Consultingwerk
software architecture and development

REST API Versioning: Concepts and Samples

Mike Fechner
Consultingwerk

The full stack modernization framework
SmartComponent Library
by Consultingwerk

Mike Fechner



- 36 years of experience in Progress (from Version 5 to OpenEdge 13)
- Active member of the OpenEdge community and speaker at international conferences
- Expert in software architecture, object-oriented design, and web technologies

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Consultingwerk
software architecture and development

Our Focus: Modernizing OpenEdge Applications



Modernization of Legacy OpenEdge Applications



Deep Technical Expertise



Global IT Partner with Local Presence



More than Consulting – We Deliver Tools & Solutions

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Sample Code on GitHub

github.com/consultingwerk/
rest-api-versioning-samples

What's inside

- WebHandlers for URL path and media type versioning
- One typed Order model, a reader and writer per version
- OpenAPI documents, ABLUnit tests, curl demo script

Scan for the samples

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Agenda

Why version at all?
APIs are contracts between business partners

Where to put the version
URL, query, header or media type — and what the RFCs say

PASOE WebHandlers in five minutes
The building block for all samples

Implementing versioning
URL path and media types with plain WebHandlers

One business logic, many versions
One typed model, a reader and writer per version

Documenting versions
Swagger / OpenAPI and API versions

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



Why Version at All?

APIs are contracts — and contracts evolve

“It Was Only a Small Change …”

Friday: the API returns

```
{
  "CustNum": 1,
  ...
```



Monday: after a “small cleanup”

```
{
  "customerNumber": 1,
  ...
```

Cleaner for us —

and every partner integration is broken on Monday morning

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

An API Is a Contract Between Business Partners

Consumers you don't control

- Partners and suppliers
- Customers' ERP integrations
- Installed mobile apps
- B2B portals

Their schedule, not yours

- They release when it suits them
- Some integrations run unchanged for years

The contract

- URLs and methods
- Field names and data types
- Semantics
- Status and error codes

Change the contract, and you change it for everyone at once

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

What Breaks the Contract — and What Doesn't

Breaking

- Removing or renaming a field or endpoint
- Changing a data type or a meaning
- Optional parameter becomes required
- Restructuring payloads (flat → nested)

Non-breaking — for tolerant readers

- Adding an endpoint
- Adding an optional request parameter
- Adding a response field

Tolerant reader: clients ignore what they don't know — then additive changes are safe

A version is the last resort — not a ritual

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

The Dilemma: Freeze or Break?

Freeze the API

- Technical debt piles up
- The domain model can’t evolve

Change the API

- Partner integrations break
- Trust is gone

Version the API

- Keep moving forward
- Old clients keep working
- Retire old versions on a schedule

API versioning: progress without breaking your partners

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Not Everybody Agrees — Yet Everybody Versions

> ***"The reason to make a real REST API is to get evolvability … a "v1" is a middle finger to your API customers, indicating RPC/HTTP (not REST)"***
>
> Roy T. Fielding, who defined REST — September 2013

Above all: Atlassian

Jira: `/rest/api/2` and `/rest/api/3`

Confluence: `/wiki/rest/api` → `/wiki/api/v2`

A great (bad) example of APIs that change relatively frequently

GitHub
`X-GitHub-Api-Version`

Stripe
`Stripe-Version`

Microsoft Graph
`/v1.0`

Azure
`api-version`

Salesforce
`/vXX.X/`

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Our Use Case: the Sports2000 Order API

Order header (table Order) + customer address

```
GET     /Orders
GET     /Orders/{Ordernum}
POST    /Orders
PUT     /Orders/{Ordernum}
PATCH   /Orders/{Ordernum}
DELETE → not allowed
         in any version
```

Plain WebHandlers, no framework magic — the pattern works in any ABL application

v0 — Switched off
Our first attempt — answers with an error only

v1 — Deprecated — still works
Flat: all Order fields + customer address

v2 — Current
Customer data nested, WarehouseNum dropped

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

v1 vs. v2 — Two Breaking Changes

v1 — flat

```
{
  "Ordernum": 6,
  "CustNum": 1,
  "Name": "Example Sports",
  "Address": "1 Main Street",
  "City": "Burlington",
  "PostalCode": "01730",
  "OrderDate": "2026-09-14",
  "WarehouseNum": 1,
  ...
}
```

v2 — nested, no WarehouseNum

```
{
  "Ordernum": 6,
  "Customer": {
    "CustNum": 1,
    "Name": "Example Sports",
    "Address": "1 Main Street",
    "City": "Burlington",
    "PostalCode": "01730" },
  "OrderDate": "2026-09-14",
  ...
}
```

(1) customer data moves into `Customer` · (2) `WarehouseNum` is gone · in v1: `"Name"` — whose name?

