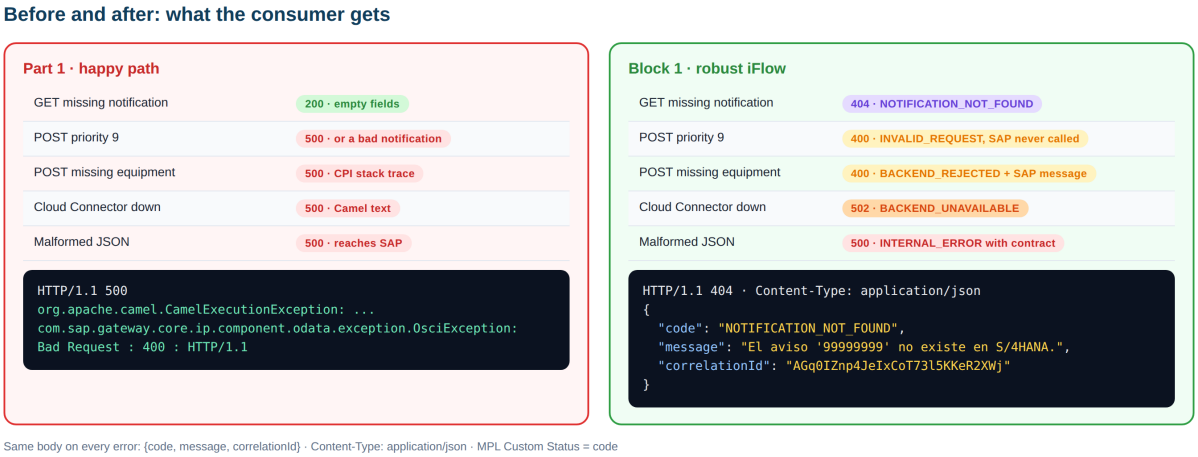
From an opaque 500 to an error contract: the robust iFlow

SAP Integration Suite · Cloud Integration · Practice case MIX-003, part 2 · Variant B, Block 1

In part 1 I left a working endpoint that creates and queries maintenance notifications in an on-premise S/4HANA through Cloud Integration, API Management and Cloud Connector. It worked on the happy path. Anything outside it was an **unexplained 500**: a non-existent notification came back as 200 with empty fields, a malformed JSON travelled all the way to SAP, and if the Cloud Connector was down the consumer got a CPI stack trace.

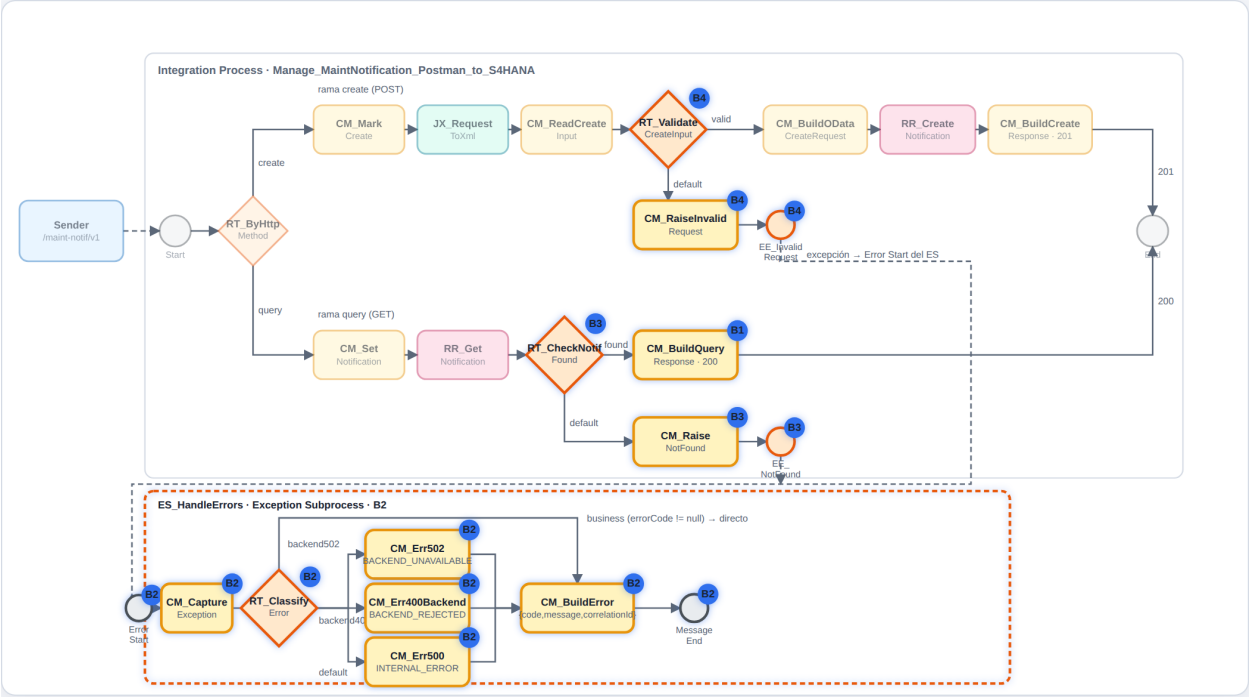
This part closes that gap. Same iFlow, **not a single line of Groovy**: input validation, a proper 404, and an Exception Subprocess that turns every failure into a response with a contract, {code, message, correlationId}, and the right HTTP status.



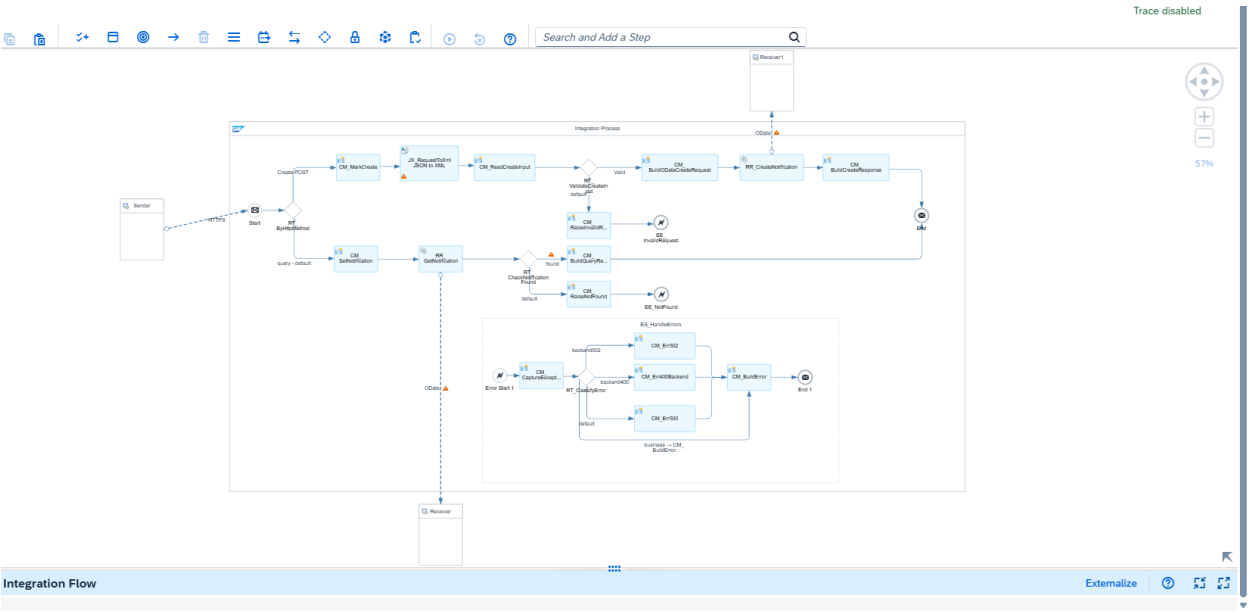
Before and after: what the consumer gets

1. The iFlow after Block 1

The two branches (create and query) that already existed, plus four new pieces, all from the standard palette: a validation Router, an existence Router, two Error End Events and an Exception Subprocess with its own classification Router. The elements numbered B1 to B4 are the ones added in this block.



