



MHI Gateway

Developer Integration Guide

SMS API v1 implementation and production guidance

Contract version 1.0.0-rc1

Document version 1.0

Publication date 28 July 2026

Classification Public integration documentation

MICRO HEALTH INITIATIVE

Table of Contents

Table of Contents	2
1. Introduction	4
Contract surface	4
Base URL	4
2. Platform Overview	5
3. Integration Architecture	6
4. Authentication	7
5. Authorization and Tenant Isolation	8
6. API Keys	9
7. Correlation IDs	10
8. Idempotency	11
Replay behavior	11
9. Sending SMS	12
Request fields	12
Request	12
Accepted response	12
10. Bulk SMS	13
Request shape	13
Result processing	13
11. Message Status	14
Full message resource	14
Compact status resource	14
Status handling	14
12. Webhooks	15
Configure the receiver	15
Delivery event	15
Verify the signature	15
13. Error Handling	16
Official error codes	16
Retry policy	16
14. SDK Examples	17
cURL	17
PHP 8	17
Node.js	17
Python	17
Java 11+	18
Go	18
C# (.NET)	18
15. Security	19
Protect sensitive data	19
Network and application controls	19
16. Best Practices	20

Submission	20
Status and webhooks	20
Resilience	20
17. Production Readiness	21
18. Troubleshooting	22
Requests return 401 AUTHENTICATION_FAILED	22
A send returns 409 IDEMPOTENCY_CONFLICT	22
A bulk request returns 202 with rejected items	22
A message remains queued, submitted, or unknown	22
Webhook signature verification fails	22
Webhook events are not retried	22
19. FAQ	23
Does 202 Accepted mean the SMS was delivered?	23
Can I omit sender_id?	23
Can I reuse a client_reference as an idempotency key?	23
Can I retrieve a bulk request by batch_id?	23
Are bulk requests transactional?	23
Which statuses generate webhooks?	23
Does the gateway retry failed webhook deliveries?	23
Can I configure several webhook URLs?	23
Is there a webhook delete operation?	23
20. Appendices	24
Appendix A: Header reference	24
Appendix B: Identifier and timestamp formats	24
Appendix C: Public status transitions	24
Appendix D: Related documentation	24
Appendix E: Contract limitations and approval items	25

1. Introduction

MHI Gateway exposes a tenant-scoped REST API for submitting SMS messages, retrieving their normalized status, and configuring one delivery webhook. Requests and responses use JSON over HTTPS.

This guide covers the approved RC1 public contract. It does not describe internal routes, provider identifiers, SMPP details, billing data, or features that are not present in OpenAPI.

Contract surface

Operation	Method and path	Success
Submit one message	POST /api/v1/messages	202
Submit a bulk request	POST /api/v1/messages/bulk	202
Retrieve one message	GET /api/v1/messages/{message_id}	200
Retrieve message status	GET /api/v1/messages/{message_id}/status	200
Replace webhook configuration	PUT /api/v1/webhook	200
Retrieve webhook configuration	GET /api/v1/webhook	200

The gateway also sends a signed delivery event to the configured webhook URL. There is no public batch retrieval resource, cancellation endpoint, webhook deletion endpoint, or webhook retry endpoint in this contract.

Base URL

The OpenAPI server path is /api/v1. Combine it with the environment host provided by the gateway operator:

```
https://<API_HOST>/api/v1
```

Keep test and production hosts, credentials, idempotency keys, and webhook secrets separate.

2. Platform Overview

An integration has four principal responsibilities:

1. Authenticate each API request with a tenant-scoped Bearer credential.
2. submit a single message or an ordered bulk request;
3. persist returned identifiers and correlate them with local business records;
4. consume signed terminal-status webhooks and reconcile non-terminal messages

through status reads.

