it in `GET /v1/callouts` and the `call.scheduled` webhook
OTP	`POST /v1/otp`	Read a code to a number

Each person on a call-out is a **recipient** with their own `vars` (e.g. `{"name": "Asha", "amount": "45000"}`) that the flow says as `{name}`, `{amount}`.

Rules the engine applies to every call-out (the client app doesn't need to):

- **Quiet hours** (account `quietHours`, default 21:00–07:00): nobody is called inside them; calls wait for the end.
- **Opt-outs**: a number on the account's opt-out list is skipped (`SKIPPED`), checked again just before dialling.
- **Retries**: busy, no answer, unreachable or failed → tried again after `retryGapSec` (≥ 300 s) up to `maxTries` (quick call ≤ 3, campaign ≤ 5). **Declined** (the person rejected the call) is never retried.
- **Pace**: campaigns dial at most `pacePerMin`.

Call-out statuses: `DRAFT` → `SCHEDULED` → `RUNNING` ⇄ `PAUSED` → `COMPLETED` / `STOPPED` / `CANCELLED`.
Recipient: `PENDING` → `DIALING` → `DONE` (with a `result`) or `SKIPPED`.

1.6 A call flow's life

```
PUT  …/scripts/{ref}                    create the flow (name, INCOMING or OUTGOING);
direction never changes
POST …/scripts/{ref}/versions           push version 1, 2, 3 … — each version is frozen once
stored
         │  the engine checks it (Part 5 §5.9): errors → 422 with the list, nothing stored
         │                                      warnings → stored, warnings returned
         ▼
```

```
      runnable?  ── no (you pushed runnable:false: it has risky blocks and waits for approval)
         │              POST …/versions/{n}/runnable  after the client app's approval step
         ▼ yes
PUT  /v1/numbers/{id}/script           put the version on a number (Incoming)     — or —
POST /v1/calls · /v1/callouts          use it for calls out (Outgoing)
POST …/versions/{n}/revoke             stop a version for good: a number still playing it
refuses calls,
                                       call-outs using it pause (VERSION_REVOKED) — move them
first
DELETE …/scripts/{ref}                 delete the flow (refused while anything uses it;
archived if it ever ran)
```

- **Versions never change.** To change a flow, push the next version and move the number to it. Calls keep pointing at the exact version they ran, so reports stay true.
- **Pushing the same version number with the same document again** is safe: `200 "Version already stored"`.
- **Approval is the client app's policy.** Some blocks are risky: `collect` (callers' digits), `request` (calls a URL), `ai_agent`, and App actions the client app registered as risky (e.g. pay). The engine lists them in `riskyBlocks` when a version is checked or stored, and trusts the `runnable` flag it is given: push such a version with `runnable: false`, let the business's reviewer approve it in the client app, then `POST …/versions/{n}/runnable`. A not-runnable version can't be put on a number. A call-out may already use it: it is saved and waits (`waitingApproval: true`), then starts by itself once the version is approved.
- `testOnly` **versions** are for test calls only and never become runnable.
- **Reading a flow back**: `GET …/scripts/{ref}/versions/{n}` returns the full document. Secrets stored in Send request blocks come back as `"(hidden)"`.
- **Layout** (block positions on the client app's canvas) belongs to the client app: the engine drops a `layout` key on push.
- **Texts are turned into speech when the version is stored**, so the first caller doesn't wait.

1.7 Agents in the browser

```
Client app (once): POST departments {name, strategy, ringTimeoutSec…}  ·  PUT agents/{ref}
{displayName}
                PUT departments/{id}/members {agentRefs: [...]}      (list order = round-
robin order)

Agent's browser:
  "Start shift"  → client app backend → POST agents/{ref}/lobby-token → {url, room, token}
```

```
                → browser joins the lobby room (LiveKit JS SDK, data only, no mic)
                → client app backend → PUT agents/{ref}/status {AVAILABLE}
  A caller reaches a Connect block → hears "tunakuunganisha…", then music
                → engine sends {type:"ring", callId, from, department} into the lobby
  "Accept"      → client app backend → POST /v1/calls/{callId}/accept {agentRef} → {url,
room, token}
                → browser joins the call room with the microphone on → they talk
  "Hold" / "Transfer" / "Hang up" → POST /v1/calls/{callId}/hold · /transfer · /hangup
{agentRef}
  Call ends     → agent is WRAP_UP for the department's wrapUpSec, then AVAILABLE (or POST
…/ready)
```

Agent statuses: `OFFLINE` → `AVAILABLE` → `RINGING` → `BUSY` → `WRAP_UP` → `AVAILABLE`; `AWAY` = on a break (online, no calls). One call at a time per agent. Missing `maxMissedBeforeAway` rings in a row (default 5) moves the agent to `AWAY`. Every change is sent as an `agent.status` webhook (for a wallboard).

Ring modes (per number, `PUT /v1/numbers/{id}/ring-mode`): `BROWSER` (agents' browsers), `MOBILE` (agents' mobiles), `BROWSER_AND_MOBILE`.

