



RIOT 5.0 by Magochi

COMMUNICATIONS SUITE — COMPLETE API DOCUMENTATION

REST API • Email • SMS • CRM • Custom Fields • Analytics • Webhooks • Remote Agents

Prepared by Kelvin Magochi

mwangikelvin278@gmail.com | admin@webninjaafrica.com | +254718265708

Version 0.4.0 Enterprise Development Candidate • Generated from the release source documentation

Part I — Complete API Reference

Communications Suite — Complete API Reference (v0.4.0 Enterprise Development)

This document is intended to be shipped with the application and kept in step with the source. It covers every currently implemented HTTP API route (`/api/v1/*` and `/agent/v1/*`) plus provider webhook callbacks and the health check. UI routes are not presented as public APIs. This is a development candidate: verify against the exact deployed build before relying on it for production integrations.

Common conventions

- Base URL: `https://your-host.example`.
- API version prefix: `/api/v1`.
- Tenant API authentication: `Authorization: Bearer $COMMS_API_TOKEN`.
- Remote agent authentication: `Authorization: Bearer $AGENT_TOKEN` (different token from a tenant API token).
- JSON requests use `Content-Type: application/json`; contact CSV import also accepts `multipart/form-data` with a `file` field.
- Tenant API tokens are tenant-scoped and require the endpoint's scope. A missing, inactive, wrong-tenant, or insufficient-scope token generally returns `401 {"error":"unauthorized_or_insufficient_scope"}`.
- Limits are bounded by the server. A `202` or `queued` response means accepted for asynchronous processing, not proof of delivery.
- Use HTTPS, keep credentials server-side, apply idempotency keys to sends, and deduplicate webhook event IDs.

Scope catalogue

Common scopes are `contacts:read`, `contacts:write`, `messages:send`, `messages:read`, `inbox:read`, `engagement:read`, `groups:read`, `groups:write`, `templates:read`, `templates:write`, `reminders:read`, `reminders:write`, `notes:read`, `notes:write`, `crm:read`, `crm:write`, `notifications:read`, `webhooks:read`, and `webhooks:write`. Grant only the scopes an integration needs. Create tokens through the CLI, then store the one-time displayed token in a secret manager:

```
flask --app app create-api-token TENANT_ID "Integration name" "contacts:read,messages:send,messages:read,webhooks:read,webhooks:write"
```

1. Health

GET /health

Unauthenticated health/status response for load balancers and deployment checks. It is not a substitute for checking database, provider, and worker readiness individually.

```
curl -sS https://your-host.example/health
```

2. Contacts

GET /api/v1/contacts — *scope* `contacts:read`

Query parameters: `q` (searches email, phone, names, organization, group and interests), `group`, `interest`, `limit` (default 100, maximum 500).

```
curl -G https://your-host.example/api/v1/contacts \
-H "Authorization: Bearer $COMMS_API_TOKEN" \
--data-urlencode 'q=Jane' --data-urlencode 'group=Renewals' --data-urlencode 'limit=100'
```

Returns `data`, `count`, and the applied filters. Each contact includes `id`, `email`, `phone`, `first_name`, `last_name`, `organization`, `group_name`, `interests`, and `consent_status`.

POST /api/v1/contacts — *scope* `contacts:write`

Requires an email or phone. Email duplicates within the tenant return `409 duplicate_contact`.

```
{ "email": "jane@example.org", "phone": "+254700000000", "first_name": "Jane", "last_name": "Kamau",
  "organization": "Example Ltd", "group_name": "Renewals", "interests": "insurance, renewal", "consent_status":
  "opted_in" }
```

Returns **201** `{ "id": 123 }`. Allowed consent values in normal use are **unknown**, **opted_in**, and **opted_out**; use **opted_in** only when you have a valid consent record.

POST /api/v1/contacts/import — **scope** contacts:write

JSON batch example (maximum 10,000 records per request):

```
{ "group_name": "Renewals", "interests": "renewal", "contacts": [ { "email": "jane@example.org", "first_name":
  "Jane", "consent_status": "opted_in" }, { "phone": "+254711111111", "first_name": "Peter", "consent_status":
  "unknown" } ] }
```

Alternatively send a UTF-8/UTF-8-BOM CSV as multipart field **file**, with optional **group_name** and **interests** form fields. Common CSV columns: **email**, **phone**, **first_name**, **last_name**, **organization**, **group_name**, **interests**, **consent_status**. Tenant-defined CRM fields may be imported by using headers in the form **custom.FIELD_KEY** (for example **custom.policy_number**). Create the field first with **POST /api/v1/contact-fields**. Unknown/inactive custom field keys or invalid values cause the batch to fail rather than silently discard those values. Response reports **imported**, **duplicates**, **skipped**, **custom_values_imported**, **total**, **group_name**, and **interests**. This endpoint reads the submitted CSV into memory and is **not** the planned resumable, 500-million-row import pipeline.

```
curl -X POST https://your-host.example/api/v1/contacts/import \
  -H "Authorization: Bearer $COMMS_API_TOKEN" \
  -F file=@contacts.csv -F group_name=Renewals
```

Example CSV with custom CRM fields:

```
email,first_name,consent_status,custom.policy_number,custom.customer_tier
jane@example.org,Jane,opted_in,PL-2048,Gold
peter@example.org,Peter,unknown,PL-2049,Standard
```

The equivalent JSON row may include **"custom.policy_number": "PL-2048"**. Custom fields are tenant-specific and must already exist. Blank custom values are not stored; custom values are validated against their configured type/options.