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



Where to Put the Version

URL, query, header or media type — and what the RFCs say

Four Places for a Version



URL path

```
GET /v2/Orders/6
```



Query parameter

```
GET /Orders/6?api-version=2
```

Microsoft Azure requires `api-version` on every request



Custom header

```
X-Api-Version: 2
```



Media type

```
Accept:
application/vnd.sports2000.order.v2+json
```

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

URL Path Versioning

```
GET /v1/Orders/6          GET /v2/Orders/6
```

Pros

- Visible, easy to test in a browser
- Easy to route, cache-friendly
- Trivial to document in Swagger

Cons

- Same order, new URL
- Versions the whole API at once
- Stored URLs carry the version forever

Major versions only: `v1`, `v2` — never `v1.3.7`. Additive changes stay in the version.

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Custom Headers and Date-Based Versions

GitHub

- Header `X-GitHub-Api-Version`
- Without it: version 2022-11-28
- Current: 2026-03-10 and 2022-11-28
- Previous version supported at least 24 months after a new one

Stripe

- Header `Stripe-Version`
- e.g. 2017-05-25
- Date-based version names
- An account is pinned to the latest version on its first API call
- Upgrade when you are ready

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Media Types 101

`application`	`/vnd.sports2000.order.v2`	`+json`	`;version=2`
type	subtype	suffix	parameter (optional)

Content-Type

- Describes what I **send**
- Request bodies: POST, PUT, PATCH
- Response: what the server actually sent

Accept

- Describes what I **want back**
- Several types possible, weighted with `q`
- The server picks — content negotiation

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

RFC 6838 — Registration Trees

Standards tree

No prefix, e.g. `application/json`

IETF / IESG registration

Vendor tree — vnd.

“Used for media types associated with publicly available products” (§3.2)

e.g. `application/vnd.github+json`

Personal tree — prs.

Experimental or non-commercial types (§3.3)

Unregistered — x.

Private, local use only (§3.4)

Cannot be registered

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

RFC 6839 — Structured Syntax Suffixes

> *“The suffix "+json" MAY be used with any media type whose representation follows that established for "application/json".”*

Why it matters

- Generic tools still know it’s JSON
- Browsers, proxies, JSON viewers, Swagger UI

Also registered

- `+xml`, `+zip`, `+der`, `+ber`, `+fastinfoset`, `+wbxml`
- Rules for suffixes: RFC 6838 §4.2.8

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Two Ways to Version a Media Type



Version in the subtype

`application/vnd.sports2000.order.v2+json`

Each version is its own media type

Our samples use this style



Version as a parameter

`application/vnd.sports2000.order+json;version=2`

One media type, versioned by parameter

Parameters must be documented (RFC 6838 §4.3)

Both are valid — pick one and use it consistently across your API

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

HTTP Content Negotiation (RFC 9110)

```
Accept: application/vnd.sports2000.order.v2+json,
        application/vnd.sports2000.order.v1+json;q=0.5
```

406

Not Acceptable

I can't produce anything you accept

415

Unsupported Media Type

I can't read what you sent

Vary

Vary: Accept

Tells caches and proxies the response depends on Accept

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Vary — Why Caches Need to Know

Response from the server

```
HTTP/1.1 200 OK
Content-Type: application/
  vnd.sports2000.order.v2+json
Vary: Accept
Cache-Control: max-age=60
```

An HTTP response header

Lists the **request** headers that chose this representation (RFC 9110 §12.5.5)

Without Vary

A cache (browser, proxy, CDN) keys the response by URL only — client A asks for v1, client B for v2 and gets A's v1 from the cache

With Vary: Accept

The cache stores one variant per `Accept` value — every client gets the version it asked for

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Case Study: Jira Cloud REST API v2 and v3



Cloud Jira Cloud platform / Reference / REST API v3

About Postman Collection OpenAPI

The Jira REST API enables you to interact with Jira programmatically. Use this API to build apps, script interactions with Jira, or develop any other type of integration. This page documents the REST resources available in Jira Cloud, including the HTTP response codes and example requests and responses.

