# Implementing a New WMS API Endpoint

This guide explains the standard process for adding an endpoint to the existing WMS API. It is written for a developer who is new to this codebase.

# Purpose

The WMS API uses one central entry point. API requests are sent to the API index file. That file determines the request path and method, then sends the request to the correct endpoint handler. Shared response, authentication, permission, URL, and request-reading helpers are kept in the API bootstrap file.

Project Forge is in Implementation Mode. Preserve the current API structure, authentication, response format, and route style unless the request specifically requires an architectural change.

# Files Used for an Endpoint

| File | Purpose |
| --- | --- |
| httpdocs/api/index.php | Contains endpoint handlers, the route list, and request dispatching. |
| httpdocs/api/bootstrap.php | Contains common JSON response, authentication, permission, URL, and request helpers. |
| API rewrite configuration file | Forwards API requests to the central API index file. It normally does not change when adding a route. |

# Plan the Endpoint First

Before editing code, define the endpoint in operational terms.

- What WMS record or warehouse action does it support?
- Who may use it?

- Is it read-only, or does it change WMS data?
- What information is required from the caller?
- What result should be returned?
- How will access be limited to the correct client?

Use business-oriented route names that match existing routes. Orders, shipments, and inventory are examples. Review the existing API routes before selecting a new name.

# Required Security Rules

## Use the Existing Authentication Flow

The current WMS API uses the authenticated WMS session. It supports the WMS and the client portal. It is not currently designed as an external integration API with separate API keys or OAuth access.

Do not bypass the existing authentication or permission flow when adding an endpoint.

## Protect Client Data

A client user may only view or change records that belong to that client. This rule applies to every endpoint that reads or changes client data.

Use the authenticated user context to determine the permitted client. Do not trust a client identifier supplied in the request by itself. Enforce ownership in the record lookup or database query.

## Validate Input

Validate required path values, query values, and request fields before reading or changing data. Reject missing, malformed, or unsupported values through the existing API error response helper.

## Use the Standard Response Format

Every endpoint must use the existing API success and error response helpers. Do not create custom JSON formats for a single endpoint and do not print JSON directly from the handler.

# Implementation Procedure

## 1. Find a Similar Endpoint

Open the API index file and locate the closest existing endpoint. Follow its approach for permissions, client scoping, response fields, error handling, and route dispatching.

## 2. Add a Handler

Add the business-logic handler in the handler section of the API index file. Use a clear name that identifies the resource and action.

A handler should follow this order:

1. Read and validate the required input.
2. Confirm the authenticated user has permission for the action.
3. Determine the permitted client from the authenticated user.
4. Load the requested record with client ownership enforced when applicable.
5. Return a standard error if the record is missing or unavailable.
6. Perform the allowed read or write action.
7. Return the result through the standard success helper.

## 3. Register the Route

Add the route to the API route list in the API index file. This keeps the available routes discoverable and consistent with the current dispatcher.

## 4. Connect the Dispatcher

In the API route-dispatch section, add the path and request-method match that calls the new handler.

- Use a read request branch for a read-only endpoint.
- Use a create or action request branch for an endpoint that changes data.
- Keep related routes inside the existing parent resource section, such as orders or shipments.
- Use the existing method-not-allowed response when a route does not support the requested method.

# 5. Confirm Routing

A normal route addition should not require a change to the API rewrite file. If the new API URL returns a server-level not-found page instead of an API JSON response, verify that the rewrite rule still forwards the request to the API index file.

# Route Examples

| Purpose | Route Shape | Request Method |
|---|---|---|
| List records | Orders collection | GET |
| Read one record | Orders collection followed by an order identifier | GET |
| Perform a controlled action | Order record followed by an action name | POST |

These are route-shape examples only. Match the actual route conventions already used in the API index file.

# Testing Before Release

1. Run the PHP syntax check on each PHP file that changed.
2. Test the normal successful request while logged in as an authorized WMS user.
3. Test a request with missing required information.
4. Test an invalid or non-existent record identifier.
5. Test an unsupported request method.
6. Test with a client-scoped user and confirm that another client record cannot be viewed or changed.
7. For a write endpoint, confirm the WMS record changes once and the response accurately reflects the completed action.

Because the API uses the logged-in WMS session, use a logged-in browser session or an approved session-aware test process. Do not place credentials, session values, tokens, or client data in source code, screenshots, documentation, or Git commits.

# Endpoint Completion Checklist

- The endpoint purpose, route, method, permission, and response fields are defined.
- A handler was added using the existing WMS pattern.
- Input validation uses the standard API error response.
- Client ownership is enforced for client data.
- The route is listed and connected in the API index file.
- Success and error responses use the existing API helpers.
- PHP syntax checks pass.
- Success, validation, not-found, permission, and client-isolation tests pass.
- The endpoint is documented before release.

# Documentation Required for Every Endpoint

When a new endpoint is added, document the following in the WMS developer documentation:

- Endpoint name and purpose
- Request method and route
- Required authentication and permissions
- Client-scoping behavior
- Required path, query, and request values
- Success response fields
- Expected error responses
- Test procedure

Keeping this information current allows the next developer to extend the WMS safely without changing its established architecture.

---

Created 2026-07-24 15:51:26 UTC by Andrew
Updated 2026-07-24 16:15:00 UTC by Andrew