3. CRM custom fields

These fields are tenant-defined; they do not change existing Contact columns. Scopes: reads use **crm:read**; writes use **crm:write**.

GET /api/v1/contact-fields

Lists the tenant's field definitions.

POST /api/v1/contact-fields

```
{ "key": "policy_number", "label": "Policy number", "type": "text", "required": false }
```

For a select field, provide options:

```
{ "key": "customer_tier", "label": "Customer tier", "type": "select", "options": [ "Standard", "Gold", "Platinum" ],
  "required": false }
```

Types: **text**, **textarea**, **email**, **phone**, **url**, **number**, **boolean**, **date**, **select**. Keys are normalized to lowercase letters/numbers/underscores and must start with a letter. Duplicate keys return **409**.

PATCH /api/v1/contact-fields/{field_id} **and** DELETE /api/v1/contact-fields/{field_id}

PATCH accepts supported definition changes such as label, required, active, and select options. DELETE soft-disables the definition so saved values remain stored.

```
curl -X PATCH https://your-host.example/api/v1/contact-fields/4 \
  -H "Authorization: Bearer $COMMS_API_TOKEN" -H 'Content-Type: application/json' \
  -d '{ "label": "Policy / account number", "required": true }'
```

GET /api/v1/contacts/{contact_id}/fields

Returns active definitions and values for a contact owned by the token's tenant.

PUT **or** PATCH /api/v1/contacts/{contact_id}/fields

```
{ "values": { "policy_number": "POL-1001", "customer_tier": "Gold", "renewal_date": "2026-12-31", "is_vip": true } }
```

Personalize queued campaign messages with custom fields

After setting values for a contact, campaign/message bodies can use `{{custom.FIELD_KEY}}`. For example, if `policy_number` and `customer_tier` are defined for the tenant and values are saved for the recipient:

```
Hello {{first_name}},
Your policy {{custom.policy_number}} is due for renewal.
Your customer tier is {{custom.customer_tier}}.
```

The queue worker resolves active custom fields for that contact at send time. Built-in tokens such as `{{first_name}}`, `{{last_name}}`, `{{email}}`, `{{phone}}`, `{{organization}}`, and `{{unsubscribe_url}}` remain supported. A missing custom value renders as an empty string. Custom fields are tenant-scoped; do not put secrets or sensitive data into message templates unless disclosure is appropriate.

Values are validated against field types/options. Custom fields can also be used in API-created dynamic groups with the `custom:FIELD_KEY` rule syntax described below. Custom-field report exports are not yet implemented.

Dynamic groups using custom CRM fields

`POST /api/v1/groups` also accepts a custom field in the rule using `custom:FIELD_KEY`. The field must already exist, be active, and belong to the token tenant. Supported operators are `equals`, `contains`, and `not_empty`. Custom-field dynamic rules are evaluated by campaign scheduling, reminder recipient selection, and API reminder/group selection.

```
{
  "name":"Gold policyholders",
  "description":"Contacts whose custom customer tier is Gold",
  "is_dynamic":true,
  "rule":{"field":"custom:customer_tier","operator":"equals","value":"Gold"}
}
```

The field value is tenant-scoped. A missing or inactive field matches no contacts. The `contains` operator is a simple substring match and is not a full-text search.

4. Send messages

`POST /api/v1/messages/email` and `POST /api/v1/messages/sms` — **scope** `messages:send`

Email example:

```
curl -X POST https://your-host.example/api/v1/messages/email \
-H "Authorization: Bearer $COMMS_API_TOKEN" \
-H 'Content-Type: application/json' -H 'Idempotency-Key: invoice-4821-email' \
-d '{"recipient":"jane@example.org","subject":"Invoice 4821","body":"Your invoice is ready.","is_html":false}'
```

SMS example:

```
curl -X POST https://your-host.example/api/v1/messages/sms \
-H "Authorization: Bearer $COMMS_API_TOKEN" \
-H 'Content-Type: application/json' -H 'Idempotency-Key: appointment-889' \
-d '{"recipient":"+254700000000","body":"Your appointment is tomorrow."}'
```

Payload fields: `recipient` (required), `body` (required), and for email `subject` and optional `is_html`. The handler supports only the `email` and `sms` channel path values. HTML email is sanitized. The recipient must already exist as a tenant contact with `consent_status=opted_in`, not be suppressed, and be under the configured contact frequency limit. The optional `provider_id` can select a provider; otherwise the tenant default is used. The current direct-send route does not accept arbitrary attachments, a `template_id`, or a caller-supplied callback URL; use templates/campaign UI or tenant webhook subscriptions for those functions.

Response: `202 {"id":456,"status":"queued"}`; repeated `Idempotency-Key` values for the same token resolve to the existing message where supported. Without the header, a random key is generated, so clients should always provide one.

`GET /api/v1/messages/{message_id}` — **scope** `messages:read`

Returns tenant-owned message `id`, `channel`, `recipient`, `status`, `attempts`, `last_error`, and `created_at`. Other tenants' IDs return `404`.

```
curl https://your-host.example/api/v1/messages/456 -H "Authorization: Bearer $COMMS_API_TOKEN"
```

Statuses such as `queued`, `processing`, `accepted`, and `sent` are not interchangeable with provider-confirmed `delivered`. A worker crash during uncertain provider submission may result in `unknown` to avoid duplicate sending.

