API Integration Guide

Generated on September 1, 2026

Central Tickets - REST API & Bot Integration Guide

This API is designed specifically for Headless AI Agents and external services to interact with Central Tickets autonomously. The API allows your agents to poll for tickets, reply to users, update state (status, labels, assignments), and receive real-time webhook events.

Authentication

All endpoints require authentication using your Tenant API Credentials. You can generate these in your Tenant Dashboard under **Settings > API**.

Include the following headers in **all** API requests:

```
X-API-Key: your_api_key_here
X-API-Secret: your_api_secret_here
Accept: application/json
```

Idempotency (Preventing Duplicate Actions)

Because network failures can happen, your AI Agent might retry a request. To prevent duplicate replies or state updates, all `POST` requests require an `Idempotency-Key` header.

You should generate a unique UUID for each discrete action your AI takes. If you send multiple requests with the same `Idempotency-Key`, the system will safely return the cached response of the first successful execution.

```
Idempotency-Key: a-unique-uuid-per-action
```

Endpoints

1. Health Check (Verify Connection)

Verify that your API keys are correct and the bot can successfully connect to the tenant's workspace.

Request: `GET /api/v2/bot/health`

Example Response:

```
{
  "success": true,
  "status": "ok",
  "message": "API credentials valid and bot connection successful."
}
```

---

2. Fetch Tickets (Intake & Polling)

Fetch tickets for your tenant. This endpoint is heavily optimized for polling, allowing you to filter for tickets that need an AI response.

Request: `GET /api/v2/bot/tickets`

Query Parameters:

- `status` (string, optional): Filter by status (`open`, `in_progress`, `closed`).
- `unanswered` (boolean, optional): If `true`, returns only tickets where the last reply was from the user (needs attention).
- `updated_since` (string, optional): ISO-8601 timestamp. Returns tickets updated after this time. Useful for periodic polling.

Example Response:

```
{
  "current_page": 1,
  "data": [
    {
      "id": 1234,
      "subject": "How do I reset my password?",
      "status": "open",
      "labels": ["urgent", "billing"],
      "user": {
         "id": 56,
         "name": "Jane Doe"
      },
      "replies": [
         {
            "id": 998,
            "body": "I forgot my password.",
            "is_automated": false,
            "created_at": "2026-08-25T10:00:00.000000Z"
         }
      ]
    }
  ],
  "total": 1
}
```


---

3. Fetch Full Ticket Thread

Fetch a single ticket and its complete conversation thread. This is essential for understanding multi-turn conversations before posting an AI reply.

Request: `GET /api/v2/bot/tickets/{ticket_id}`

Example Response:

```
{
  "success": true,
  "data": {
    "id": 1234,
    "title": "How do I reset my password?",
    "description": "I forgot my password.",
    "status": "open",
    "labels": ["urgent", "billing"],
    "user": {
       "id": 56,
       "name": "Jane Doe"
    },
    "replies": [
       {
          "id": 998,
          "message": "I tried the reset link but it's expired.",
          "is_automated": false,
          "created_at": "2026-08-25T10:00:00.000000Z",
          "user": {
             "id": 56,
             "name": "Jane Doe"
          }
       }
    ]
  }
}
```


---

4. Post an AI Reply

Post a reply to a ticket on behalf of the AI. You can simultaneously change the ticket status and update its AI labels in the same atomic request. *(Note: Replies created via this endpoint are automatically marked as `is_automated: true`)*

Request: `POST /api/v2/bot/tickets/{ticket_id}/replies`

Headers:

- `Idempotency-Key: <uuid>` (Required)

Body:


```
{
  "message": "Hello Jane, you can reset your password by clicking 'Forgot Password' on the logi
  "agent_id": 45,
  "set_status": "in_progress",
  "labels": ["password-reset", "resolved-by-ai"]
}
```

(Only `message` and `agent_id` are required. `set_status` and `labels` are optional).

Agent Identification: The `agent_id` parameter represents the internal user creating the reply. We strongly recommend creating a dedicated "AI Assistant" agent in your workspace or fetching the existing "System Admin" user ID (via the `GET /api/v2/bot/agents` endpoint) to avoid attributing automated actions to human staff. All replies created via this endpoint are securely forced to `is_automated: true` regardless of the assigned `agent_id`.