The iFlow after Block 1: two branches and an Exception Subprocess



Actual iFlow canvas in Cloud Integration

2. How it is built

2.1 Validate before touching SAP

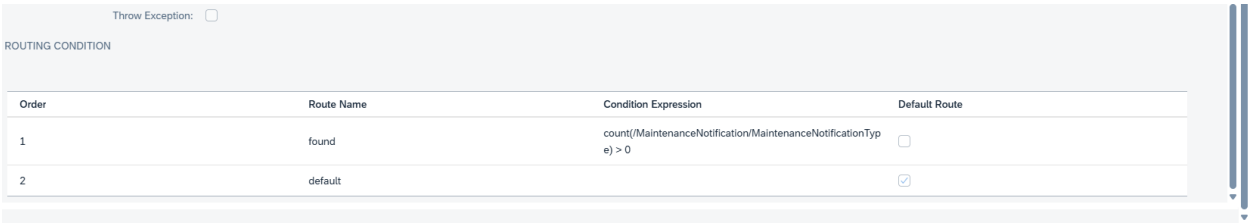
RT_ValidateCreateInput is a Non-XML Router with a single condition over the properties already extracted from the JSON. If it fails, a Content Modifier sets errorStatus = 400, errorCode = INVALID_REQUEST and the message, and an **Error End Event** raises the exception. SAP receives nothing: the MPL of a rejected test shows no receiver call.

```
${property.req_type} regex '^(M1|M2|M3)$' and ${property.req_priority} regex '^[1-4]$'
and ${property.req_equipment} regex '^[0-9]{1,18}$' and ${property.req_text} regex '^.{1,40}$'
and ${property.req_reportedBy} regex '^[A-Za-z0-9_]{0,12}$'
```

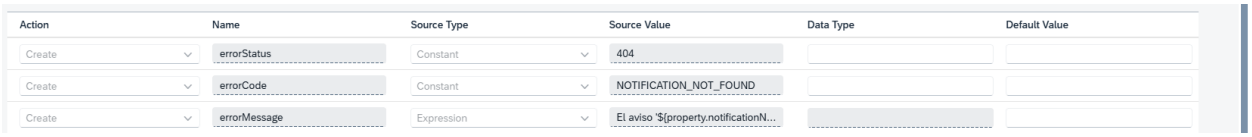
Condition of the valid route

2.2 A 404 on GET

The GET queries with $filter, so a non-existent number is not an error: it is an empty feed with 200. RT_CheckNotificationFound (XML) decides with a count() over the feed; the default route sets 404 / NOTIFICATION_NOT_FOUND and fires another Error End Event.

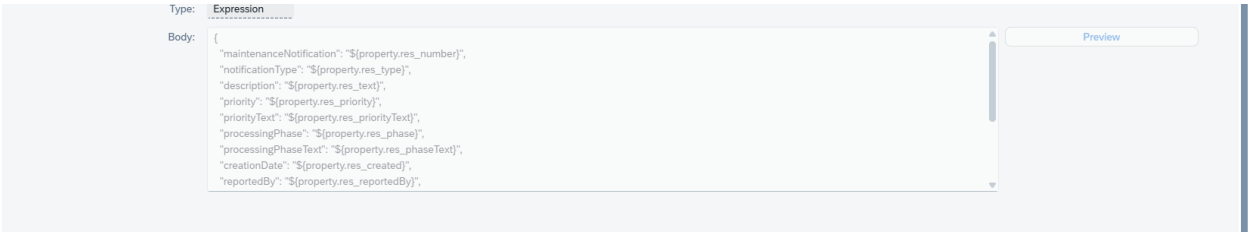


RT_CheckNotificationFound: found and default routes



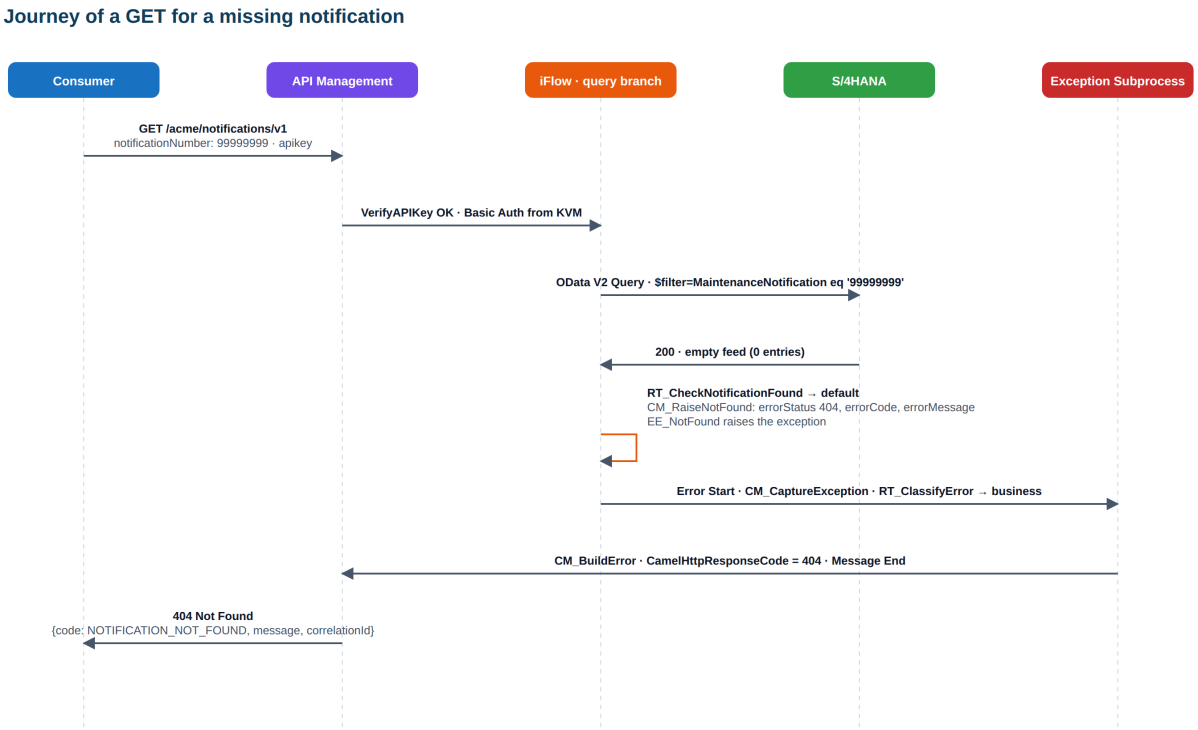
CM_RaiseNotFound: the three properties the Exception Subprocess consumes

When the notification does exist, CM_BuildQueryResponse turns the OData Atom XML into flat JSON with XPath properties and an Expression body.