5. Inbox and engagement

GET /api/v1/inbox — *scope* inbox:read

Query parameters: **channel**=email|sms, **status**=unread|read, **address** (sender or recipient exact match), **q** (searches sender, recipient, subject, body), and **limit** (default 100, maximum 500).

```
curl -G https://your-host.example/api/v1/inbox \
  -H "Authorization: Bearer $COMMS_API_TOKEN" \
  --data-urlencode 'channel=email' --data-urlencode 'status=unread' --data-urlencode 'limit=50'
```

Returns inbox records with **id**, **channel**, **sender**, **recipient**, **subject**, **body**, **is_read**, **received_at**, **source_agent**, **source_address**, and **destination_address**. This endpoint is read-only; use the UI for read/star/reply actions unless a separate route is documented in the deployed version.

GET /api/v1/engagement — *scope* engagement:read

Query parameters: **address**, **type**, **days** (1–365, default 30), and **limit** (default 200, maximum 500).

```
curl -G https://your-host.example/api/v1/engagement \
  -H "Authorization: Bearer $COMMS_API_TOKEN" \
  --data-urlencode 'address=jane@example.org' --data-urlencode 'days=30'
```

Returns engagement events matching the selected filters.

6. Groups

GET /api/v1/groups — *scope* groups:read

Lists tenant groups and includes **id**, **name**, **description**, **is_dynamic**, and **rule**.

POST /api/v1/groups — *scope* groups:write

Static example:

```
{"name":"Renewal customers","description":"Policies due for renewal","is_dynamic":false}
```

Dynamic group example:

```
{"name":"Opted-in renewals","description":"Contacts eligible for renewal notices","is_dynamic":true,
"rule":{"field":"consent_status","operator":"equals","value":"opted_in"}}
```

Supported rule fields: **email**, **phone**, **first_name**, **last_name**, **organization**, **group_name**, **interests**, **consent_status**. Operators: **equals**, **contains**, **not_empty**. This is a simple rule engine, not arbitrary SQL or custom-field querying.

7. Templates

GET /api/v1/templates — *scope* templates:read

Optional **limit** query parameter (maximum 500). Returns templates and attachment metadata.

POST /api/v1/templates — *scope* templates:write

```
{"name":"Renewal reminder","channel":"email","subject":"Your renewal is approaching","body":"Hello,
please review your renewal date.","is_html":false}
```

channel must be **email** or **sms**; **name** and **body** are required. Email templates may set **is_html**. Optional **attachments** is a list of objects with **filename**, optional **content_type**, and **content_base64**. Each attachment is limited to 10 MiB and combined attachment payload to 15 MiB. Supported extensions include PDF, TXT, CSV, DOC/DOCX, XLS/XLSX, PNG/JPG/JPEG/GIF/WEBP, ZIP. This API currently creates and lists templates; the direct send endpoint does not accept **template_id**.

8. Reminders

GET /api/v1/reminders — *scope* reminders:read

Optional **limit** (maximum 500). Returns reminders including due/completed times and any associated email delivery status.

POST /api/v1/reminders — **scope** reminders:write

```
{
  "title": "Send renewal documents",
  "details": "Prepare and send the renewal pack",
  "due_at": "2026-11-05T09:00:00+03:00",
  "email": {
    "recipient_mode": "emails",
    "emails": ["jane@example.org"],
    "subject": "Renewal documents",
    "body": "Please review the attached documents.",
    "is_html": false
  }
}
```

`due_at` must be ISO date-time. Email is optional. Recipient modes: `emails` with `emails`, `contacts` with `contact_ids`, `group` with `group_name`, or `groups` with `group_names`. Suppressed and opted-out contacts are excluded for contact/group modes. The implementation caps resolved recipients at 1,000. Attachments use `filename`, optional `content_type`, and `content_base64`; optional per-recipient files are keyed by recipient email in `per_recipient_attachments`. Attachment limits match the template API. Run the reminder/queue worker on a schedule; creating a reminder does not mean an email has already been sent.

9. Notes, CRM interactions and notifications

GET /api/v1/notes — **scope** notes:read

Lists team-visible notes, with optional `limit` (maximum 500).

POST /api/v1/notes — **scope** notes:write

```
{"title": "Customer follow-up", "body": "Discussed next steps; send the revised quote."}
```

Both title and body are required. Notes are team-visible.

GET /api/v1/crm/interactions — **scope** crm:read

Optional query parameters: `contact_id`, `limit` (maximum 500). Lists CRM timeline events.

POST /api/v1/crm/interactions — **scope** crm:write

```
{"contact_id": 123, "kind": "call", "subject": "Renewal discussion", "details": "Customer requested a revised quote.", "outcome": "Quote requested", "follow_up_at": "2026-11-06T10:00:00"}
```

`contact_id` must belong to the token tenant and `subject` is required. `follow_up_at` is optional ISO date-time.

GET /api/v1/notifications — **scope** notifications:read

Query parameters: `unread=1` to filter unread notifications and `limit` (default 100, maximum 500). Returns notification metadata including `id`, `title`, `body`, `level`, `category`, `target_url`, `read_at`, and `created_at` where available. This API is read-only in this version; use the UI to mark notifications read.