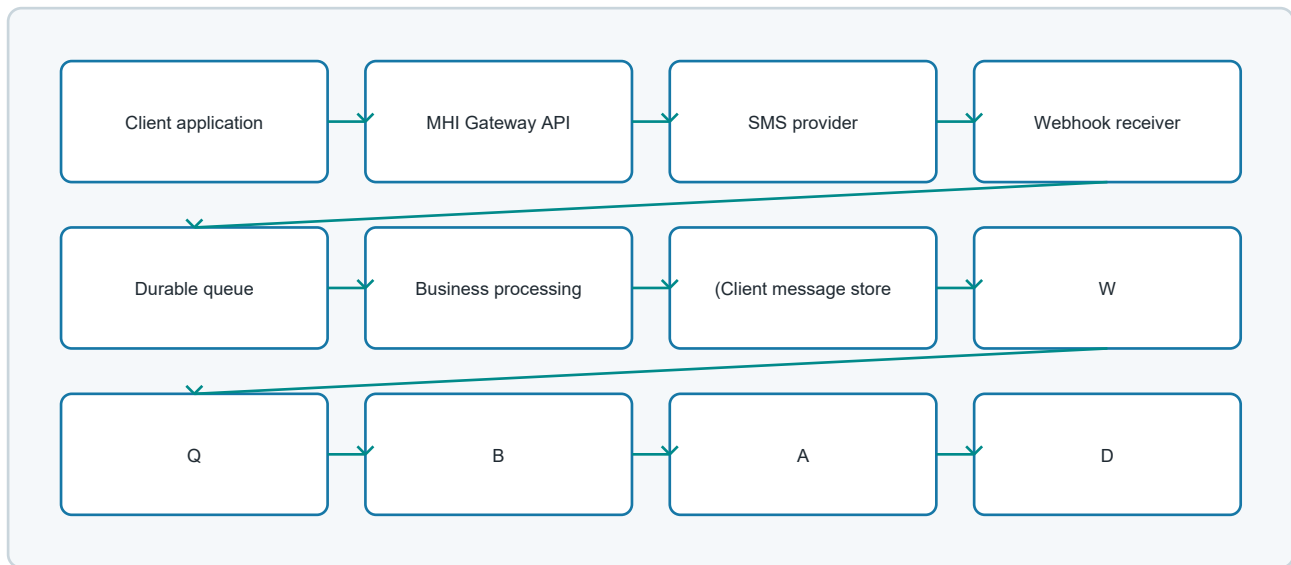
The public SMS lifecycle is normalized across underlying providers:

```
queued -> submitted -> delivered | failed | expired | rejected
      |           |           ^
      +-----+-----+
      |
      unknown -> delivered | failed | expired | rejected
```

delivered, failed, expired, and rejected are terminal. unknown is non-terminal and indicates that the authoritative outcome is unavailable or contradictory.

3. Integration Architecture

Keep the API client, durable state, webhook receiver, and business processing separate:



Recommended client-side components are:

- a gateway client that owns authentication, timeouts, JSON parsing, and stable error handling;
- a durable message store containing `message_id`, `client_reference`, idempotency key, original result, and current normalized status;
- a webhook edge handler that verifies the raw request before parsing JSON;
- an idempotent event processor behind a durable queue;
- a reconciliation worker that polls messages which remain non-terminal.

4. Authentication

Every API operation uses HTTP Bearer authentication:

```
Authorization: Bearer <API_TOKEN>
```

The credential format is:

```
mhi_<environment>_<12-character-prefix>.<48-character-secret>
```

Treat the entire value as opaque. Do not parse it to make authorization or environment decisions. A missing or invalid credential returns 401 with `AUTHENTICATION_FAILED`.

Use HTTPS for all production requests. Send:

```
Accept: application/json  
Content-Type: application/json
```

Content-Type is needed when a request has a JSON body.

5. Authorization and Tenant Isolation

Credentials are tenant-scoped. Resources are also scoped to the authenticated tenant and application. Never use a credential issued to one tenant to access another tenant's messages.

Webhook configuration operations may return 403 `FORBIDDEN` when the authenticated principal lacks permission. A message outside the authenticated scope is reported as absent; integrations must not use error behavior to infer the existence of another tenant's resource.

6. API Keys

Obtain credentials from the gateway operator. Credential issuance, expiry, revocation, and storage are operator-managed.

Client applications should:

- store credentials in a secret manager;
- inject credentials at runtime;
- use different credentials in each environment;
- restrict credential access to the smallest necessary set of workloads;
- redact the `Authorization` header from logs and tracing;
- stop using a credential immediately after revocation.

The RC1 contract notes that a zero-downtime credential rotation procedure remains an operational approval item. Coordinate rotation with the gateway operator; do not assume an undocumented overlap period.

