How to Perform Request/Response Validation with Test Case Designer

In the article related to APIs, we looked at the basics of applying TCD to testing the business logic (e.g., successful payment given many factors, their interactions, and associated rules).

In this one, let’s talk about the data as it relates to the communication between systems – using this guide by Pact as a reference (*note, however – these Test Case Designer methods are not limited to or specific to contract testing*).

We will leverage a slightly modified setup. There are still a consumer (Order Web) and its provider (the Order API). And we will still be submitting the request for product information from Web to API, but will “enrich” the attributes a bit:

Request:

- Market (e.g., NA, EU)
- Product Category (e.g., A, B)

One or both factors need to be present for the successful request

Response :

(keeping in mind we care more about the structure and format compatibility than about the business logic calculations)

- Quantity (e.g., whole, partial)
- Value (e.g., whole, decimal)
- Date last sold (e.g., mm/dd/yyyy format, dd/mm/yyyy format)
- primary vendor ID (e.g., company itself, partner, unrelated 3rd party)
- Status (200 for the scope of this article, but other non-error ones could be included in the same model as well with relevant triggers)

Some new products have not been evaluated and/or sold yet.

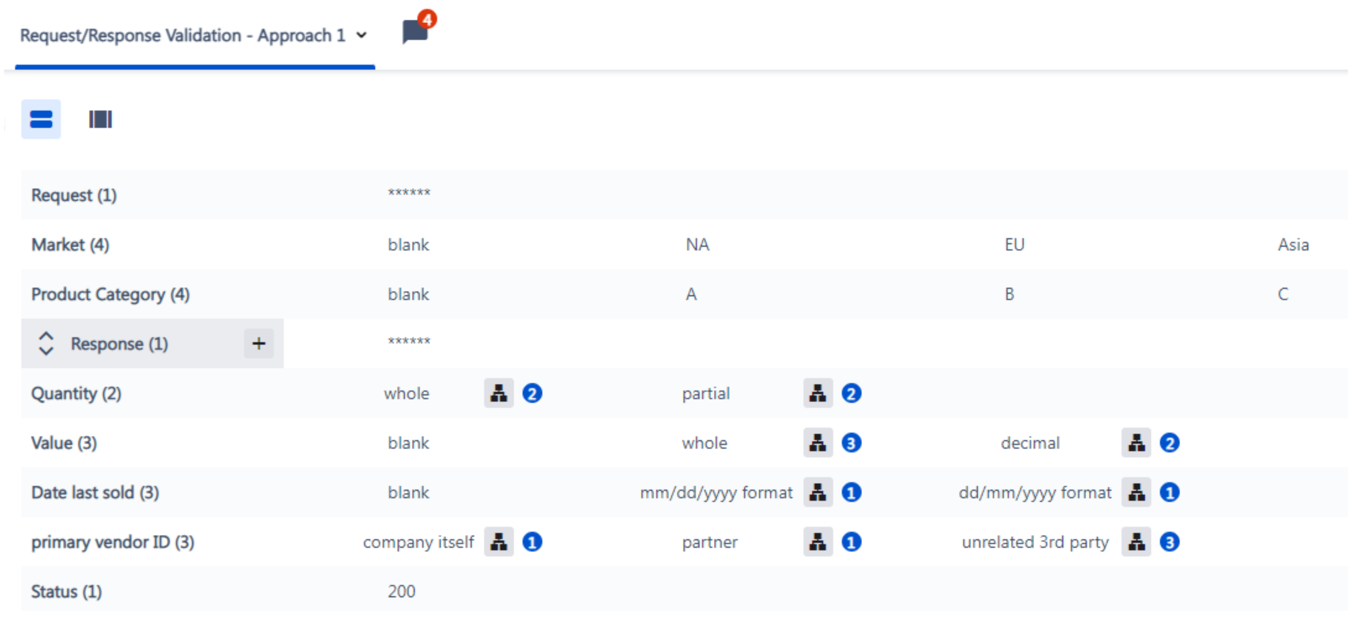
- Modeling in Test Case Designer
    - Approach 1 – Whole response profile per test case
    - Approach 2 – Attribute per test case
- Decision points
- Conclusion

Modeling in Test Case Designer

Both approaches described below are viable, and the choice will depend on the specific system & testing goals, as discussed in the “Decision Points” section later.

Approach 1 – Whole response profile per test case

This is similar to the model designed in the API article. We will have a TCD parameter for each eligible request and response attribute that a) varies in a finite manner; b) needs to be validated.