10. Tenant webhooks / external callbacks

Webhook subscription endpoints require `webhooks:read` or `webhooks:write`. A subscription is tenant-scoped and receives only selected event types. URLs must be public HTTPS endpoints; localhost/private-network targets are rejected.

GET /api/v1/webhooks — **scope** webhooks:read

Lists active/inactive subscriptions for the tenant (up to 200).

POST /api/v1/webhooks — **scope** webhooks:write

```
curl -X POST https://your-host.example/api/v1/webhooks \
  -H "Authorization: Bearer $COMMS_API_TOKEN" -H "Content-Type: application/json" \
  -d '{"name": "External CRM", "url": "https://crm.example.org/hooks/communications", "events": ["message.queued", "message.accepted", "message.sent", "message.delivered", "message.failed", "email.received", "sms.received", "inbox.message_received"]}'
```

The response includes a random `secret` **once**. Store it securely. Supported events include `message.queued`, `message.accepted`, `message.sent`, `message.delivered`, `message.failed`, `email.received`, `sms.received`, and `inbox.message_received` (check the live response for the supported list). The wildcard `*` subscribes to all supported events.

PATCH /api/v1/webhooks/{subscription_id} — **scope** webhooks:write

```
{ "name": "CRM callback endpoint", "url": "https://crm.example.org/hooks/communications-v2", "events": ["email.received", "sms.received"], "active": true }
```

Fields are optional. New URLs must pass the same public HTTPS validation.

DELETE /api/v1/webhooks/{subscription_id} — **scope** webhooks:write

Soft-deactivates the subscription; it does not erase its delivery history.

GET /api/v1/webhooks/{subscription_id}/deliveries — **scope** webhooks:read

Optional **limit** (default 100, maximum 500). Returns event ID/type, delivery status, attempts, last HTTP status/error, and timestamps.

Webhook signature verification

The delivery worker sends **X-CS-Event-ID**, **X-CS-Timestamp**, and **X-CS-Signature**. Signature format is `sha256=<hex HMAC-SHA256(timestamp + "." + raw_request_body, secret)>`. Verify the signature using the exact raw bytes received, reject stale timestamps, and persist event IDs to prevent duplicate processing. Delivery is at-least-once; return a **2xx** only after accepting the event. Run `flask --app app process-webhooks` on a schedule. **POST /api/v1/webhooks/process** is deliberately not an inline worker; it returns **405 use_worker_command**.

Example Python verification sketch:

```
import hashlib, hmac, time

def verify_webhook(raw_body: bytes, headers, secret: str, max_age=300) -> bool:
    stamp = headers.get("X-CS-Timestamp", "")
    supplied = headers.get("X-CS-Signature", "")
    try:
        if abs(time.time() - int(stamp)) > max_age:
            return False
    except (TypeError, ValueError):
        return False
    expected = "sha256=" + hmac.new(
        secret.encode(), stamp.encode() + b"." + raw_body, hashlib.sha256
    ).hexdigest()
    return hmac.compare_digest(expected, supplied)
```

Inbound event payload example

```
{
  "event_id": "inbox_abc123",
  "event_type": "email.received",
  "created_at": "2026-10-22T16:30:00Z",
  "data": {
    "inbox_id": 987,
    "channel": "email",
    "sender": "customer@example.org",
    "recipient": "support@example.org",
    "subject": "Question about my order",
    "body": "Could you provide an update?",
    "external_id": "provider-or-imap-message-id",
    "source": "support mailbox"
  }
}
```

Actual wrapper fields and available source metadata can vary by event producer. Email/SMS imports enqueue callbacks asynchronously; a failed external app should not prevent the inbound message from being saved. The callback body is bounded/truncated for safety. Inbound callback events only occur when the corresponding source importer/provider path is active and successfully imports the message.

11. Provider callbacks (provider-to-Communications Suite)

These are **incoming requests to this application**, not tenant webhook subscriptions. Configure the provider to POST to the correct endpoint and send **X-Webhook-Token** matching the secret configured for that provider.

POST /webhooks/provider/{provider_id}

Message status callback body:

```
{ "message_id": "provider-message-id-123", "status": "delivered" }
```

Supported status values are **accepted**, **sent**, **delivered**, and **failed**. Unknown provider message IDs return an acknowledged unmatched response; provider IDs must match the message's stored provider message ID.

POST /webhooks/inbound/{provider_id}

Inbound SMS-style callback body:

```
{"from":"+254700000000","to":"+254711111111","body":"I need help with my account","message_id":"incoming-987"}
```

The current generic handler requires sender and body; provider-specific payloads may need an adapter to map their fields into this format. It uses the provider's configured webhook token. Use the tenant subscription events **sms.received** / **inbox.message_received** to fan the stored inbound message out to external applications.

12. Remote GSM/email agent API

These endpoints use a remote-agent bearer token issued by the platform's agent management workflow, **not** a tenant API token. They are designed for the installed remote agent to poll the server.

POST /agent/v1/heartbeat

```
curl -X POST https://your-host.example/agent/v1/heartbeat \
-H "Authorization: Bearer $AGENT_TOKEN" -H 'Content-Type: application/json' \
-d '{"platform":"Windows 11","modem":{"ready":true,"remote_email_ready":false,"model":"USB GSM modem"}}'
```

Updates agent presence/platform/capability metadata and returns server time.