7. Correlation IDs

Send an optional `X-Correlation-ID` on any API request to connect client and gateway telemetry:

```
X-Correlation-ID: order-1042-attempt-1
```

The value must:

- contain 1 to 100 characters;
- start with an ASCII letter or digit;
- contain only ASCII letters, digits, `.`, `_`, `:`, or `-` thereafter.

Every HTTP response, whether successful or an error, carries two request identifier headers:

- `X-Request-ID` is a unique identifier for the current HTTP request.
- `X-Correlation-ID` is the current request correlation identifier. It contains

the valid client-supplied value or a UUIDv4 generated by the gateway.

These headers describe the current HTTP exchange, including an idempotent replay. Capture both headers for successful and error responses so support and client telemetry can identify the exact request.

The body `correlation_id` on a single-message submission identifies the original message submission. On replay, the body retains that original submission correlation identifier, while the two response headers describe the current replay request. Do not replace the stored body value with a replay response header.

8. Idempotency

POST /messages and POST /messages/bulk require:

Idempotency-Key: <UNIQUE_OPERATION_KEY>

An idempotency key is tenant-scoped, visible ASCII, and 1 to 191 characters. Generate a fresh, unpredictable key for each intended operation. Reuse it only when retrying the same canonical request.

Replay behavior

- The first valid request binds the key to its canonical request and original result.

- An equivalent replay returns the original result and status code.
- Reusing the key with different canonical input returns

409 IDEMPOTENCY_CONFLICT.

- Failed validation does not bind the key.
- Concurrent equivalent requests converge on one stable result.
- Different tenants may use the same key independently.

Bulk array order is part of canonical equivalence. Reordering otherwise identical items is a different request.

Persist the key with the request payload and result. Never automatically replace a key after an ambiguous timeout: retry the same request with the same key to discover the stable result.

> RC1 approval item: idempotency retention is not defined by the approved > contract. Do not build cleanup or replay guarantees around an assumed > retention period; obtain an operator-approved value before production launch.

9. Sending SMS

Send one SMS using POST /api/v1/messages.

Request fields

Field	Required	Rules
recipient	Yes	E.164: +, then 8-15 digits; first digit non-zero
message	Yes	String, 1-4096 characters
sender_id	Yes	1-11 ASCII alphanumeric; active assignment
client_reference	No	1-100 characters matching [A-Za-z0-9][A-Za-z0-9._:-]{0,99}

Unknown fields are rejected. sender_id is required, active, and assigned to the authenticated application. Provider approval occurs outside the gateway.

Request

```
curl --request POST 'https://<API_HOST>/api/v1/messages' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --header 'Authorization: Bearer <API_TOKEN>' \
  --header 'Idempotency-Key: 6b11f086-1e8d-4bb9-a680-1524e8d23b21' \
  --header 'X-Correlation-ID: order-1042-attempt-1' \
  --data '{
    "recipient": "+255712345678",
    "message": "Your verification code is 123456.",
    "sender_id": "MHI",
    "client_reference": "order-1042"
  }'
```

Accepted response

```
HTTP/1.1 202 Accepted
Content-Type: application/json
X-Request-ID: 01JREQUEST000000000000000000
X-Correlation-ID: order-1042-attempt-1
```

```
{
  "message_id": "01JEXAMPLE000000000000000000",
  "status": "queued",
  "client_reference": "order-1042",
  "created_at": "2026-07-19T12:30:40.123456Z",
  "correlation_id": "order-1042-attempt-1"
}
```

message_id, status, created_at, and correlation_id are required. client_reference appears when supplied. Persist the complete original result before starting dependent work.

The endpoint's OpenAPI responses are 202, 401, 409, 422, 429, and 500.

10. Bulk SMS

Submit 1 to 100 ordered candidates using POST `/api/v1/messages/bulk`. Processing is best effort: a rejected item does not reject valid siblings.

Request shape

```
{
  "client_reference": "campaign-42",
  "messages": [
    {
      "recipient": "+255712345678",
      "message": "Your appointment is confirmed.",
      "sender_id": "MHI",
      "client_reference": "row-1"
    },
    {
      "recipient": "+255713456789",
      "message": "Your appointment is confirmed.",
      "sender_id": "MHI",
      "client_reference": "row-2"
    }
  ]
}
```

The top-level `client_reference` is optional and follows the same 1-to-100 character pattern as a single-message reference. `messages` is required and must contain 1 to 100 items. Unknown top-level and item fields are rejected. Candidate fields are evaluated independently after the top-level request is accepted.