CM_BuildQueryResponse: JSON body from the res_* properties

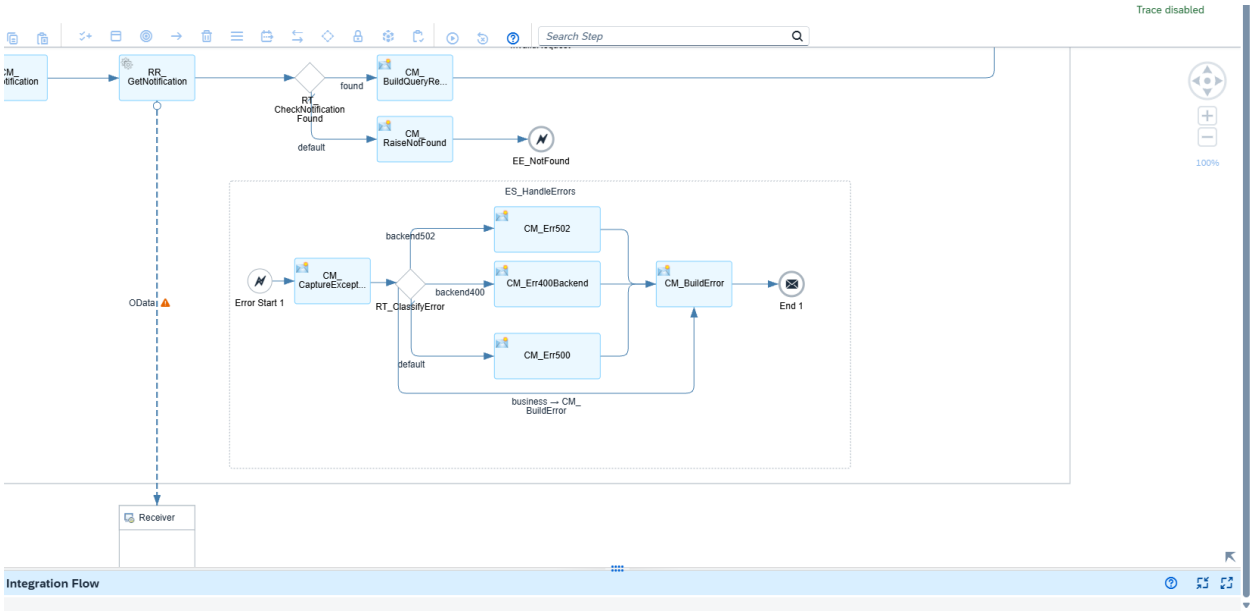
This is the full journey of a GET for a missing notification, from the consumer to the 404:



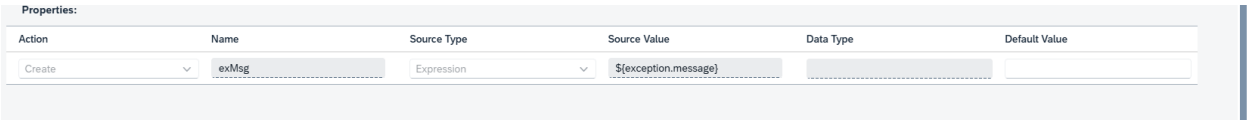
Journey of a GET for a missing notification

2.3 The Exception Subprocess

Any exception in the Integration Process, ours or the adapter's, enters through the Error Start. A Content Modifier copies ${exception.message} into a property, a Router classifies it, each route sets errorStatus / errorCode / errorMessage, and a final Content Modifier builds the response. The subprocess ends with a **Message End**, not an Error End: that is what lets the consumer receive our JSON and our status instead of CPI's 500.