Version

This documentation is for **version 3** of the Jira Cloud platform REST API, which is the latest version. Version 2 and version 3 of the API offer the same collection of operations. However, version 3 provides support for the Atlassian Document Format (ADF) in:

- `body` in comments, including where comments are used in issue, issue link, and transition resources.
- `comment` in worklogs.
- `description` and `environment` fields in issues.
- `textarea` type custom fields (multi-line text fields) in issues. Single line custom fields (`textfield`) accept a string and don't handle Atlassian Document Format content.

Two versions side by side

- `/rest/api/2/…` and `/rest/api/3/…`
- “Version 2 and version 3 of the API offer the same collection of operations. However, version 3 provides support for the Atlassian Document Format (ADF)”

Same data, new representation:

plain text in v2 → ADF JSON in v3 — exactly our v1 → v2

Screenshot: Atlassian Developer docs, Jira Cloud REST API v3, section “Version” → open live

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Jira: Where a String Became an ADF Object

v2 — wiki markup in a string

```
PUT /rest/api/2/issue/DEMO-1
{ "fields": {
    "description":
      "Order *6* is late" } }
```



Affected fields

- Description, environment
- Comment bodies, worklog comments
- Multi-line text custom fields

v3 — Atlassian Document Format (ADF)

```
PUT /rest/api/3/issue/DEMO-1
{ "fields": {
  "description": {
    "type": "doc", "version": 1,
    "content": [ {
      "type": "paragraph",
      "content": [
        { "type": "text", "text": "Order " },
        { "type": "text", "text": "6",
          "marks": [ { "type": "strong" } ] },
        { "type": "text", "text": " is late" }
      ] } ] } } }
```

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Case Study: Jira Cloud — Two Deprecations

Privacy: `username` / `userKey` → `accountId`

migration guide 2018

Oct 2018 — Deprecation period begins, migration guide

Apr 2019 — username and userKey removed from the API

Search: `/rest/api/3/search` → `/rest/api/3/search/jql`

Oct 2024 — Deprecation announced in the changelog

May 2025 — Planned removal — still working

Aug–Oct 2025 — Progressive shutdown → 410 Gone

No new major version

Both changes broke v2 and v3 clients alike

Date, replacement, migration guide — and in the end: 410 Gone

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Comparison

Approach	Visible in URL	Browser test	Caching	OpenAPI	Scope	Examples
URL path	✓	✓	✓ distinct URL	One doc per version	Whole API	Jira, Confluence, MS Graph
Query parameter	✓	✓	✓ distinct URL	Parameter	Whole API	Azure
Custom header	✗	✗	Needs Vary	Header param	Whole API	GitHub, Stripe
Media type	✗	✗	Needs Vary: Accept	Content per media type	Per resource	GitHub vnd.github+json

Public API, unknown consumers → **URL path** · per-resource evolution → **media types**

Either way: define a deprecation policy up front — how many versions, how long

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

v1 Is Deprecated — Tell Your Clients

HTTP response headers on every v1 response

```
Deprecation: @1782864000
Sunset: Wed, 30 Jun 2027 23:59:59 GMT
Link: <https://docs.example.com/api/orders/migrate-v2>; rel="deprecation"; type="text/html"
```

Deprecation

- RFC 9745 (2025)
- `@` + seconds since 1970-01-01 UTC
- `@1782864000` = 1 July 2026

Sunset

- RFC 8594 (2019)
- HTTP-date, like `Date` or `Expires`
- After it, v1 may stop working

Runtime, not spec

- Sent by the server with each response
- OpenAPI: only `deprecated: true` — no date

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

v0 Is Switched Off — Answer with a Proper Error

```
HTTP/1.1 410 Gone
Content-Type: application/problem+json
Link: </v2/Orders/6>; rel="successor-version"

{
  "type": "https://docs.example.com/
           problems/api-version-switched-off",
  "title": "API version switched off",
  "status": 410,
  "detail": "Version v0 of the Orders API
             was switched off on 30 June 2026.
             Please use v2."
}
```