Result processing

A valid evaluated request returns 202, even when every item is rejected:

```
{
  "batch_id": "550e8400-e29b-41d4-a716-446655440001",
  "client_reference": "campaign-42",
  "accepted": 1,
  "rejected": 1,
  "results": [
    {
      "index": 0,
      "message_id": "01JEXAMPLE0000000000000000",
      "status": "queued",
      "client_reference": "row-1"
    },
    {
      "index": 1,
      "error": {
        "code": "INVALID_RECIPIENT",
        "message": "Recipient is invalid.",
        "details": {}
      }
    }
  ]
}
```

Every input has exactly one result at the same zero-based index. `accepted` + `rejected` equals the number of results. An accepted item has `message_id` and the fixed initial status `queued`. A rejected item has `error.code`, `error.message`, and `error.details`.

The approved item error codes are:

- `INVALID_REQUEST`
- `INVALID_RECIPIENT`
- `INVALID_SENDER_ID`
- `MESSAGE_TOO_LONG`
- `SERVICE_UNAVAILABLE`
- `INTERNAL_ERROR`

`batch_id` is a replay-stable UUIDv4 used for tracking the result. It is not a public resource, so do not attempt to retrieve it with a separate endpoint. Persist every accepted `message_id`.

The endpoint's OpenAPI responses are 202, 401, 409, 422, 429, and 500.

11. Message Status

There are two distinct retrieval operations.

Full message resource

GET /api/v1/messages/{message_id} returns:

```
{
  "message_id": "01JEXAMPLE0000000000000000",
  "recipient": "+255712345678",
  "sender_id": "MHI",
  "client_reference": "order-1042",
  "status": "submitted",
  "created_at": "2026-07-19T12:30:40.123456Z",
  "submitted_at": "2026-07-19T12:30:41.234567Z",
  "correlation_id": "status-read-1042"
}
```

Required fields are message_id, recipient, status, created_at, and correlation_id. sender_id, client_reference, submitted_at, and finalized_at are optional.

Compact status resource

GET /api/v1/messages/{message_id}/status returns:

```
{
  "message_id": "01JEXAMPLE0000000000000000",
  "status": "delivered",
  "occurred_at": "2026-07-19T12:30:44.987654Z",
  "client_reference": "order-1042",
  "correlation_id": "status-read-1042"
}
```

Required fields are message_id, status, occurred_at, and correlation_id. client_reference is optional.

message_id is a 26-character public ULID using characters in [0-9A-HJKMNP-TV-Z]. Both retrieval operations may return 200, 401, 404, 429, or 500.

Status handling

Status	Terminal	Client action
queued	No	Wait for submission or terminal evidence
submitted	No	Continue webhook consumption and reconciliation
unknown	No	Reconcile until a terminal outcome is available
delivered	Yes	Record successful delivery
failed	Yes	Record final failure
expired	Yes	Record final expiry
rejected	Yes	Record final rejection

Do not rewrite one terminal state with a different terminal state. The gateway treats terminal public history as immutable.

12. Webhooks

The API supports exactly one delivery webhook configuration per tenant.

Configure the receiver

PUT /api/v1/webhook creates the configuration when absent and completely replaces it when present:

```
{
  "url": "https://example.com/webhooks/mhi-gateway",
  "secret": "<AT_LEAST_32_RANDOM_BYTES>",
  "enabled": true
}
```

All three fields are required. Production URLs must use HTTPS. The secret is write-only and is never returned. A replacement secret is active immediately; the previous secret remains valid for receiver verification for 24 hours.

The safe response contains only url, enabled, created_at, and updated_at. GET /api/v1/webhook retrieves that safe representation.

Delivery event

The gateway emits one event when a message first reaches delivered, failed, expired, or rejected. It does not emit events for queued, submitted, or unknown.

```
{
  "message_id": "01JEXAMPLE0000000000000000",
  "status": "delivered",
  "client_reference": "order-1042",
  "delivered_at": "2026-07-19T12:30:44.987654Z",
  "timestamp": "2026-07-19T12:30:45.123456Z"
}
```

message_id, status, and timestamp are required. client_reference and delivered_at are optional.