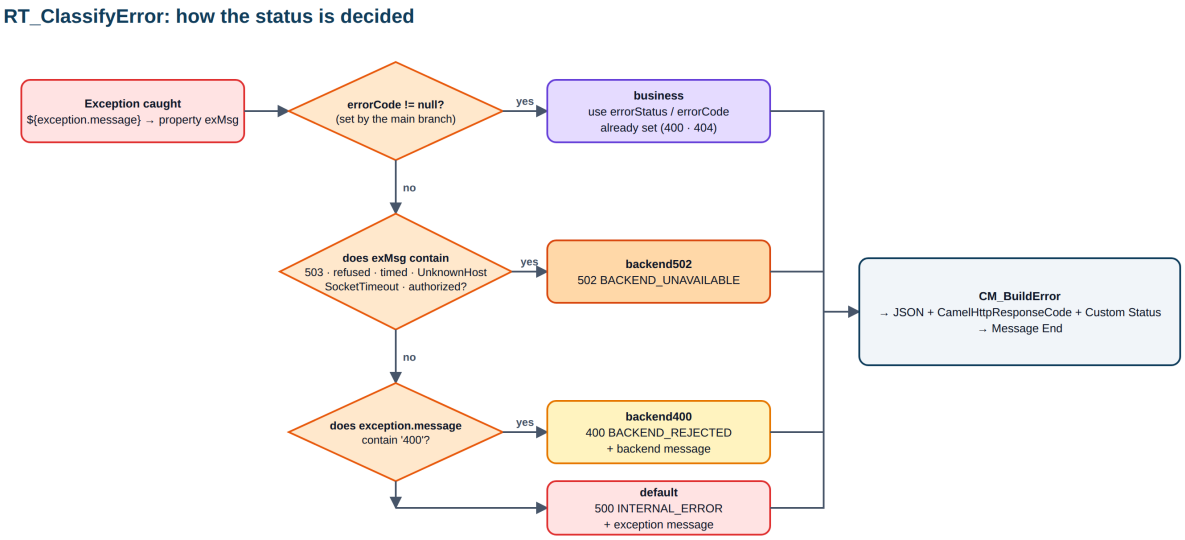


Exception Subprocess ES_HandleErrors

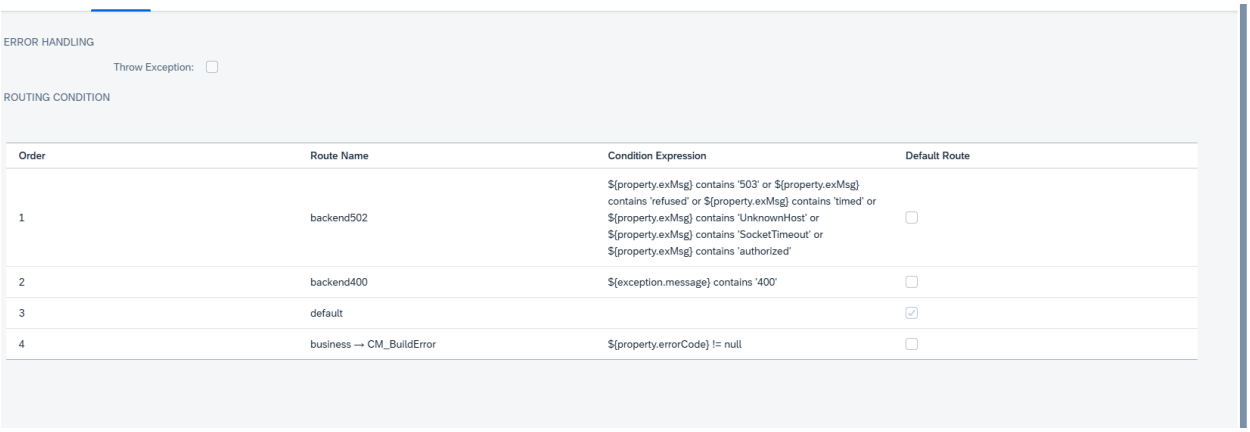


CM_CaptureException: the exception message goes into the exMsg property

Classification is a tree of three questions. The first checks whether the main branch already decided the error (a validation 400 or a 404); the other two look for keywords in the exception message.

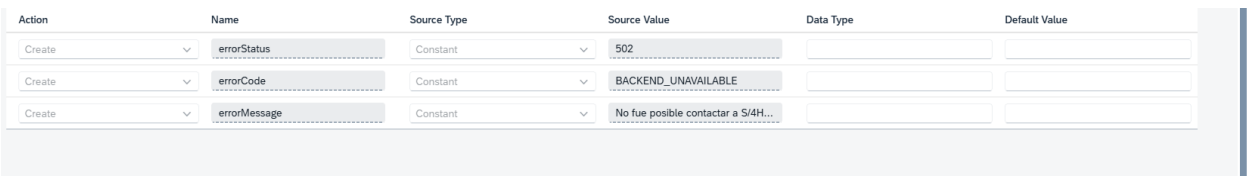


RT_ClassifyError: how the status is decided

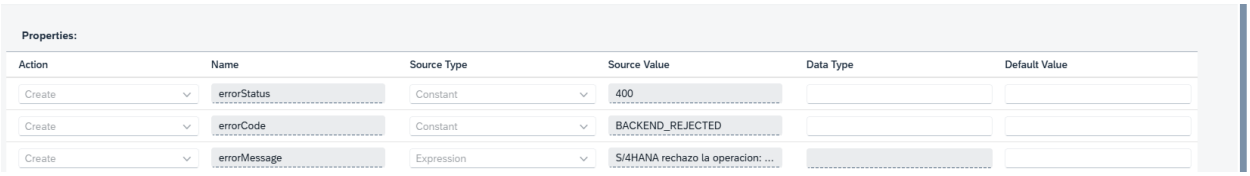


RT_ClassifyError routes and conditions in Cloud Integration

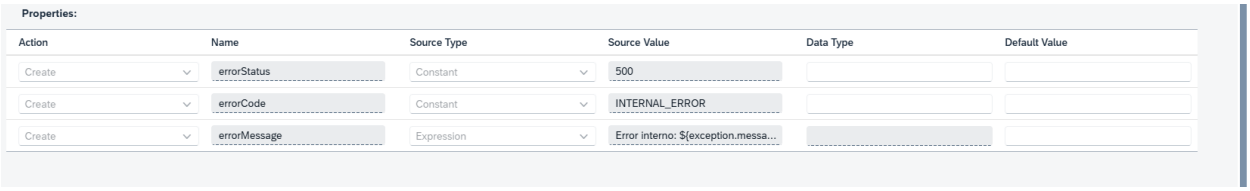
Every route lands on a Content Modifier with three properties. They are identical except for the values; the backend400 and default ones use Expression to include the real message.



CM_Err502: 502 BACKEND_UNAVAILABLE



CM_Err400Backend: 400 BACKEND_REJECTED with the backend message



CM_Err500: 500 INTERNAL_ERROR with the exception message

2.4 The error response

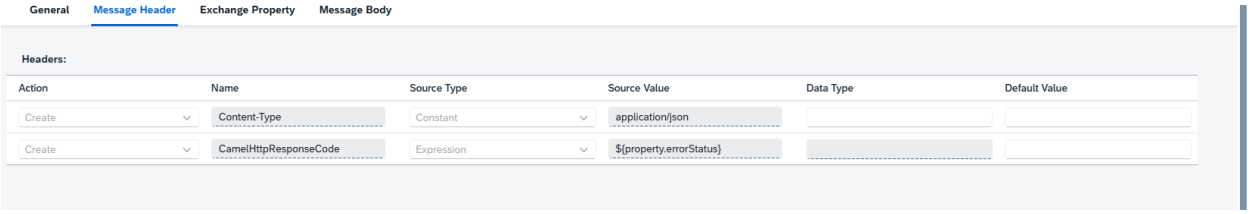
CM_BuildError is where the four routes converge. It sets two headers, Content-Type and CamelHttpResponseCode = ${property.errorStatus}, writes the body, and also stores the code in SAP_MessageProcessingLogCustomStatus, so the MPL can be filtered by BACKEND_REJECTED or NOTIFICATION_NOT_FOUND without opening each message.

Anatomy of the error response

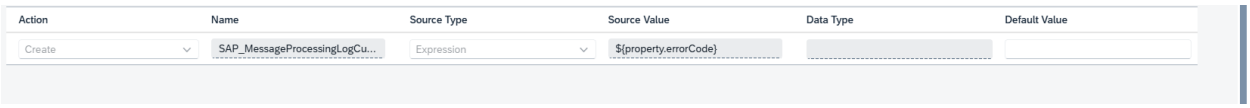
```
HTTP/1.1 400 Bad Request                ◂ CamelHttpResponseCode
Content-Type: application/json
{
  "code": "BACKEND_REJECTED",                ◂ errorCode = MPL Custom Status
  "message": "S/4HANA rechazo la operacion:
              Bad Request : 400 : HTTP/1.1",  ◂ errorMessage
  "correlationId": "AGq0Q5jBX3wjOOND0zlL81I-yoT7" ◂ MPL ID
}
```

`code`	One constant per route. It is also the MPL Custom Status: filter the monitor without opening messages.
`message`	Text for humans. On backend400 it carries the real S/4HANA message; on business, the one set by the main branch.
`correlationId`	SAP_MessageProcessingLogID. The consumer quotes it in the ticket; support opens the exact MPL in a second.
`HTTP status`	Header CamelHttpResponseCode = ${property.errorStatus}. Without it the Message End answers 200.