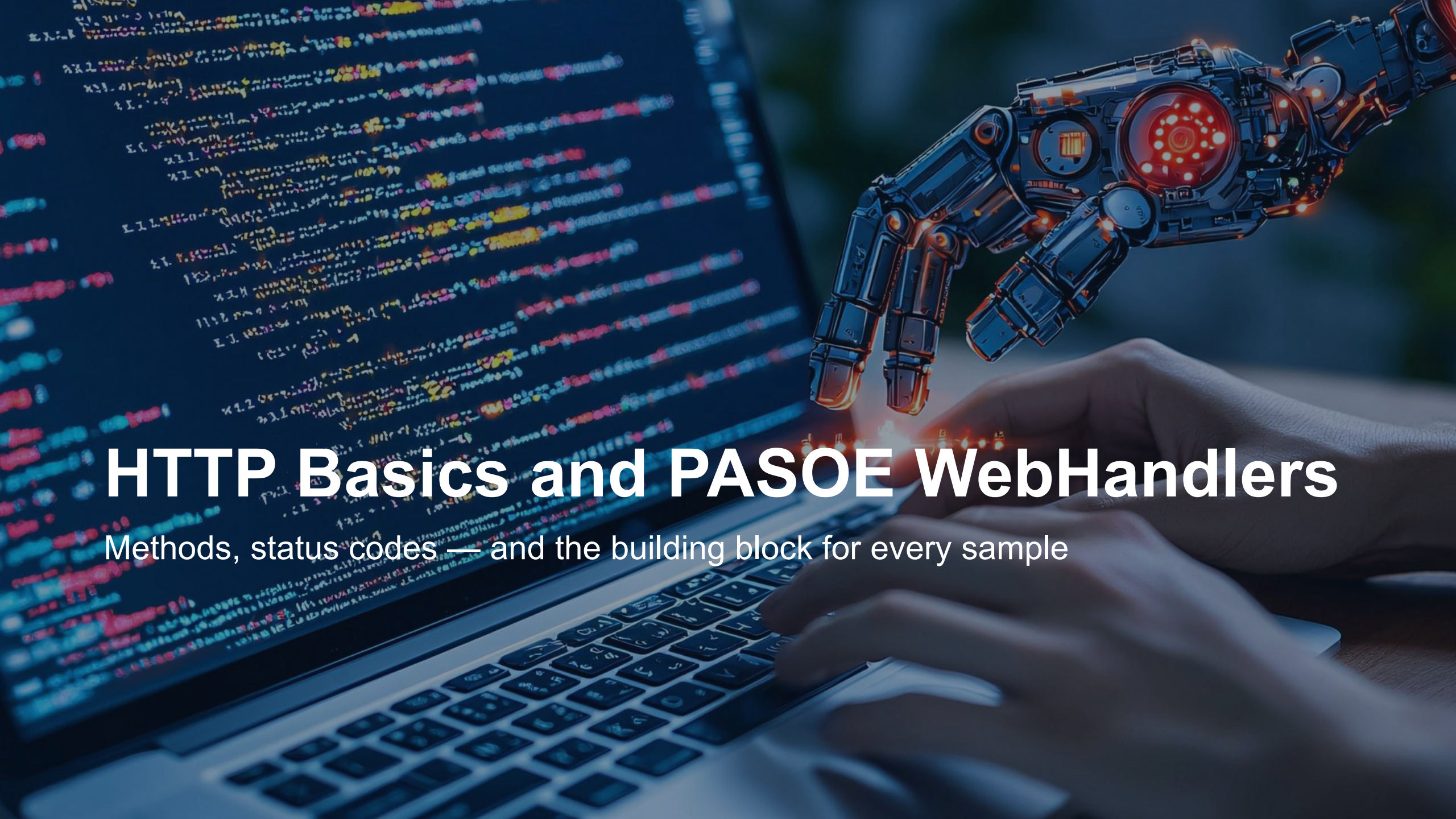
410, not 404

The client learns that the version existed — and where to move

Problem details

- RFC 9457 — successor of RFC 7807
- No code behind v0 any more

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



HTTP Basics and PASOE WebHandlers

Methods, status codes — and the building block for every sample

HTTP Request Methods

Method	Meaning	Safe	Idempotent	Our Order API
GET	Read a resource	✓	✓	`GET /Orders/6`
POST	Create — the server assigns the URI	✗	✗	`POST /Orders`
PUT	Replace the whole resource	✗	✓	`PUT /Orders/6`
PATCH	Change only the fields sent (RFC 5789)	✗	✓ ours — not by spec	`PATCH /Orders/6`
DELETE	Remove the resource	✗	✓	405 in every version

Safe: changes nothing on the server. **Idempotent:** sending it twice has the same effect as once — safe to retry.
PATCH may contain relative operations (“append”, “+1”) — with absolute values, as in our API, it is idempotent.