Verify the signature

Each delivery includes:

```
X-Webhook-Timestamp: <UNIX_SECONDS>
X-Webhook-Signature: v1=<64_LOWERCASE_HEX_CHARACTERS>
```

Verification order:

1. Read and retain the exact raw request body.
2. Parse X-Webhook-Timestamp as Unix seconds.
3. Reject a timestamp more than five minutes from the receiver's current time.
4. Build the exact byte sequence <timestamp>.<raw-request-body>.
5. Calculate HMAC-SHA256 with the configured secret.
6. Prefix the lowercase hexadecimal digest with v1=.
7. Compare the supplied and calculated signatures in constant time.
8. Atomically reject a previously processed replay identity.
9. Parse JSON only after verification.
10. Durably hand off the event and return any 2xx promptly.

The outbound timeout is 10 seconds. RC1 performs exactly one delivery attempt and schedules no retries. A delivery failure does not roll back the message's status. Continue status reconciliation even when webhooks are enabled.

See the dedicated [Webhook Integration Guide](#) when that documentation milestone is available.

13. Error Handling

Request-level failures use one envelope:

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/json
X-Request-ID: 01JREQUEST000000000000000001
X-Correlation-ID: order-1042-attempt-1

{
  "error": {
    "code": "INVALID_REQUEST",
    "message": "The request is invalid.",
    "details": {},
    "correlation_id": "01JEXAMPLE0000000000000000"
  }
}
```

Branch on `error.code`, not `error.message`. Messages and details are sanitized and may change without altering the stable code. Capture X-Request-ID and X-Correlation-ID before handling the body, including on non-2xx responses.

Official error codes

HTTP status	Code	General client action
401	AUTHENTICATION_FAILED	Supply a valid credential
403	FORBIDDEN	Request the required permission
400	INVALID_REQUEST	Correct the request structure
422	INVALID_RECIPIENT	Correct the E.164 recipient
422	INVALID_SENDER_ID	Use an approved valid sender
422	MESSAGE_TOO_LONG	Reduce the message to at most 4096 characters
409	IDEMPOTENCY_CONFLICT	Do not retry with changed input under the same key
404	MESSAGE_NOT_FOUND	Verify the ID and authenticated scope
422	WEBHOOK_CONFIGURATION_INVALID	Correct the webhook configuration
429	RATE_LIMIT_EXCEEDED	Respect Retry-After and apply bounded backoff
503	SERVICE_UNAVAILABLE	Retry safely with bounded backoff
500	INTERNAL_ERROR	Retry safely only when the operation is idempotent

The OpenAPI operation response lists determine which HTTP statuses are declared for each endpoint. The complete code vocabulary above is the official error schema; a code should be interpreted with the returned HTTP status and endpoint context.

Retry policy

Do not retry authentication, authorization, validation, not-found, or idempotency-conflict failures without correcting their cause.

For 429, follow Retry-After. For a transient transport failure, 500, or 503, use exponential backoff with jitter, a maximum attempt count, and an overall deadline. Retry submission requests with the original body and original idempotency key.

14. SDK Examples

These examples submit the exact MessageSubmission schema. Set BASE_URL without a trailing slash and load API_TOKEN from a secret manager.

cURL

```
curl --request POST "${BASE_URL}/api/v1/messages" \
  --header "Authorization: Bearer ${API_TOKEN}" \
  --header "Accept: application/json" \
  --header "Content-Type: application/json" \
  --header "Idempotency-Key: 6b11f086-1e8d-4bb9-a680-1524e8d23b21" \
  --data '{"recipient":"+255712345678","message":"Your verification code is 123456.","sender_id":"MHI","client_reference":"order-1042"}'
```

PHP 8

```
<?php

$payload = json_encode([
    'recipient' => '+255712345678',
    'message' => 'Your verification code is 123456.',
    'sender_id' => 'MHI',
    'client_reference' => 'order-1042',
], JSON_THROW_ON_ERROR);

$handle = curl_init(trim((string) getenv('BASE_URL'), '/') . '/api/v1/messages');
curl_setopt_array($handle, [
    CURLOPT_POST => true,
    CURLOPT_POSTFIELDS => $payload,
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_TIMEOUT => 15,
    CURLOPT_HTTPHEADER => [
        'Authorization: Bearer ' . getenv('API_TOKEN'),
        'Accept: application/json',
        'Content-Type: application/json',
        'Idempotency-Key: 6b11f086-1e8d-4bb9-a680-1524e8d23b21',
    ],
]);
$response = curl_exec($handle);
if ($response === false) {
    throw new RuntimeException('Gateway request failed.');
}
$status = curl_getinfo($handle, CURLINFO_RESPONSE_CODE);
curl_close($handle);
$body = json_decode($response, true, 512, JSON_THROW_ON_ERROR);
```

Node.js