The browser needs HTTPS (or `localhost`) for the microphone, and the "Start shift" click is what allows the ring sound. Details of every signal: Part 3.

1.8 Webhooks

Every event is a signed `POST` to the client app's webhook URL:

```
POST https://app.example.co.tz/hooks/ringmaster
Content-Type: application/json
X-Ringmaster-Event: call.ended
X-Ringmaster-Delivery: 5b0c7f8e-…            (same as the body's id; use it to drop repeats)
X-Ringmaster-Timestamp: 1794588244
X-Ringmaster-Signature: 9f2c…                 hex HMAC-SHA256(secret, "<timestamp>.<raw
body>")

{ "id": "5b0c7f8e-…", "type": "call.ended", "createdAt": "2026-11-13T16:44:04Z", "data": { … }
}
```

- Answer **2xx quickly** (do the work after). Anything else is retried after 1 min, 5 min, 30 min, 2 h, 6 h, 12 h, then every 12 h until 24 h have passed; then the event is marked

failed.
- Events are written in the same database step as the change they describe, so none is lost; a repeat can arrive — drop it by `id`.
- Check the signature on the **raw body** before parsing, and refuse timestamps more than 5 minutes old.

All events and their data: Part 3.

1.9 App actions (the client app does something during a call)

Ringmaster never sends SMS, takes payments or tags contacts itself. The client app **registers** what it can do; a flow's **App action** block asks for it; the client app does it and answers.

```
PUT /v1/actions  [ {name: "pay", label: "Pay now", params: {...}, wait: true, outcomes:
["paid","failed"], timeoutSec: 120}, … ]

Caller in a flow reaches  "pay_1": {type: "app_action", action: "pay", params: {amount:
"{amount}"}, x: {paid: …, failed: …, timeout: …}}
   engine ─► webhook action.requested {actionId, name: "pay", callId, params: {amount:
"45000", phone: "+2557…"}, wait: true, timeoutAt}
   caller waits (music, "Thibitisha malipo kwenye simu yako")
   client app pushes M-Pesa … PIN entered …
   client app ─► POST /v1/calls/{callId}/action-result {actionId, outcome: "paid", data:
{receipt: "QK7…"}}
   engine: the flow goes on through the "paid" exit; {action.pay_1.receipt} can be said
```

`wait: false` actions (send SMS, tag a contact) are fired and the call goes on at once.

1.10 Recordings, transcripts, summaries

Recorded	When
Voicemail	Always, when a flow reaches a Voicemail block (up to `voicemailMaxSeconds`)

Recorded	When
Live parts	Talk with an agent, when the account's `recording.mode` is `LIVE_PARTS` (default) and the department / block allows it. Callers hear the `noticeText` first
Callers' spoken answers	Short clips used for speech recognition; not kept as recordings

After a recording: `recording.ready` webhook → `GET /v1/calls/{id}/recording` (MP3). If `recording.transcribe` is on (default), a transcript and a short summary follow: `summary.ready` → `GET /v1/calls/{id}/transcript`. Recordings are encrypted at rest and deleted after the account's `retentionDays` (default 90).

1.11 Costs

Every call carries `providerCost`: what it cost Ringmaster in phone minutes and AI, line by line.

```
"providerCost": { "tzs": 68.61, "usd": 0.000334, "items": [
  { "provider": "AT", "tzs": 68.61, "lines": [
      { "item": "AT_OUTBOUND", "qty": 1, "unit": "MINUTE", "rate": 60, "currency": "TZS",
"amount": 60, "source": "RATE" },
      { "item": "AT_SIP_TRUNK", "qty": 1, "unit": "MINUTE", "rate": 8.61, "currency": "TZS",
"amount": 8.61, "source": "RATE" } ] },
  { "provider": "OPENAI", "usd": 0.000334, "lines": [
      { "item": "OPENAI_TRANSCRIBE", "qty": 0.0742, "unit": "MINUTE", "rate": 0.0045,
"currency": "USD", "amount": 0.0003339, "source": "MEASURED", "model": "gpt-transcribe" } ] }
] }
```

The client app decides what to charge its customers; these are inputs, not prices.

1.12 Formats

What	Format
Phone numbers	E.164 with `+`: `+255745051250`
IDs	UUID strings; your own ids go in `externalRef` (accounts), `ref` (scripts, agents)
`ref` / `externalRef`	Your text id, up to 120 characters; letters, digits, `-`, `_` recommended
Times in responses	UTC ISO 8601 (`2026-11-13T14:41:00.802627Z`)

What	Format
Times inside an account	`HH:mm` in the account's `timezone` (opening hours, quiet hours, campaign windows)
Languages	`sw` (Kiswahili, default), `en`
Lists	`GET /v1/calls` pages with `page` (from 0) and `size` (default 50) → `{calls, page, size, total}`; other lists return everything
Money in costs	TZS with 2 decimals, USD with up to 7

Revision #2
Created 13 November 2026 18:45:13 by Admin Qbit
Updated 13 November 2026 18:59:30 by Admin Qbit