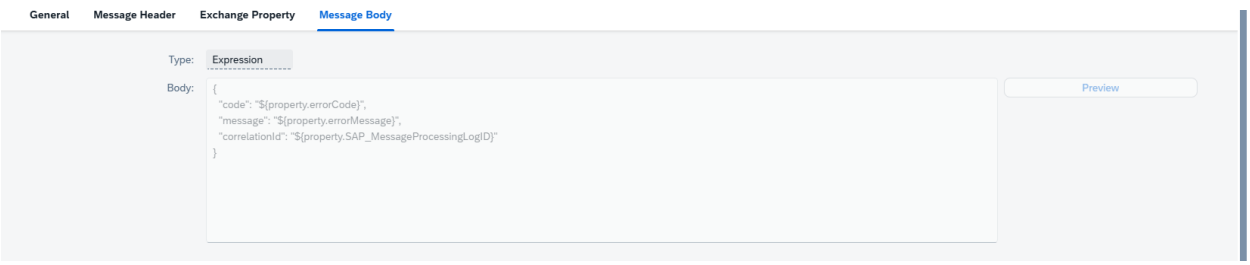
Anatomy of the error response



CM_BuildError: headers



CM_BuildError: MPL Custom Status



CM_BuildError: body

3. The tests

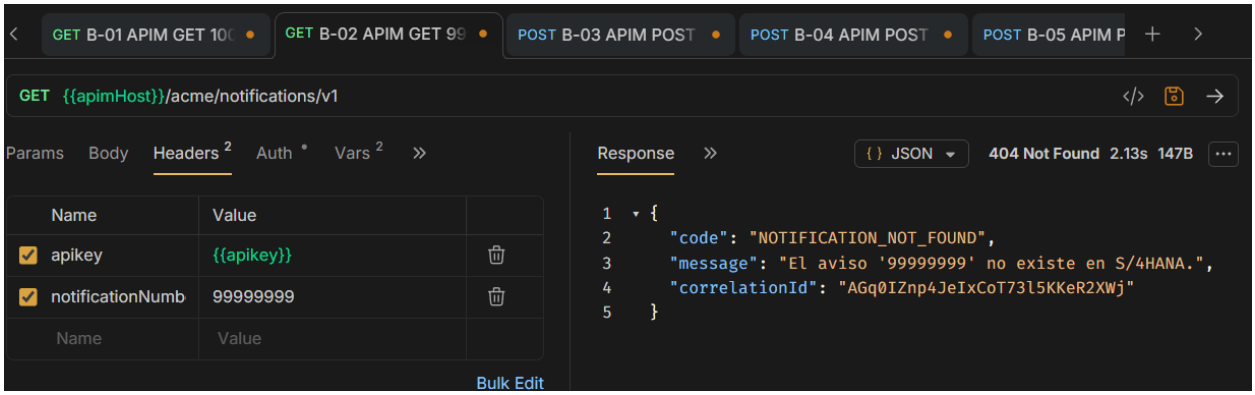
Nine requests in Bruno, all through the API proxy with apikey. The proxy has no Fault Rules yet, so the status and JSON shown are exactly what the iFlow returns.

Block 1 test matrix (Bruno, through the API proxy)

#	Request	Expected	Result	Decides
B-01	GET 10000793	200 flat JSON	✔ OK · 15 fields (after the XPath fix)	CM_BuildQueryResponse
B-02	GET 99999999	404 NOTIFICATION_NOT_FOUND	✔ OK	RT_CheckNotificationFound
B-03	POST priority 9	400 INVALID_REQUEST	✔ OK · no SAP call	RT_ValidateCreateInput
B-04	POST empty equipment	400 INVALID_REQUEST	✔ OK	RT_ValidateCreateInput
B-05	POST equipment 99999999	400 BACKEND_REJECTED	✔ OK · message = Bad Request : 400	ES · backend400
B-06	GET with Cloud Connector stopped	502 BACKEND_UNAVAILABLE	✔ OK	ES · backend502
B-07	POST malformed JSON	500 INTERNAL_ERROR	✔ OK · parser message	ES · default
B-08	Valid POST	201	✔ OK · notification 10000960, equipment without zeros	validation + RR_Create
B-09	GET the notification from B-08	200 same number	✔ OK · priorityText, mainWorkCenter…	CM_BuildQueryResponse

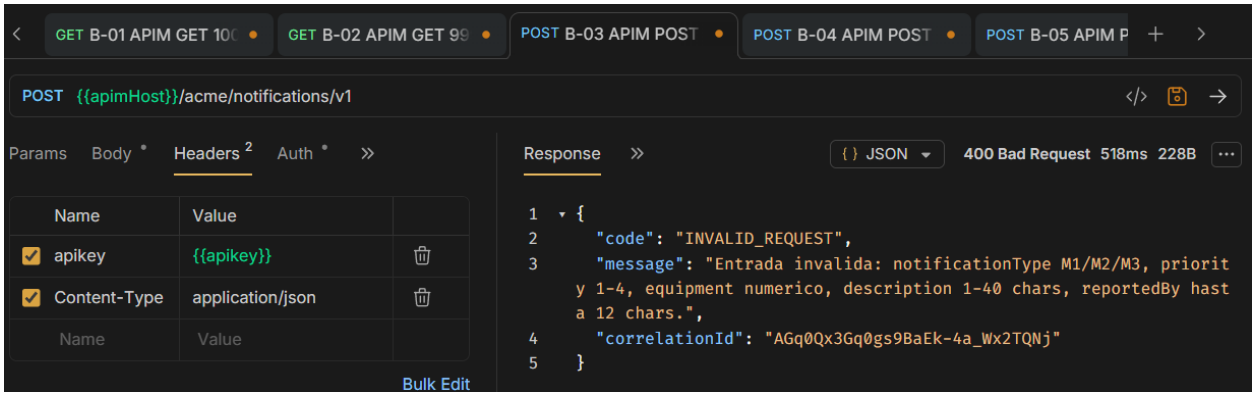
Block 1 test matrix

404 · non-existent notification. The backend returned an empty feed; the iFlow turned it into a 404.

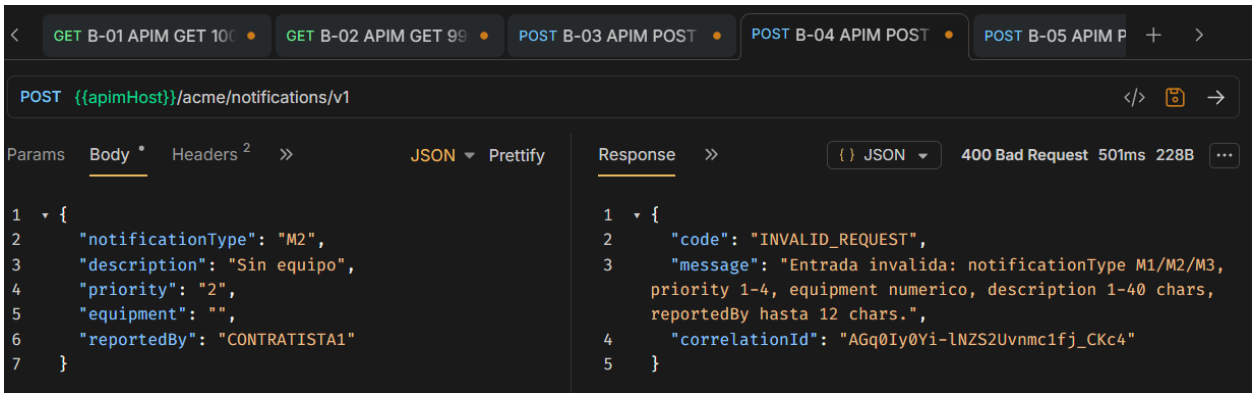


B-02: 404 NOTIFICATION_NOT_FOUND

400 · priority 9 and empty equipment. Rejected by the validation Router. The receiver call never shows up in the MPL: SAP was never involved.