```
const response = await fetch(`${process.env.BASE_URL}/api/v1/messages`, {
  method: 'POST',
  signal: AbortSignal.timeout(15000),
  headers: {
    Authorization: `Bearer ${process.env.API_TOKEN}`,
    Accept: 'application/json',
    'Content-Type': 'application/json',
    'Idempotency-Key': '6b11f086-1e8d-4bb9-a680-1524e8d23b21',
  },
  body: JSON.stringify({
    recipient: '+255712345678',
    message: 'Your verification code is 123456.',
    sender_id: 'MHI',
    client_reference: 'order-1042',
  }),
});
const body = await response.json();
```

Python

```
import os
import requests

response = requests.post(
    f"{os.environ['BASE_URL'].rstrip('/')}/api/v1/messages",
    headers={
        "Authorization": f"Bearer {os.environ['API_TOKEN']}",
        "Accept": "application/json",
        "Idempotency-Key": "6b11f086-1e8d-4bb9-a680-1524e8d23b21",
    },
    json={
        "recipient": "+255712345678",
        "message": "Your verification code is 123456.",
        "sender_id": "MHI",
        "client_reference": "order-1042",
    },
    timeout=15,
)
body = response.json()
```

Java 11+

```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;

String json =
    "{\"recipient\":\"+255712345678\", \""
    + "\"message\":\"Your verification code is 123456.\", \""
    + "\"sender_id\":\"MHI\", \""
    + "\"client_reference\":\"order-1042\"}";
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(System.getenv("BASE_URL") + "/api/v1/messages"))
    .timeout(Duration.ofSeconds(15))
    .header("Authorization", "Bearer " + System.getenv("API_TOKEN"))
    .header("Accept", "application/json")
    .header("Content-Type", "application/json")
    .header("Idempotency-Key", "6b11f086-1e8d-4bb9-a680-1524e8d23b21")
    .POST(HttpRequest.BodyPublishers.ofString(json))
    .build();
HttpResponse<String> response = HttpClient.newHttpClient()
    .send(request, HttpResponse.BodyHandlers.ofString());
```

Go

```
payload := strings.NewReader(`{"recipient":"+255712345678","message":"Your verification code is 123456.", "sender_id":"MHI","client_reference":"order-1042"}`)
ctx, cancel := context.WithTimeout(context.Background(), 15*time.Second)
defer cancel()
req, err := http.NewRequestWithContext(
    ctx,
    http.MethodPost,
    strings.TrimRight(os.Getenv("BASE_URL"), "/")+"/api/v1/messages",
    payload,
)
if err != nil {
    log.Fatal(err)
}
req.Header.Set("Authorization", "Bearer "+os.Getenv("API_TOKEN"))
req.Header.Set("Accept", "application/json")
req.Header.Set("Content-Type", "application/json")
req.Header.Set("Idempotency-Key", "6b11f086-1e8d-4bb9-a680-1524e8d23b21")
response, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
defer response.Body.Close()
```

C# (.NET)

```
using System.Net.Http.Headers;
using System.Net.Http.Json;