POST /agent/v1/inbound

Posts up to 100 inbound SMS records per request; duplicate fingerprints are ignored.

```
{"messages":[{"from":"+254700000000","to":"+254711111111","body":"HELP","timestamp":"2026-10-22T16:30:00+03:00"}]}
```

Returns **created**, **duplicates**, and **inbox_ids**. STOP/UNSUBSCRIBE/END/QUIT/CANCEL prefixes trigger opt-out handling for a matching contact.

GET /agent/v1/commands

Polls queued remote-agent commands and marks returned commands as processing. Response contains **commands** and **poll_after_seconds**.

POST /agent/v1/commands/{command_id}/result

```
{"status":"completed","detail":"Modem status checked","data":{"signal":18}}
```

status must be **completed** or **failed**; command ownership is checked against the agent and tenant.

GET /agent/v1/jobs

Polls up to 10 eligible remote sending jobs (SMS or remote email when the agent reports those capabilities). The response can contain recipient, channel, body, subject, HTML flag, sender metadata and base64 attachments for email jobs. Poll regularly and report each result exactly once where possible.

POST /agent/v1/jobs/{job_id}/result

```
{"status":"sent","detail":"Provider accepted message"}
```

Supported result statuses are **sent**, **failed**, and **unknown**. **sent** means the agent reported sending; it does not automatically mean handset/inbox delivery. Use **unknown** when the agent cannot determine whether provider submission succeeded.

13. Errors, retry and idempotency

Typical HTTP responses: **200** success/read or idempotent replay, **201** resource created, **202** accepted/queued, **400** invalid input, **401** missing/invalid scope or token, **403** tenant/consent/suppression/frequency policy denial, **404** missing or tenant-inaccessible resource, **409** duplicate, **413** request/batch/attachment too large, **429** frequency limit, **500** unexpected server failure, **405** unsupported operation. Error body is normally JSON with an **error** key; some routes add details.

Use a stable **Idempotency-Key** for each logical outbound send and do not create a new key just because a request timed out. Check message status before deciding whether to retry. Webhook receivers should be idempotent. Remote agents

should not report a message as delivered unless a delivery receipt confirms it.

14. Themes and appearance

Themes are UI settings, not messaging API scopes. Available values are `system`, `light`, `dark`, and `riot`. RIOT 5.0 by Magochi is optional: users can select it in personal Preferences, and permitted administrators can set appearance defaults in Appearance Settings. Accent color, font family, font size and compact layout are appearance preferences. There is no public `/api/v1/theme` endpoint in this build; use the authenticated UI preferences/appearance screens. Do not assume setting a tenant default overrides a user's explicit personal preference.

15. Current implementation boundaries (important)

- Contact CSV import currently caps each request at 10,000 records and reads the whole upload; resumable/streaming 500-million-record ingestion is not implemented or benchmarked.
- The direct send endpoint does not currently accept a template ID, custom per-message attachments, or a one-off callback URL. Use template endpoints for template storage and webhook subscriptions for callbacks.
- Custom CRM fields are stored and validated, and queued message bodies resolve `{{custom.FIELD_KEY}}` tokens at send time. Dynamic-group rules, CSV import mapping, and report exports do not yet universally resolve custom values.
- Webhook delivery is asynchronous and requires the worker/scheduler. Source import/provider availability determines whether inbound events are emitted.
- The provider callback schema is generic. Provider-specific payload formats may require a mapping adapter.
- This documentation is exhaustive for the source routes at the time of this package, not a claim that every planned Enterprise feature is finished. Check `openapi.yaml`, this file, `API_GUIDE_ENTERPRISE.md`, and `RELEASE_STATUS.md` together; if they disagree, treat source code as authoritative and report the discrepancy before production rollout.

Appendix A — Source route inventory

This table is generated from the Flask route decorators in `app.py` for health, tenant API, remote-agent API, and provider webhook routes. Variable path parameters are shown in Flask notation. It is an inventory, not a replacement for the endpoint behavior and payload examples above.

Path | Methods | Handler

<code>/health</code>	<code>GET</code>	<code>health</code>
<code>/agent/v1/inbound</code>	<code>POST</code>	<code>agent_inbound_messages</code>
<code>/agent/v1/heartbeat</code>	<code>POST</code>	<code>agent_heartbeat</code>
<code>/agent/v1/commands</code>	<code>GET</code>	<code>agent_commands_poll</code>
<code>/agent/v1/commands/<int:command_id>/result</code>	<code>POST</code>	<code>agent_command_result</code>
<code>/agent/v1/jobs</code>	<code>GET</code>	<code>agent_jobs</code>
<code>/agent/v1/jobs/<int:job_id>/result</code>	<code>POST</code>	<code>agent_job_result</code>
<code>/api/v1/contacts</code>	<code>GET, POST</code>	<code>api_contacts</code>
<code>/api/v1/contact-fields</code>	<code>GET, POST</code>	<code>api_contact_custom_fields</code>
<code>/api/v1/contact-fields/<int:field_id></code>	<code>PATCH, DELETE</code>	<code>api_contact_custom_field_detail</code>
<code>/api/v1/contacts/<int:contact_id>/fields</code>	<code>GET, PUT, PATCH</code>	<code>api_contact_custom_field_values</code>
<code>/api/v1/contacts/import</code>	<code>POST</code>	<code>api_contacts_import</code>
<code>/api/v1/messages/<channel></code>	<code>POST</code>	<code>api_send</code>
<code>/api/v1/messages/<int:mid></code>	<code>GET</code>	<code>api_message_status</code>
<code>/api/v1/inbox</code>	<code>GET</code>	<code>api_inbox</code>
<code>/api/v1/engagement</code>	<code>GET</code>	<code>api_engagement</code>
<code>/api/v1/webhooks</code>	<code>GET, POST</code>	<code>api_webhooks</code>
<code>/api/v1/webhooks/<int:subscription_id></code>	<code>PATCH, DELETE</code>	<code>api_webhook_detail</code>