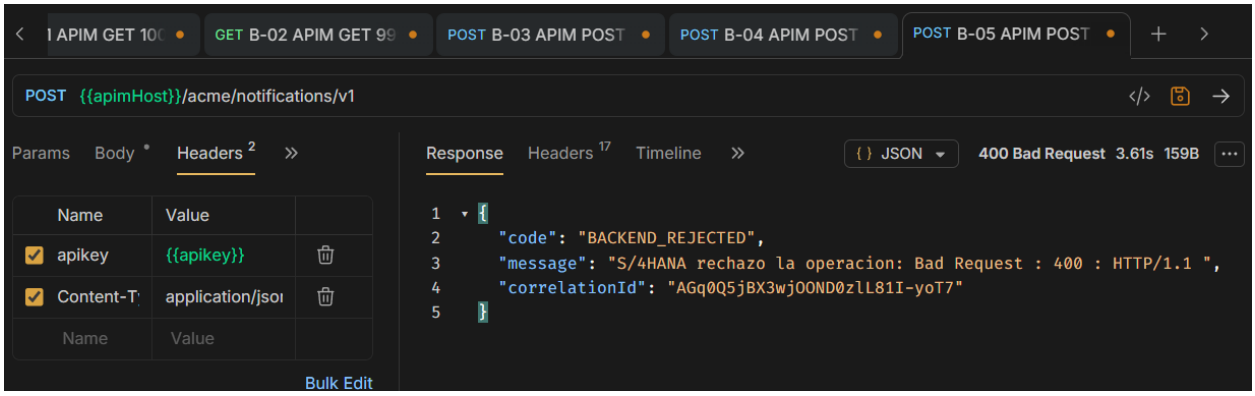


B-03: 400 INVALID_REQUEST for priority 9



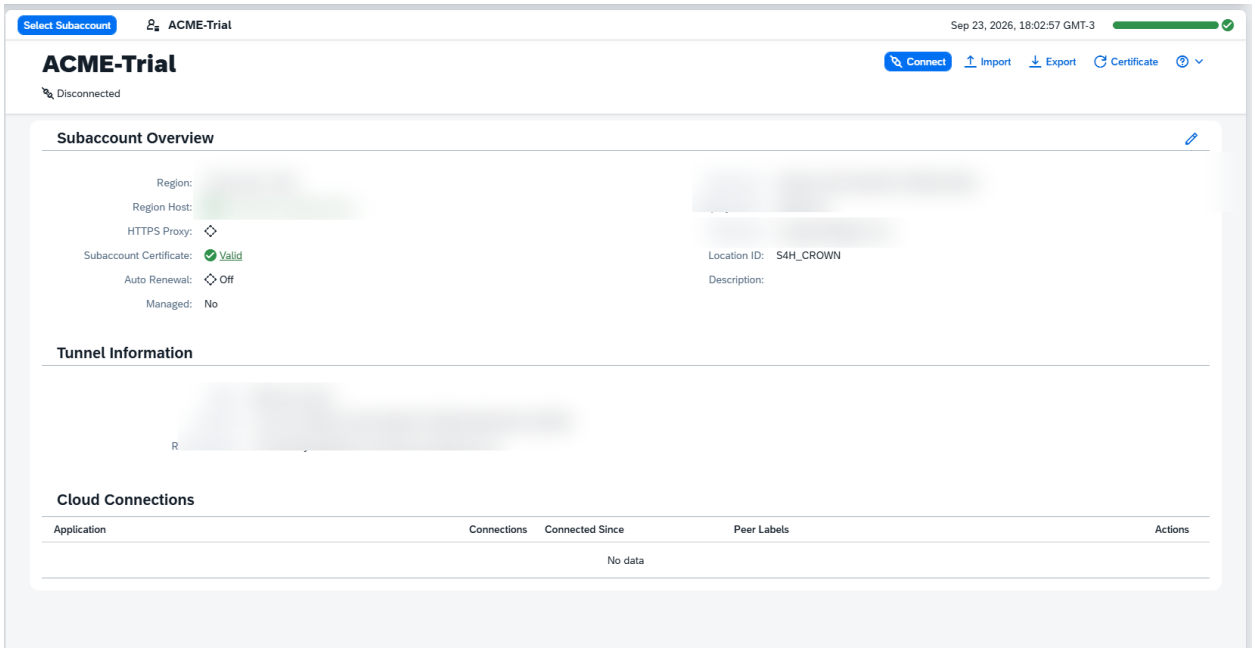
B-04: 400 INVALID_REQUEST for empty equipment

400 · equipment 99999999. Passes the syntactic validation, S/4HANA rejects it, and the backend message travels in the response.

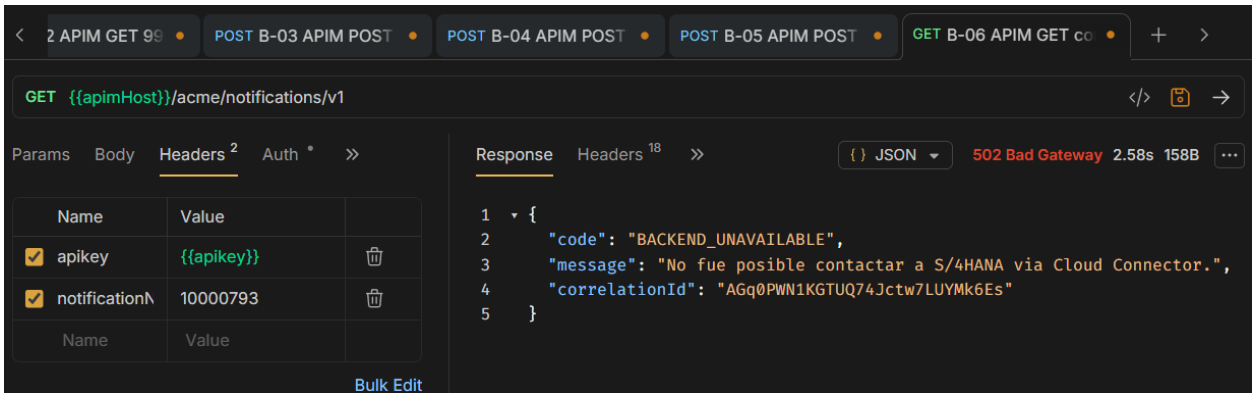


B-05: 400 BACKEND_REJECTED

502 · Cloud Connector stopped. I disconnected the tunnel on purpose.