using var client = new HttpClient {
    BaseAddress = new Uri(Environment.GetEnvironmentVariable("BASE_URL")!.TrimEnd('/')),
    Timeout = TimeSpan.FromSeconds(15)
};
client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", Environment.GetEnvironmentVariable("API_TOKEN"));
client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("application/json"));
using var request = new HttpRequestMessage(HttpMethod.Post, "/api/v1/messages");
request.Headers.Add(
    "Idempotency-Key",
    "6b11f086-1e8d-4bb9-a680-1524e8d23b21");
request.Content = JsonContent.Create(new {
    recipient = "+255712345678",
    message = "Your verification code is 123456.",
    sender_id = "MHI",
    client_reference = "order-1042"
});
using HttpResponseMessage response = await client.SendAsync(request);
```

Production clients must also validate the HTTP status, validate that the body is a JSON object, capture both X-Request-ID and X-Correlation-ID on successful and error responses, parse the error envelope, and apply the retry policy in [Error Handling](#).

Repository-maintained examples are available under [examples/](#).

15. Security

Protect sensitive data

Never log or expose:

- API credentials or the `Authorization` header;
- webhook secrets or signature headers;
- SMS message content or full request bodies;
- provider credentials or provider message identifiers;
- raw SMPP data, internal database IDs, or stack traces.

Log safe metadata such as a client-generated correlation ID, public `message_id`, stable error code, HTTP status, elapsed time, and retry count. Apply access controls and retention policies to those logs.

Network and application controls

- Require HTTPS and verify certificates.
- Use bounded connection and response timeouts.
- Store secrets in a managed secret store.
- Keep server clocks synchronized for webhook verification.
- Compare webhook signatures using a timing-safe function.
- Use an atomic shared replay store with bounded retention.
- Parse and process webhook data only after signature verification.
- Preserve tenant boundaries in every client-side data lookup.

16. Best Practices

Submission

- Validate recipients and length constraints before sending.
- Use a locally unique `client_reference` to join gateway and business records.
- Generate and persist the idempotency key before the first network attempt.
- Persist the original response and every accepted `message_id`.
- Treat 202 as acceptance for processing, not proof of delivery.

Status and webhooks

- Make terminal-event processing idempotent by `message_id` and status.
- Acknowledge verified webhooks promptly after durable handoff.
- Reconcile queued, submitted, and unknown records periodically.
- Treat terminal statuses as immutable.
- Expect optional fields to be absent.

Resilience

- Set finite connection and read timeouts.
- Use exponential backoff with jitter and an overall retry deadline.
- Apply circuit breaking or load shedding during sustained gateway failure.
- Respect `Retry-After` on rate limiting.
- Alert on sustained authentication, rate-limit, service-unavailable, webhook-verification, or reconciliation failures.

17. Production Readiness

Before enabling production traffic, confirm:

- the production base URL and tenant-scoped credential are provisioned;
- the approved sender or tenant default sender is known;
- secrets are stored outside source control and logs;
- every submission uses a durable, unique idempotency key;
- the agreed idempotency retention period is documented with the operator;
- timeouts, retry limits, and rate-limit handling are configured;
- response and item-level error codes are handled;
- `message_id` and original submission results are persisted;
- the HTTPS webhook is configured and signature verification is tested;
- webhook clock-skew and replay checks are enabled;
- webhook events are durably queued before acknowledgment;
- non-terminal reconciliation is scheduled;
- monitoring, alerting, audit retention, and incident ownership are defined;
- credential and webhook-secret rotation procedures are rehearsed.

For the full release review, use the Go-Live Checklist when that documentation milestone is available.

18. Troubleshooting

Requests return 401 AUTHENTICATION_FAILED

Confirm that the Bearer header is present, the full opaque token is being sent, the token belongs to the current environment, and it has not expired or been revoked. Do not print the token while diagnosing.