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

HTTP Status Codes

2xx — it worked

200 OK
201 Created + `Location`
204 No Content

5xx — our fault

500 Internal Server Error
501 Not Implemented
502 / 503 / 504 gateway, unavailable

4xx — the client's fault

400 Bad Request	**409** Conflict
401 Unauthorized	**410** Gone
403 Forbidden	**412** Precondition Failed
404 Not Found	**415** Unsupported Media Type
405 Method Not Allowed	**422** Unprocessable Content
406 Not Acceptable	**429** Too Many Requests

Highlighted: used in our samples

All codes: en.wikipedia.org/wiki/List_of_HTTP_status_codes

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

What Is a WebHandler?



One ABL class per endpoint

- Runs in PASOE's WEB transport
- Inherits `WebHandler` from `OpenEdge.Web`
- One method per HTTP verb
- Full control: headers, status codes, body

Skeleton

```
class OrderWebHandler
    inherits OpenEdge.Web.WebHandler:

  method override protected integer
      HandleGet (poRequest as IWebRequest):
      ...
  end method.

  /* HandlePost, HandlePut, HandlePatch, */
  /* HandleDelete, HandleNotAllowedMethod, */
  /* HandleNotImplemented ...             */
end class.
```

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Mapping URIs to Handlers

ROOT.handlers

```
{ "class": "<handler class>",
  "uri": "/{Version}/Orders/{Ordernum}",
  "enabled": true }
```



Path parameters

- URI templates with `{name}`
- Read in ABL:

```
cVersion = poRequest:GetPathParameter ("Version").
```

Order matters: the most specific URI template wins — register specific before generic

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

A Minimal WebHandler: the Class

```
using OpenEdge.Net.HTTP.StatusCodeEnum.
using OpenEdge.Web.*.
using Progress.Json.ObjectModel.JsonObject.

class Demo.SimpleOrderHandler inherits WebHandler:

  method override protected integer
      HandleNotAllowedMethod (poRequest as IWebRequest):
    return integer (StatusCodeEnum:MethodNotAllowed).
  end method.

  method override protected integer
      HandleNotImplemented (poRequest as IWebRequest):
    return integer (StatusCodeEnum:NotImplemented).
  end method.

  /* HandleGet and HandlePut: next slides */
end class.
```

Two must-haves

`WebHandler` is abstract: these two methods are required

The return value

- `0` = I wrote the response
- Any other number: PASOE sends that status

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

A Minimal WebHandler: GET with a Path Parameter

```
method override protected integer HandleGet
    (poRequest as IWebRequest):
  var integer iOrdernum.
  var JsonObject oOrder = new JsonObject ().
  var WebResponse oResponse = new WebResponse ().
  var WebResponseWriter oWriter.

  iOrdernum = integer (poRequest:GetPathParameter ("Ordernum")).
  oOrder:Add ("Ordernum", iOrdernum).
  oOrder:Add ("Carrier", "FedEx").  /* no DB access here */

  assign oResponse:StatusCode  = integer (StatusCodeEnum:OK)
         oResponse:ContentType = "application/json"
         oResponse:Entity      = oOrder.
  oWriter = new WebResponseWriter (oResponse).
  oWriter:Open ().
  oWriter:Close ().
  return 0.
end method.
```

Request → response

```
GET /Orders/6

{ "Ordernum": 6,
  "Carrier": "FedEx" }
```

Three steps

- Read the path parameter
- Build a `JsonObject`
- `WebResponse` + writer

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

A Minimal WebHandler: PUT with a JSON Body

```
method override protected integer HandlePut
    (poRequest as IWebRequest):
  var character cContentType, cAccept.
  var JsonObject oBody.
  var WebResponse oResponse = new WebResponse ().
  var WebResponseWriter oWriter.

  cContentType = poRequest:ContentType.
  if poRequest:HasHeader ("Accept") then
    cAccept = poRequest:GetHeader ("Accept"):Value.
  oBody = cast (poRequest:Entity, JsonObject).
  /* validate and save oBody: out of scope */
  assign oResponse:StatusCode  = integer (StatusCodeEnum:OK)
         oResponse:ContentType = "application/json"
         oResponse:Entity      = oBody.
  oWriter = new WebResponseWriter (oResponse).
  oWriter:Open ().  oWriter:Close ().
  return 0.
end method.
```

Two headers

- `Content-Type`: what came in
- `Accept`: what to send back

Entity

- PASOE parses JSON into a `JsonObject`
- Also for `vnd.…+json`

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Our Base Class: SmartWebHandler