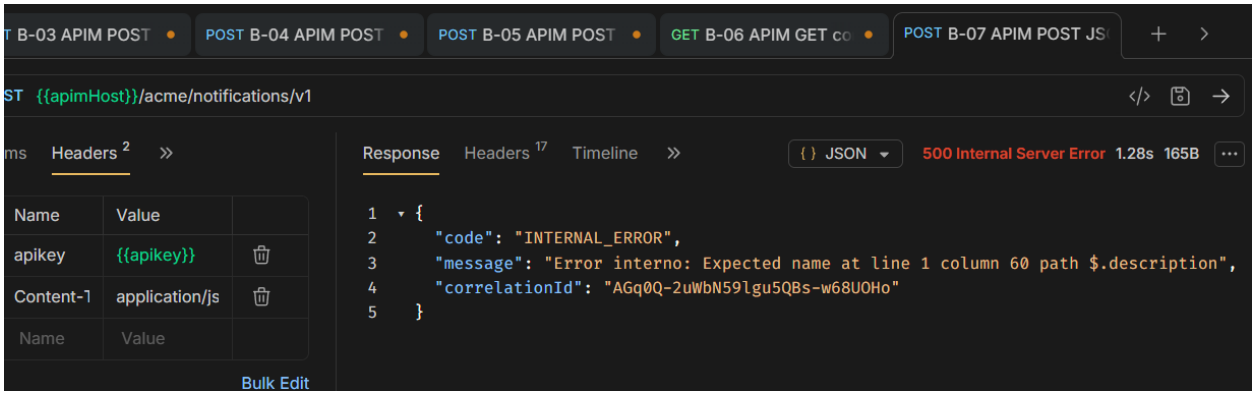


Cloud Connector disconnected



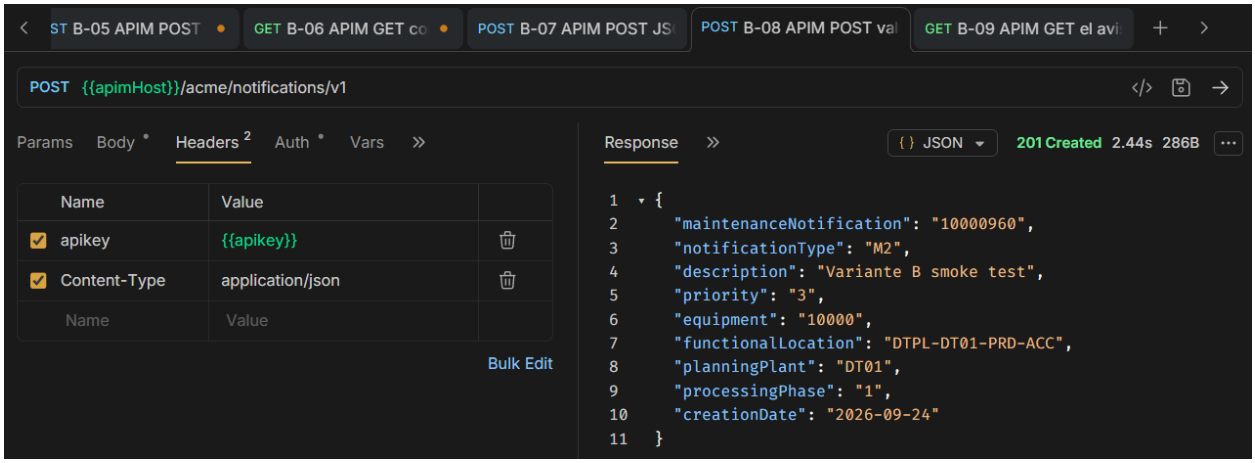
B-06: 502 BACKEND_UNAVAILABLE

500 · malformed JSON. One extra comma breaks in the JSON to XML Converter, before any validation. It falls through the default route with the parser message. A 500 is accepted for this block; turning it into 400 MALFORMED_JSON is one more route on the Router.



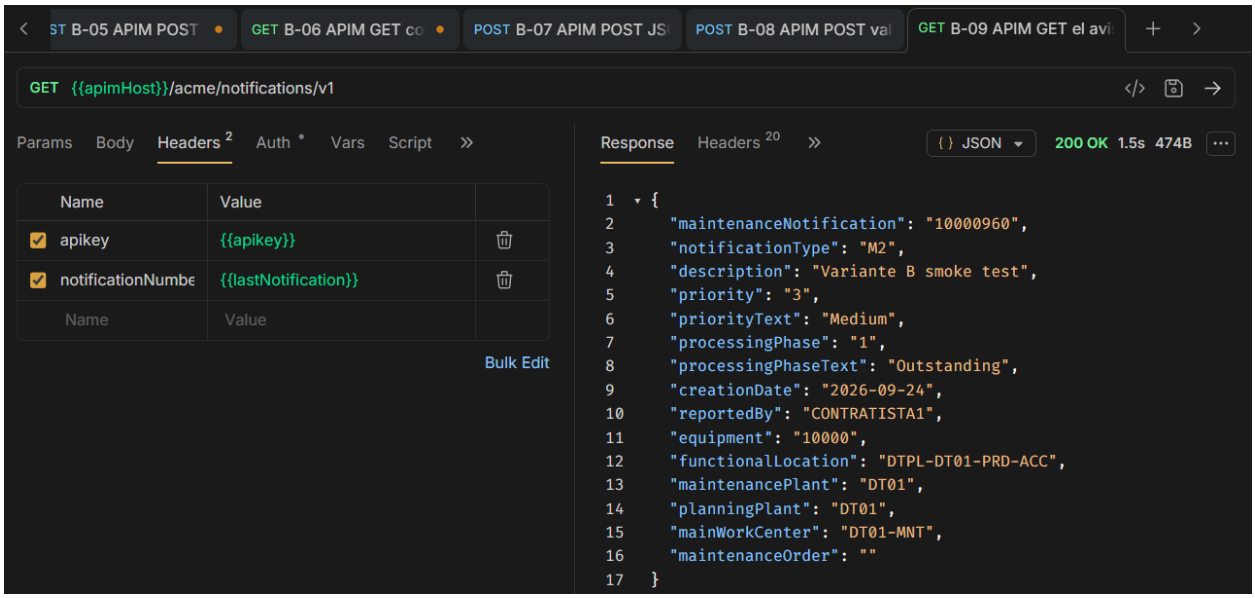
B-07: 500 INTERNAL_ERROR with the parser message

201 · valid POST. The happy path keeps working with validation and the subprocess active: notification 10000960, with functional location and plant derived by S/4HANA and the equipment number already without leading zeros (TechnicalObjectLabel).



B-08: 201 Created

200 · GET of the new notification. The flat 15-field JSON: priority with its text, processing phase, derived functional location, plant and work center, and an empty maintenanceOrder because the notification has no order yet.



B-09: 200 with the flat JSON of notification 10000960

4. What broke during the build (and how I fixed it)

The Router's design-time validator was the main character. It is stricter than the runtime and fails in layers: every time I fixed one error, the next one showed up.