/api/v1/webhooks/<int:subscription_id>/deliveries | GET | api_webhook_deliveries
/api/v1/webhooks/process | POST | api_process_webhooks
/webhooks/provider/<int:provider_id> | POST | provider_webhook
/webhooks/inbound/<int:provider_id> | POST | inbound_provider_webhook
/api/v1/groups | GET, POST | api_groups
/api/v1/templates | GET, POST | api_templates
/api/v1/notifications | GET | api_notifications
/api/v1/reminders | GET, POST | api_create_reminder
/api/v1/notes | GET, POST | api_notes
/api/v1/crm/interactions | GET, POST | api_crm_interactions

Part II — Enterprise Integration Guide

Communications Suite API Guide — Enterprise Development Candidate

Base URL: `https://your-host.example` (replace with your deployment URL) **API version:** `/api/v1` **Authentication:** `Authorization: Bearer <API_TOKEN>` **Content type:** `application/json` unless the endpoint accepts multipart CSV upload.

Important: this guide describes the current API routes identified in this source package. It does not imply that planned webhook subscription management, custom-field CRUD, async XLSX import, or every endpoint in the Enterprise plan has already been implemented. See `openapi.yaml` and `ENTERPRISE_GAP_REGISTER.md` for the exact current surface and remaining work.

Complete API reference: See `API_REFERENCE_COMPLETE.md` (`API_REFERENCE_COMPLETE.md`) for request/response examples for all implemented `/api/v1/*` and `/agent/v1/*` endpoints, provider callbacks, authentication scopes, webhook signatures, and theme settings. Keep this reference bundled with the source release.

API token scopes

The current `ApiToken` model stores comma-separated scopes. The token creation CLI prints the raw token once; the database stores its SHA-256 hash. Grant only required scopes. Typical scopes include `contacts:read`, `contacts:write`, `messages:send`, `messages:read`, `inbox:read`, `engagement:read`, `groups:read`, `groups:write`, `templates:read`, `templates:write`, `reminders:read`, `reminders:write`, `notes:read`, `notes:write`, `crm:read`, `crm:write`, and `notifications:read`.

Create a token from the application environment:

```
flask --app app create-api-token TENANT_ID "Integration name" "contacts:read,messages:send,messages:read"
```

Save the token securely when it is printed. Never place it in browser JavaScript, source control, screenshots, or logs.

Send one email

The currently implemented send route uses `recipient` and `body` fields. The recipient must already be a tenant contact with `consent_status=opted_in`, must not be suppressed, and must be within the configured frequency limit.

```
curl -X POST "https://your-host.example/api/v1/messages/email" \
-H "Authorization: Bearer $COMMS_API_TOKEN" \
-H "Content-Type: application/json" \
-H "Idempotency-Key: invoice-4821-email" \
-d '{"recipient":"customer@example.com","subject":"Invoice 4821","body":"Your invoice is ready.",
"is_html":false}'
```

For HTML email, set `is_html` to `true`; the current handler sanitizes HTML. A `202` response means queued, not delivered. Poll `GET /api/v1/messages/{id}` for current status.

Send one SMS

```
curl -X POST "https://your-host.example/api/v1/messages/sms" \
-H "Authorization: Bearer $COMMS_API_TOKEN" \
-H "Content-Type: application/json" \
-H "Idempotency-Key: invoice-4821-sms" \
-d '{"recipient":"+2547XXXXXXX","body":"Your invoice 4821 is due tomorrow."}'
```

Use a real E.164 recipient and ensure the contact is opted in and not suppressed. Provider delivery receipts depend on the configured SMS provider or GSM agent.

Python example

```
import os
import requests

BASE = os.environ["COMMS_BASE_URL"].rstrip("/")
TOKEN = os.environ["COMMS_API_TOKEN"]
response = requests.post(
    f"{BASE}/api/v1/messages/email",
    headers={"Authorization": f"Bearer {TOKEN}", "Idempotency-Key": "welcome-1001"},
    json={"recipient": "customer@example.com", "subject": "Welcome",
        "body": "Welcome to our service.", "is_html": False},
    timeout=20,
)
response.raise_for_status()
print(response.json())
```