---

5. Update Ticket State (Categorization & Escalation)

Update the metadata of a ticket without posting a message. This is highly useful for autonomous triage: updating labels, changing status, or escalating to a human agent.

Request: `POST /api/v2/bot/tickets/{ticket_id}/state`

Headers:

- `Idempotency-Key: <uuid>` (Required)

Body:

```
{
  "status": "in_progress",
  "labels": {
    "add": ["needs-human", "complex-issue"],
    "remove": []
  },
  "assigned_to": 45
}
```

(All fields are optional. Pass only what you want to change. Pass `null` to `assigned_to` to unassign the ticket).

---

6. Fetch Available Agents

Retrieve a list of human agents within your tenant. You can use this data to find an agent's `id` for use in the `assigned_to` escalation payload or for the `agent_id` when posting a reply.

Request: `GET /api/v2/bot/agents`

Example Response:

```
[
  {
    "id": 45,
    "name": "John Smith",
    "display_name": "John S. (Support)"
  },
  {
    "id": 82,
    "name": "Sarah Connor",
    "display_name": "Sarah C."
  }
]
```

---

7. Fetch Available Categories

Retrieve a list of active ticket categories within your tenant. This helps your AI accurately categorize and route tickets.

Request: `GET /api/v2/bot/categories`

Example Response:

```
[
  {
    "id": 1,
    "name": "General Inquiry",
    "description": "Basic questions and support"
  },
  {
    "id": 2,
    "name": "Billing Support",
    "description": "Issues with payments and invoices"
  }
]
```

External Integration Endpoints

If you are integrating a custom user dashboard or app, you can use the external endpoint to create tickets on behalf of users and optionally seamlessly SSO them into the ticket interface.

Create Ticket (External System)

Request: `POST /api/v2/tickets`

Headers:

- `X-API-Key: <your_api_key>`
- `X-API-Secret: <your_api_secret>`

Body:

```
{
  "email": "customer@example.com",
  "name": "Customer Name",
  "external_user_id": "cust_123",
  "title": "Need help with billing",
  "description": "I was double charged on my last invoice.",
  "priority": "high"
}
```

Example Response:

```
{
  "success": true,
  "data": {
    "ticket": {
      "id": 154,
      "title": "Need help with billing",
      "status": "open"
    },
    "redirect_url": "https://ticket.yourtenant.com/tickets/154",
    "auth_redirect_url": "https://tickets.flare99.com/api/auth/redirect/yourtenant",
    "auth_payload": {
      "email": "customer@example.com",
      "name": "Customer Name",
      "redirect_url": "https://ticket.yourtenant.com/tickets/154"
    }
  }
}
```

Webhooks (Real-time Event Notifications)

Instead of constantly polling `GET /api/v2/tickets`, you can configure a Webhook in the **Settings > API** dashboard. The system will send a `POST` request to your webhook URL immediately when events occur.

Webhook Security (HMAC)

To verify that the webhook actually came from Central Tickets, every request includes an `X-Ticket-Signature` header.

The signature is generated using an **HMAC SHA-256** hash of the raw JSON request body, signed with your `webhook_secret`.

Verification Example (Node.js):

```
const crypto = require('crypto');

function verifyWebhook(payloadBody, signatureHeader, secret) {
    const hash = crypto.createHmac('sha256', secret)
                       .update(payloadBody)
                       .digest('hex');
    return hash === signatureHeader;
}
```

Webhook Events

1. `ticket.created`

Fired when a brand new ticket is opened.

```
{
  "event": "ticket.created",
  "data": {
    "id": 1234,
    "subject": "Help with login",
    "status": "open",
    "labels": [],
    "user": { ... }
  }
}
```

2. `ticket.updated`

Fired when a ticket's status, priority, or labels change.

```
{
  "event": "ticket.updated",
  "data": {
    "id": 1234,
    "subject": "Help with login",
    "status": "pending",
    "labels": ["login-issue"]
  }
}
```

3. `reply.created`

Fired when a user or human agent replies to a ticket. (Internal agent notes do not trigger this webhook).

```
{
  "event": "reply.created",
  "data": {
    "ticket_id": 1234,
    "reply": {
      "id": 999,
      "body": "Thank you, that worked!",
      "is_automated": false,
      "user_id": 56
    }
  }
}
```