The Router validator fails in layers

1

```
Invalid format expression
```

regex with (?is) flags or with spaces

→ **contains chained with or**

2

```
Token ''Connection' not supported after 'contains'
```

tokenizes on spaces: 'Connection refused' is two tokens

→ **single-word constants: 'refused', 'timed', '400'**

3

```
Token '${exception.message}' not supported after 'or'
```

accepts ${exception.message} only as the first operand

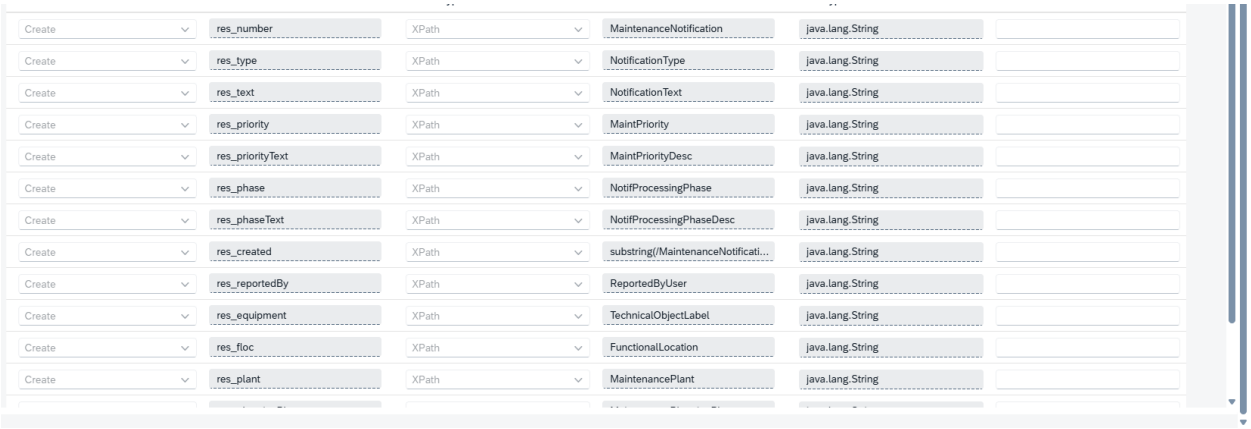
→ **CM_CaptureException copies the message into exMsg; conditions use ${property.exMsg}**

Rule of thumb: single-word constants + the exception message in a property. The runtime would accept all of the above; the blocker is the design-time validator.

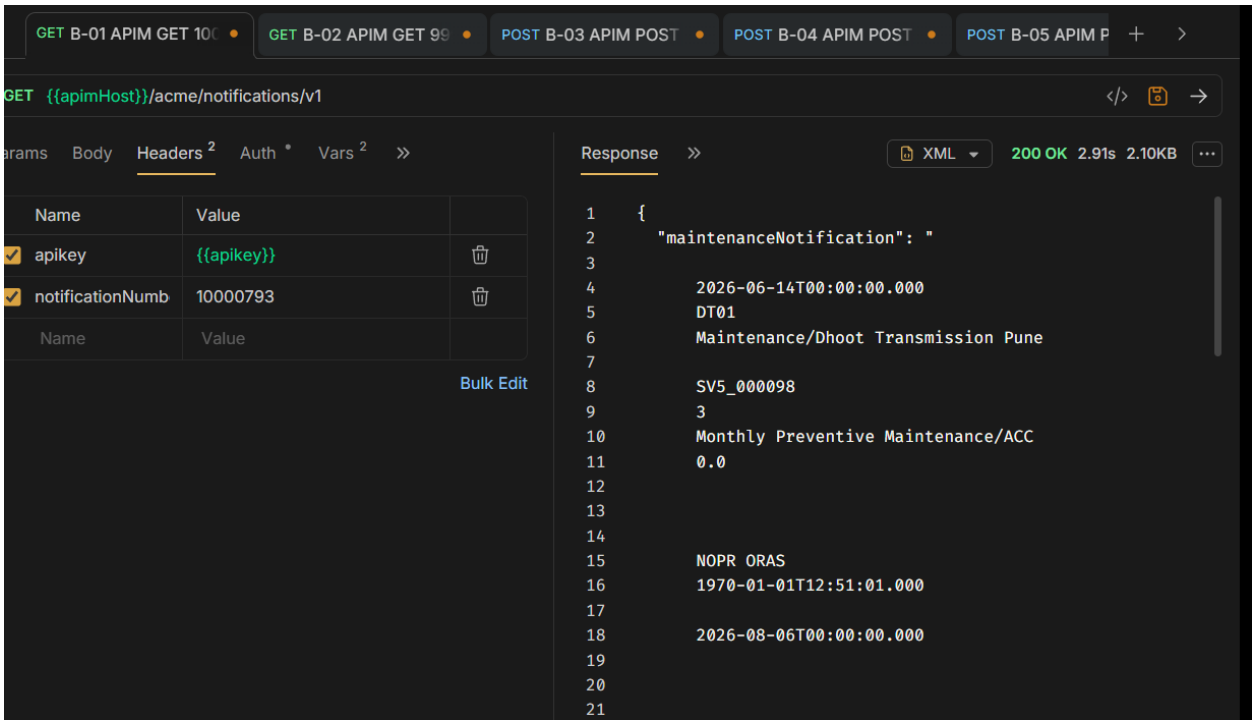
The Router validator fails in layers

Symptom	Real cause	Fix
Valid POST answers INTERNAL_ERROR with Invalid xpath: ${exchangeProperty.req_type} regex …	the valid route was left with Expression Type XML: CPI tried to compile the condition as XPath	Non-XML. Same error as in part 1, on a different Router

Symptom	Real cause	Fix
GET of a missing notification answers INTERNAL_ERROR with message Error Event Exception	the Content Modifiers before the Error End Events existed but **without the Exchange Properties**; the business route never matched	fill in the three properties. As a bonus, it proved that an Error End Event does trigger the Exception Subprocess, no Groovy needed
The error JSON arrives with status 200	CamelHttpResponseCode header set as Constant instead of Expression	Type = Expression, value ${property.errorStatus}
GET returns 200 with maintenanceNotification full of concatenated values	the response Content Modifier's XPath properties held only the field name, no path; MaintenanceNotification matched the root element, whose string value is the whole document	full XPath on every row: /MaintenanceNotification/MaintenanceNotificationType/<field>



XPath without a path: the GET bug



The bug's result: 200 with degenerate JSON

Two design lessons. First: **a number with an invalid format is not a 404**. My first

"non-existent notification" test used a 14-digit number and S/4HANA rejected the $filter itself

with a 400; the 404 only appears for a valid number that does not exist. Second: **the exception of an Error End Event arrives with a fixed text**

(Error Event Exception), so keyword-based classification will never confuse it with a backend

error.

5. What is next

Block 2, in API Management: Spike Arrest, per-application Quota, stripping the apikey header before calling CPI, and Fault Rules with the same error contract for proxy-side rejections.
Block 3: externalized parameters and a consumer-provided X-Correlation-ID replacing the MPL ID in responses.

Tools: SAP Integration Suite Trial (Cloud Integration, API Management), SAP Cloud Connector, SAP S/4HANA 2025 on-premise, Bruno. The company is fictional; technical data comes from a lab system. No credentials or internal hosts in this document.