PHP example

```
<?php
$base = rtrim(getenv('COMMS_BASE_URL'), '/');
$token = getenv('COMMS_API_TOKEN');
$payload = json_encode([
    'recipient' => 'customer@example.com',
    'subject' => 'Welcome',
    'body' => 'Welcome to our service.',
    'is_html' => false
]);
$ch = curl_init($base . '/api/v1/messages/email');
curl_setopt_array($ch, [
    CURLOPT_POST => true,
    CURLOPT_POSTFIELDS => $payload,
    CURLOPT_HTTPHEADER => ['Authorization: Bearer ' . $token,
        'Content-Type: application/json', 'Idempotency-Key: welcome-1001'],
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_TIMEOUT => 20
]);
$result = curl_exec($ch);
$status = curl_getinfo($ch, CURLINFO_HTTP_CODE);
if ($result === false || $status >= 400) { throw new RuntimeException(curl_error($ch) ?: $result); }
echo $result;
curl_close($ch);
```

JavaScript / Node example

```
const response = await fetch(`${process.env.COMMS_BASE_URL}/api/v1/messages/sms`, {
  method: "POST",
  headers: {
    Authorization: `Bearer ${process.env.COMMS_API_TOKEN}`,
    "Content-Type": "application/json",
    "Idempotency-Key": "appointment-889"
  },
  body: JSON.stringify({
    recipient: "+2547XXXXXXX",
    body: "Your appointment is tomorrow."
  })
});
if (!response.ok) throw new Error(`API failed: ${response.status} ${await response.text()}`);
console.log(await response.json());
```

Contacts, groups and templates

- **GET /api/v1/contacts** and **POST /api/v1/contacts**: tenant-scoped contact access/creation.
- **POST /api/v1/contacts/import**: JSON or multipart CSV import, as supported by the current route.
- **GET /api/v1/groups** and **POST /api/v1/groups**: list/create static or supported dynamic groups. API dynamic-group rules also support tenant custom fields via **field: "custom:FIELD_KEY"** with **equals**, **contains**, or **not_empty**; create the field first and keep it active.
- **GET /api/v1/templates** and **POST /api/v1/templates**: list/create templates; template creation supports bounded base64 attachments in JSON.
- **GET /api/v1/notes**, **POST /api/v1/notes**: CRM notes.
- **GET /api/v1/crm/interactions**, **POST /api/v1/crm/interactions**: CRM timeline interactions.

- `GET /api/v1/reminders`, `POST /api/v1/reminders`: reminder operations.
- `GET /api/v1/notifications`: tenant-visible notifications.
- `GET /api/v1/inbox`: inbox list with channel/status/address/search/limit filters.
- `GET /api/v1/engagement`: engagement events with address/type/day filters.

Request and response details should be checked against `openapi.yaml` and the deployed version. Keep page sizes bounded and do not assume every field shown in the UI is writable through the API.

Provider callbacks currently present

Provider integrations may use:

- `POST /webhooks/provider/{provider_id}` for message status updates.
- `POST /webhooks/inbound/{provider_id}` for inbound provider messages.

These routes use the configured provider webhook token via `X-Webhook-Token`. Treat that token as a secret. The current routes are provider-specific callbacks, not a full customer-managed webhook subscription system. Durable customer webhooks, event filtering, HMAC signatures and retry history remain Enterprise implementation items.

Status and errors

- `queued`: accepted for asynchronous processing.
- `processing`: worker currently handling the message.
- `accepted` / `sent`: provider-specific progress; these do not guarantee inbox delivery.
- `delivered`: provider reported delivery, when the channel supports receipts.
- `failed`: permanent failure or exhausted retries.
- `unknown`: outcome could not safely be determined, for example after a worker crashed during provider submission.

Use `Idempotency-Key` on send requests. Reusing a key with the same token should return the existing message where supported. Do not retry a timed-out send with a new idempotency key until the previous outcome is checked.

Excel/CSV and custom fields

The current contact import route is not a certification of a 500-million-row ingestion pipeline. For very large datasets, implement a staged streaming import with checkpoints, bulk writes, resumable jobs, dedupe strategy and error exports. Do not load the whole file into memory. Personalized sends should validate the recipient field, field mapping, consent, suppression, frequency limits and preview samples before enqueueing.

Security

Use HTTPS; rotate tokens after suspected exposure; keep secrets server-side; use least-privilege scopes; restrict upload sizes; and audit exports and bulk sends. Publicly discovered email addresses are not proof of consent to market to them. Verify provider webhook authentication and protect callback endpoints against replay and excessive request rates.

Tenant webhooks (Enterprise development)

Webhook API endpoints use tenant-scoped API tokens with `webhooks:read` and/or `webhooks:write` scopes:

- `GET /api/v1/webhooks` — list subscriptions.
- `POST /api/v1/webhooks` — create a subscription; the signing secret is returned once.
- `PATCH /api/v1/webhooks/{id}` — update subscription fields.
- `DELETE /api/v1/webhooks/{id}` — deactivate a subscription.
- `GET /api/v1/webhooks/{id}/deliveries` — inspect delivery history.
- `flask --app app process-webhooks` — deliver due outbox records and retry transient failures.