What it does

- Abstract base class, inherits `OpenEdge.Web.WebHandler`
- `HandleError` turns exceptions into HTTP responses
- Exceptions carry their status code: `IHttp400Error` ... `IHttp511Error`

Pluggable error handler

- `ISmartWebErrorHandler` service
- `Rfc7807WebErrorHandler` writes problem details
- All we take from the framework — the rest is plain ABL

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



Implementing Versioning with WebHandlers

URL path and media types — the Order API in code

URL Versioning, Option A: Fixed URIs

ROOT.handlers — one set per version

```
{ "uri": "/v1/Orders",              ... },
{ "uri": "/v1/Orders/{Ordernum}", ... },
{ "uri": "/v2/Orders",              ... },
{ "uri": "/v2/Orders/{Ordernum}", ... },
{ "uri": "/v0/Orders",              ... }
```



Explicit

Easy to see what exists



Grows

Every version adds handlers

Each registration points to its own small handler class

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Option B: One WebHandler, Version as Path Parameter

Sketch — real code from the sample app follows

```
method override protected integer HandleGet
    (poRequest as IWebRequest):

  case poRequest:GetPathParameter ("Version"):
    when "v0" then
      undo, throw new VersionSwitchedOffException ().
    when "v1" then return this-object:Send (
      OrderV1Writer:Write (OrderService:Get (iOrdernum))).
    when "v2" then return this-object:Send (
      OrderV2Writer:Write (OrderService:Get (iOrdernum))).
    otherwise
      undo, throw new NotFoundException ().
  end case.
```



Service interface

Every version the API ever had — in one place



Your code

Version handling is now your responsibility

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Media Types: One Handler Negotiates

Accept / Content-Type → version

```
…order.v2+json  or  application/json → v2
…order.v1+json                       → v1
…order.v0+json                       → 410
anything else                        → 406
unreadable Content-Type              → 415
```



Then

- Response `Content-Type` = negotiated type
- Always `Vary: Accept`
- PATCH uses the same versioned type

Negotiation lives in one small class — the handler just asks it

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Not Everything Is Versioned: DELETE

```
HTTP/1.1 405 Method Not Allowed
Allow: GET, PUT, PATCH
Content-Type: application/problem+json
```

Orders are never deleted via the API

- In no version
- Thrown as `HttpMethodNotAllowedException`
- `SmartWebHandler` does the rest

405, not 501

- 405: the resource doesn't allow this method — RFC 9110 requires an `Allow` header
- 501 Not Implemented: the server doesn't know the method at all

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Live Demo: One Order, Three Versions

```
curl -i https://<host>/…
GET    /web/v2/Orders/6              200  nested
GET    /web/v1/Orders/6              200  flat + Deprecation
GET    /web/v0/Orders/6              410  Gone
GET    /web/Orders/6
       Accept: …order.v1+json        200  flat
PATCH  /web/v2/Orders/6
       {"Carrier": "FedEx"}          200  one field
PATCH  /web/v2/Orders/6
       {"WarehouseNum": 2}           400
DELETE /web/v2/Orders/6              405
GET    Accept: text/csv              406
```



Same order — three versions, two styles

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

The Catch



v1 code

- Handler
- Data access
- Business logic



v2 code

- Handler
- Data access
- Business logic — again



And v3 …

- Another copy
- Another place for the same bug

Works on day one — and from then on

every bug fix and every new rule has to be made twice

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



One Business Logic, Many API Versions

One typed model — a reader and a writer per version

The Question

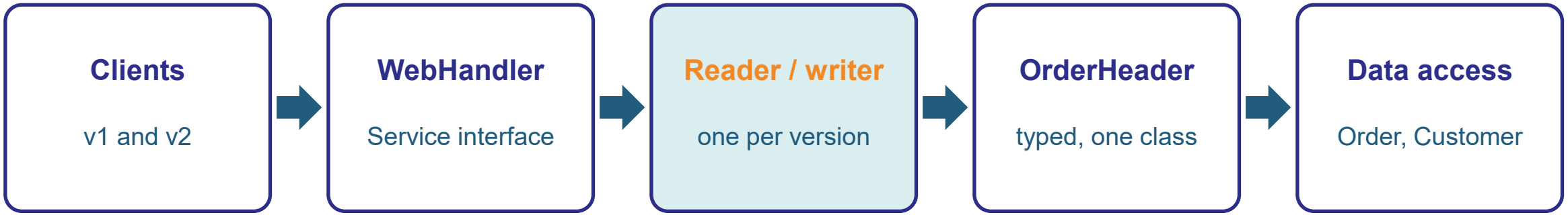
An interface layer keeps older API versions alive — and one implementation holds the business logic. How do I map between them?