A send returns 409 IDEMPOTENCY_CONFLICT

The idempotency key is already bound to different canonical input. Retrieve the locally stored original request and result. Do not retry changed input with that key. Use a new key only for a genuinely new operation.

A bulk request returns 202 with rejected items

This is expected best-effort behavior. Match each result by index, persist accepted `message_id` values, and correct rejected candidates using their item error codes.

A message remains queued, submitted, or unknown

These states are non-terminal. Continue bounded reconciliation through the compact status endpoint and investigate sustained delays with the response `X-Request-ID`, `X-Correlation-ID`, and public `message_id`.

Webhook signature verification fails

Verify that the receiver signs the exact raw bytes, includes the literal `.` between timestamp and body, uses the timestamp header verbatim, calculates lowercase HMAC-SHA256, adds the `v1=` prefix, and checks both active and still-valid previous secrets during the 24-hour rotation overlap.

Webhook events are not retried

RC1 makes exactly one delivery attempt with a 10-second timeout. A failure does not change message persistence. Restore the receiver and reconcile message status through the API.

19. FAQ

Does 202 Accepted mean the SMS was delivered?

No. It means the gateway accepted and evaluated the submission. Use webhooks or the status endpoint to observe a terminal result.

Can I omit sender_id?

No. Every single and bulk message candidate must include an active Sender ID assigned to the authenticated application.

Can I reuse a client_reference as an idempotency key?

They have different roles. `client_reference` is client metadata. Idempotency-Key is the replay authority and is mandatory on submission operations.

Can I retrieve a bulk request by batch_id?

No. `batch_id` is tracking metadata, not a public resource.

Are bulk requests transactional?

No. After top-level validation, candidates are evaluated independently and valid siblings may be accepted when another item is rejected.

Which statuses generate webhooks?

Only delivered, failed, expired, and rejected.

Does the gateway retry failed webhook deliveries?

No. RC1 performs exactly one attempt.

Can I configure several webhook URLs?

No. The contract supports one delivery webhook configuration per tenant.

Is there a webhook delete operation?

No. Use PUT `/api/v1/webhook` to replace the configuration, including its `enabled` value.

20. Appendices

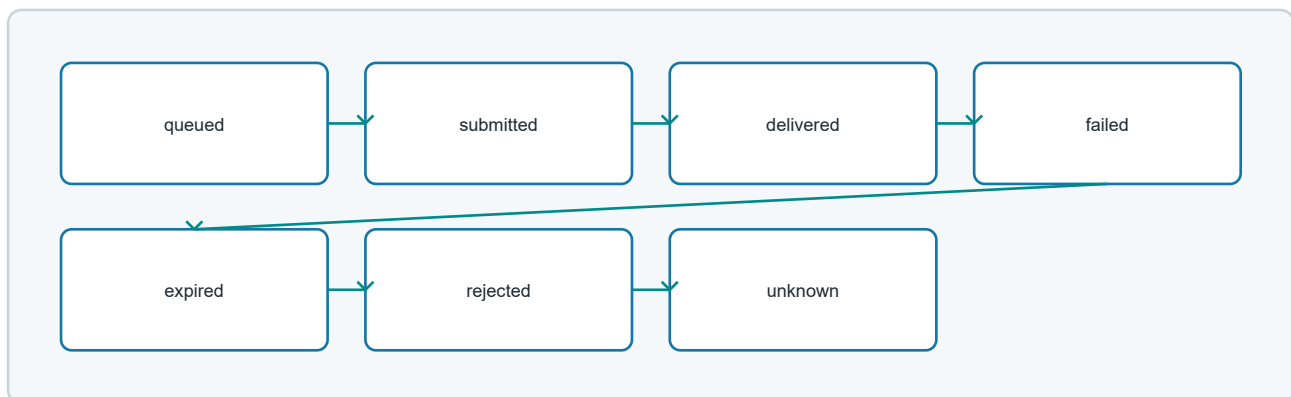
Appendix A: Header reference

Header	Direction	Required	Contract
Authorization	Request	Yes	Bearer <API_TOKEN>
Accept	Request	Recommended	application/json
Content-Type	Request with body	Yes	application/json
Idempotency-Key	Submission request	Yes	Visible ASCII, 1-191 characters
X-Correlation-ID	API request	No	1-100 characters; restricted pattern
X-Request-ID	Every API response	Yes	Unique identifier for the current HTTP request
X-Correlation-ID	Every API response	Yes	Current request correlation identifier
X-Webhook-Timestamp	Webhook request	Yes	Unix seconds
X-Webhook-Signature	Webhook request	Yes	v1= plus 64 lowercase hex characters

Appendix B: Identifier and timestamp formats

Value	Format
message_id	26-character public ULID
batch_id	Lowercase UUIDv4
API timestamps	UTC RFC 3339; persisted resources retain microseconds
Webhook signature timestamp	Unix seconds

Appendix C: Public status transitions



Appendix D: Related documentation

- [OpenAPI 3.1](#) — authoritative machine-readable contract
- [API v1 Contract](#) — authoritative contract narrative
- [Client API Integration](#) — concise integration notes
- [Client Quick Start](#) — existing quick-start sequence
- [API Reference](#) — endpoint reference, when available
- [Webhook Integration Guide](#) — webhook

implementation details, when available

- [API Error Reference](#) — error catalog, when available

Appendix E: Contract limitations and approval items

The authoritative RC1 contract identifies these integration-relevant open items:

- idempotency retention and its migration design;
- active Sender ID inventory and application assignments;
- a zero-downtime API credential rotation runbook;
- legacy-field compatibility and deprecation policy.

Resolve applicable approval items before production launch. Do not infer undocumented behavior.