Creation example:

```
curl -X POST https://your-domain.example/api/v1/webhooks \
  -H 'Authorization: Bearer YOUR_TOKEN' -H 'Content-Type: application/json' \
```

```
-d '{"name":"Billing callbacks","url":"https://hooks.example.net/communications","events":["message.queued","message.accepted","message.delivered","message.failed","email.received","sms.received"]}'
```

The response's `secret` is displayed only once. The receiver should calculate HMAC-SHA256 over `timestamp + "." + raw_request_body` using that secret and compare it to `X-CS-Signature (sha256=<hex>)`, while validating `X-CS-Timestamp` freshness and deduplicating `X-CS-Event-ID`. Deliveries are at-least-once; duplicate event IDs are possible. HTTPS endpoints resolving to non-public IP addresses are rejected to reduce SSRF risk. Run the webhook worker or the protected scheduler regularly. A `2xx` response is considered delivered; other responses are retried with bounded exponential backoff.

Inbound message callbacks and outbound API sends

An integration can subscribe its public HTTPS callback URL to these events:

- `message.queued` — the API accepted a send request and queued the message; this is **not** proof of delivery.
- `message.accepted`, `message.sent`, `message.delivered`, `message.failed` — later message state events where available from the provider/worker.
- `email.received` — a new message was imported from a configured IMAP mailbox or received through an inbound provider callback.
- `sms.received` — a new SMS was received through a supported remote/GSM agent or inbound provider callback.
- `inbox.message_received` — channel-neutral inbound event for email or SMS.

Example subscription body:

```
{"name":"CRM inbound sync","url":"https://crm.example.org/hooks/communications","events":["message.queued","email.received","sms.received","inbox.message_received"]}
```

Inbound event payloads include `inbox_id`, `channel`, `sender`, `recipient`, `body`, `external_id`, and source metadata; email payloads may also include `subject`. Payloads intentionally truncate body text to 20,000 characters. Event delivery is asynchronous via the durable outbox: run `flask --app app process-webhooks` periodically or ensure the protected scheduler is active. The external application must return a `2xx` response, verify the HMAC signature, and deduplicate `X-CS-Event-ID`. A subscription is only invoked for events explicitly selected (or `*`). Callback targets must be public HTTPS URLs.

Message analytics report API

`GET /api/v1/reports/messages` requires the existing `messages:read` scope and only returns data for the token's tenant. Optional query parameters are `from=YYYY-MM-DD`, `to=YYYY-MM-DD`, and `channel=email|sms`. If both dates are supplied, the response includes daily counts; date ranges are limited to 366 calendar days. Invalid dates, reversed ranges, unsupported channels, and oversized ranges return a `400` response. Status values reflect the application's recorded queue/provider state and should not be interpreted as proof of final delivery unless the status is explicitly `delivered` and the provider supplies that confirmation.

Example:

```
GET /api/v1/reports/messages?from=2026-10-01&to=2026-10-22&channel=email
Authorization: Bearer YOUR_API_TOKEN
```

Example response:

```
{
  "total": 128,
  "by_status": {"delivered": 110, "failed": 3, "queued": 5, "sent": 10},
  "by_channel": {"email": 128},
  "daily": [{"date": "2026-10-01", "count": 7}],
  "filters": {"from": "2026-10-01", "to": "2026-10-22", "channel": "email"}
}
```

The example numbers are illustrative. The endpoint returns actual values from the tenant database.

Tenant-defined CRM custom fields (v0.4.0 development increment)

Custom field endpoints use the existing `crm:read` and `crm:write` token scopes and are always tenant-scoped.

- `GET /api/v1/contact-fields` lists the tenant's field definitions.
- `POST /api/v1/contact-fields` creates a field. Example: `{"key":"invoice_number","label":"Invoice number","type":"text","required":false}`.

- **PATCH** `/api/v1/contact-fields/{field_id}` updates its label, required state, active state or select options.
- **DELETE** `/api/v1/contact-fields/{field_id}` soft-disables a field; it does not erase stored contact values.
- **GET** `/api/v1/contacts/{contact_id}/fields` returns active definitions and saved values for a tenant-owned contact.
- **PUT** or **PATCH** `/api/v1/contacts/{contact_id}/fields` accepts `{"values":{"invoice_number":"INV-1001","is_vip":true}}`.

Supported types are `text`, `textarea`, `email`, `phone`, `url`, `number`, `boolean`, `date`, and `select`. Select fields require a non-empty `options` array. Values are validated before saving. Invalid field keys or values return a structured 4xx response. Definitions and values are stored in separate tables, so existing Contact columns and records remain unchanged.

Custom fields are also available to supported dynamic group rules (`field: "custom:FIELD_KEY"`) and are resolved for personalized queued email/SMS bodies using `{{custom.FIELD_KEY}}`. To export a tenant contact directory with active custom fields, use **GET** `/api/v1/contacts/export.csv` with `contacts:read`. The export streams records in bounded batches, includes custom columns named `custom.FIELD_KEY`, and prefixes spreadsheet formula-leading cells to reduce CSV formula-injection risk. The export is a snapshot assembled batch-by-batch; contacts created or edited while a long export is running may not represent one transactionally consistent point-in-time snapshot.

Export tenant contacts to CSV

```
GET /api/v1/contacts/export.csv?batch_size=1000
Authorization: Bearer YOUR_API_TOKEN
```

Example response headers include `Content-Type: text/csv; charset=utf-8` and `Content-Disposition: attachment; filename="contacts_export.csv"`. `batch_size` is optional and clamped to 100–2,000 rows. The endpoint is tenant-scoped and requires `contacts:read`; it includes active custom fields only. Do not publish exported contact data or treat an exported address as proof of marketing consent.