The goal

Maintain the implementation **once** — in the shape of the **newest** API version

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

The Architecture



JSON ↔ objects at the edge

- Readers merge a request into `OrderHeader`
- Writers render it in the requested version

One interface per version

- `OrderHeader` implements `IOrderHeaderV1` and `IOrderHeaderV2`
- v3 adds `IOrderHeaderV3` — v1 code untouched

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Prior Art: Stripe's Version Change Modules

> ***"Version change modules keep older API versions abstracted out of core code paths."***

How Stripe does it (2017)

- Each breaking change is wrapped in a version change module
- Responses start from the current version and walk backward through the modules until they reach the client's version
- Engineers build for the latest version only

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Use Case 1: New Fields

Responses — harmless

- Old version: drop the new fields
- … or just leave them: tolerant reader

Requests — careful

- New mandatory fields: an old client can't send them
- The v1 reader sets default values

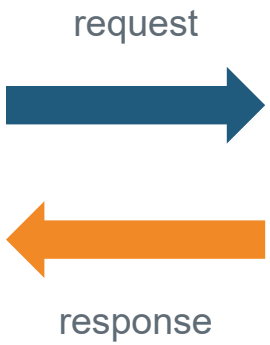
"Adding a field" is only harmless in one direction

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Use Case 2: Flat Becomes Nested

v1

```
"CustNum": 1,
"Name": "Example Sports",
"City": "Burlington",
...
```



v2

```
"Customer": {
  "CustNum": 1,
  "Name": "Example Sports",
  "City": "Burlington", ... }
```

Wrap and lift

- Request v1 → v2: wrap into `Customer`
- Response v2 → v1: lift back to the top

Address fields are read-only

They come from the `Customer` table — on writes only `CustNum` counts

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Use Case 3: a Field Is Removed

```
PATCH /v2/Orders/6
{ "WarehouseNum": 2 }

400 Bad Request
{ "title": "Unknown field",
  "detail": "WarehouseNum is not
             part of v2" }
```

Why reject, not ignore?

- The client believes the value was stored
- Silent data loss is worse than an error

Strict in, tolerant out

Clients tolerate unknown fields in responses — the server rejects them in requests

Contract ≠ database

v1 still has `WarehouseNum` — its reader and writer keep it

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Use Case 3 in Code: Contract ≠ Database

Pseudo code — the real code comes from the sample app

```
/* v2 write: the reader rejects the field */
oOrder = OrderService:Get (iOrdernum).
OrderV2Reader:Merge (oJson, oOrder).
    /* "WarehouseNum" → 400 */
OrderService:Update (oOrder).

/* v1 read: the adapter still maps it */
oOrder = OrderService:Get (iOrdernum).
oJson  = OrderV1Writer:Write (oOrder).
    /* flat, incl. WarehouseNum */

/* v1 write: allowed to change it */
OrderV1Reader:Merge (oJson, oOrder).
OrderService:Update (oOrder).
```



Internal model

Newest shape plus every field an older version still needs



Drop the column …

… only after the last version using it is switched off

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Mapping in ABL — Two Options

JSON level

- `Progress.Json.ObjectModel`
- Move properties into / out of a nested `JsonObject`
- Generic and fast

Temp-table level

- A v1 temp-table filled from the v2 structure
- `buffer-copy`, explicit assignments
- Type-safe

At the edge, JSON wins for PATCH: it can tell “field missing” from “field is null”

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Business Logic Speaks Types, Not JSON

Class-based: one class, two contracts (sketch)

```
interface IOrderHeaderV1:
  define property CustNum      as integer get. set.
  define property WarehouseNum as integer get. set.
  ...
end interface.

interface IOrderHeaderV2:
  define property Customer as ICustomerV2 get. set.
  ...
end interface.

class OrderHeader
    implements IOrderHeaderV1, IOrderHeaderV2:
  /* CustNum delegates to Customer:CustNum */
```

JSON only at the edge

- Readers and writers handle JSON
- Business logic gets strong types: temp-tables, ProDataSets or classes
- **Next format — XML revival, YAML, TOML? Just another reader/writer**

PATCH, typed

Track what was set: before-image or assigned list

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Writes Are the Hard Part

Read-merge-write

Fields the v1 client doesn't know must survive a v1 PUT

PATCH: only what was sent

Map only the fields present:

`{"CustNum": 2}` → `{"Customer": {"CustNum": 2}}`

Errors in the client's language

Validation errors must point to v1 field names

ETags across versions

`If-Match` must still work — same record, different representations

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

ETag and If-Match: Optimistic Locking over HTTP

```
GET /v2/Orders/6
→ 200 OK
  ETag: "a1b2c3"

PUT /v2/Orders/6
If-Match: "a1b2c3"
→ 200 OK
  ETag: "d4e5f6"

PUT /v2/Orders/6        (stale copy)
If-Match: "a1b2c3"
→ 412 Precondition Failed
```

ETag — response header

Version stamp of what the client got

If-Match — request header

Write only if the stamp still matches

Across versions

v1 and v2: different ETags, same record check

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Testing Every Version



Contract tests with ABLUnit

- Golden JSON files per version
- Request in, response compared
- Old versions are frozen — their tests never change
- Part of our ~29,000 unit tests

Golden files (sketch)

```
UnitTests/…/RestApiVersioning/
  v1/
    get-order-6.json
    put-order-6.json
  v2/
    get-order-6.json
    patch-warehousenum.json
  v0/
    any-request-410.json
```

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



Documenting Versions with OpenAPI

Swagger knows media types

One Spec per Version — or One for All?

URL versioning

- One OpenAPI document per version
- `/v1/openapi.json`, `/v2/openapi.json`
- Like Atlassian: separate specs for Jira v2 and v3

Media type versioning

- One path, several media types
- One document shows all versions side by side

Code generators usually cope better with one document per version

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Media Types Are First-Class in OpenAPI

openapi.yaml (simplified)

```
/Orders/{Ordernum}:
  get:
    responses:
      "200":
        content:
          application/vnd.sports2000.order.v2+json:
            schema: { $ref: "#/…/OrderV2" }
          application/vnd.sports2000.order.v1+json:
            schema: { $ref: "#/…/OrderV1" }
      "406": { description: Not Acceptable }
      "410": { description: Gone }
```



Per media type

- A `content` map per response and request body
- Own schema per version
- No extra properties allowed → the 400 is documented
- Address fields read-only

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Our Order API in Swagger UI

Open the document → Pick a media type → See its schema → Try it out



Response

- Media type drop-down: v1 or v2
- v1 marked deprecated
- 400, 405, 406, 410, 415 documented



Request body

- Same choice for POST, PUT and PATCH
- PATCH: same schema, no required fields

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Marking the Past

OpenAPI 3.x

```
application/vnd.sports2000.order.v1+json:
  schema:
    $ref: "#/components/schemas/OrderV1"
    deprecated: true
```



At design time

- `deprecated: true` on operations and schemas
- v1 in our example



At runtime

`Deprecation` and `Sunset` headers on every v1 response — document them as response `headers`; v0 with its 410

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Generated Clients

Watch out

- Many generators expect `application/json`
- Vendor media types are often handled poorly

Before you promise it to partners

- Generate C#, TypeScript and Java clients
- Run them against your versioned endpoints

Your API is only as good as the client your partner can generate

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Key Takeaways

An API is a contract
Version only for breaking changes

URL path
Simple and visible — Atlassian's choice

Media types
Precise and per resource — RFC 6838

Plain WebHandlers
Both styles work on PASOE

One business logic
Typed model, a reader and writer per version

Announce the end
Deprecation, Sunset — and keep your promises

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Sample Code on GitHub

github.com/consultingwerk/
rest-api-versioning-samples

Take it home

- Every sample from this session — plain ABL, PASOE WebHandlers
- Try the curl demo script against your own PASOE
- Issues and pull requests welcome

Scan — before you leave

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

My Sessions at PUG Challenge Amsterdam

Session	Day	Time	Room
Say Bye-Bye to the IDE. Here Comes Claude Code.	Wed 4	16:30 – 17:30	New York 3
AI-Powered OpenEdge Modernization: Faster, Smarter, Scalable	Thu 5	11:45 – 12:45	Athens
REST API Versioning: Concepts and Samples	**Thu 5**	**16:30 – 17:30**	**New York 3**
Cool Things Your Coding Assistant Can Do For You	Fri 6	13:45 – 14:45	New York 1

Highlighted: this session

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.



Subscribe Now to Consultingwerk's Official Newsletter!

www.consultingwerk.com/resources/newsletter

or marketing@consultingwerk.com

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Questions



© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.


Consultingwerk
software architecture and development