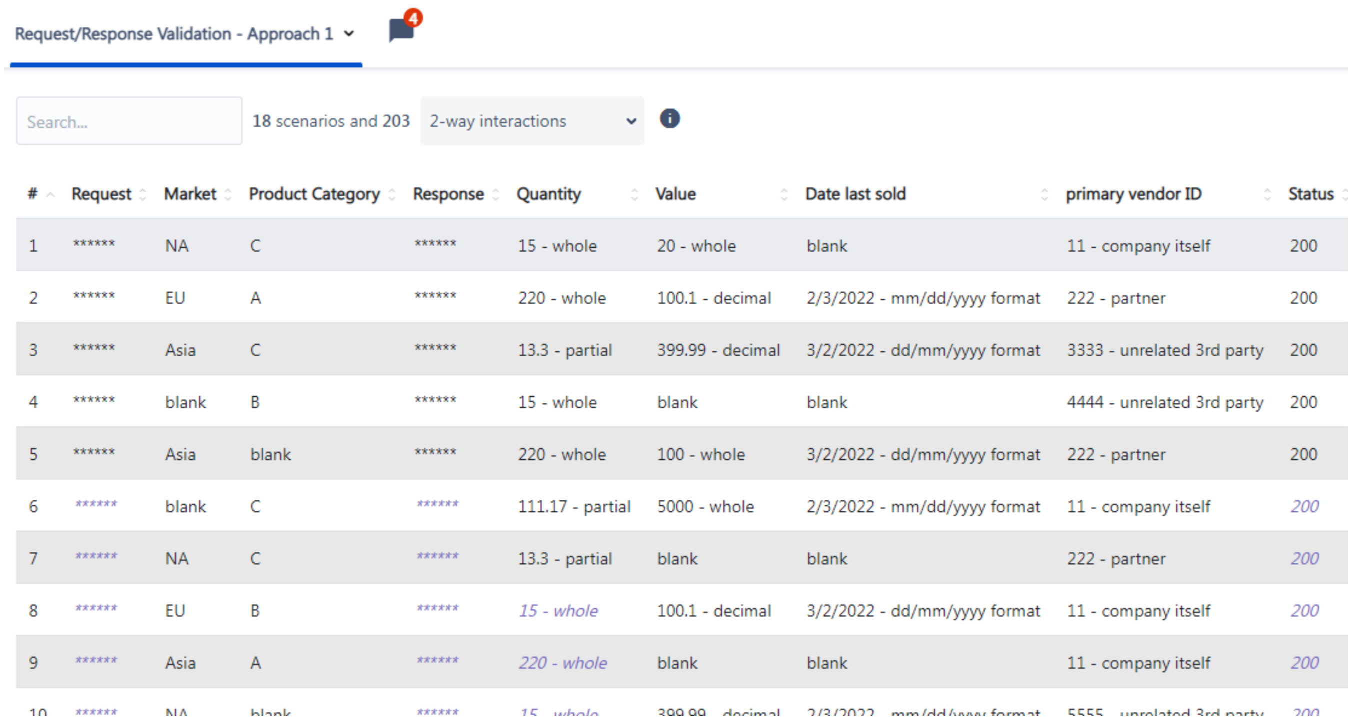


Request/Response Validation - Approach 1

Request (1)	******			
Market (4)	blank	NA	EU	Asia
Product Category (4)	blank	A	B	C
Response (1)	******			
Quantity (2)	whole ②	partial ②		
Value (3)	blank	whole ③	decimal ②	
Date last sold (3)	blank	mm/dd/yyyy format ①	dd/mm/yyyy format ①	
primary vendor ID (3)	company itself ①	partner ①	unrelated 3rd party ③	
Status (1)	200			

Value expansions play the role of data specification in such models (e.g., 2/3/2022) since they would be the only element populated in the CSV or Automate export (typically used for this testing).

Each row in the Scenarios table would describe **all** parameterizable response attributes for a given request combination.



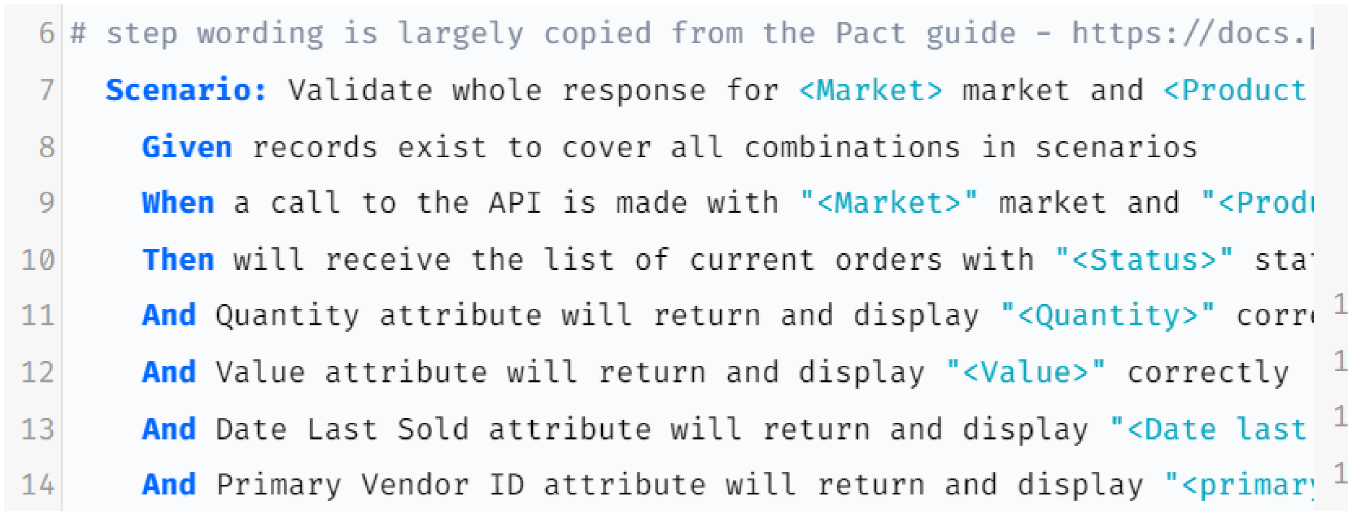
Request/Response Validation - Approach 1

Search... 18 scenarios and 203 2-way interactions

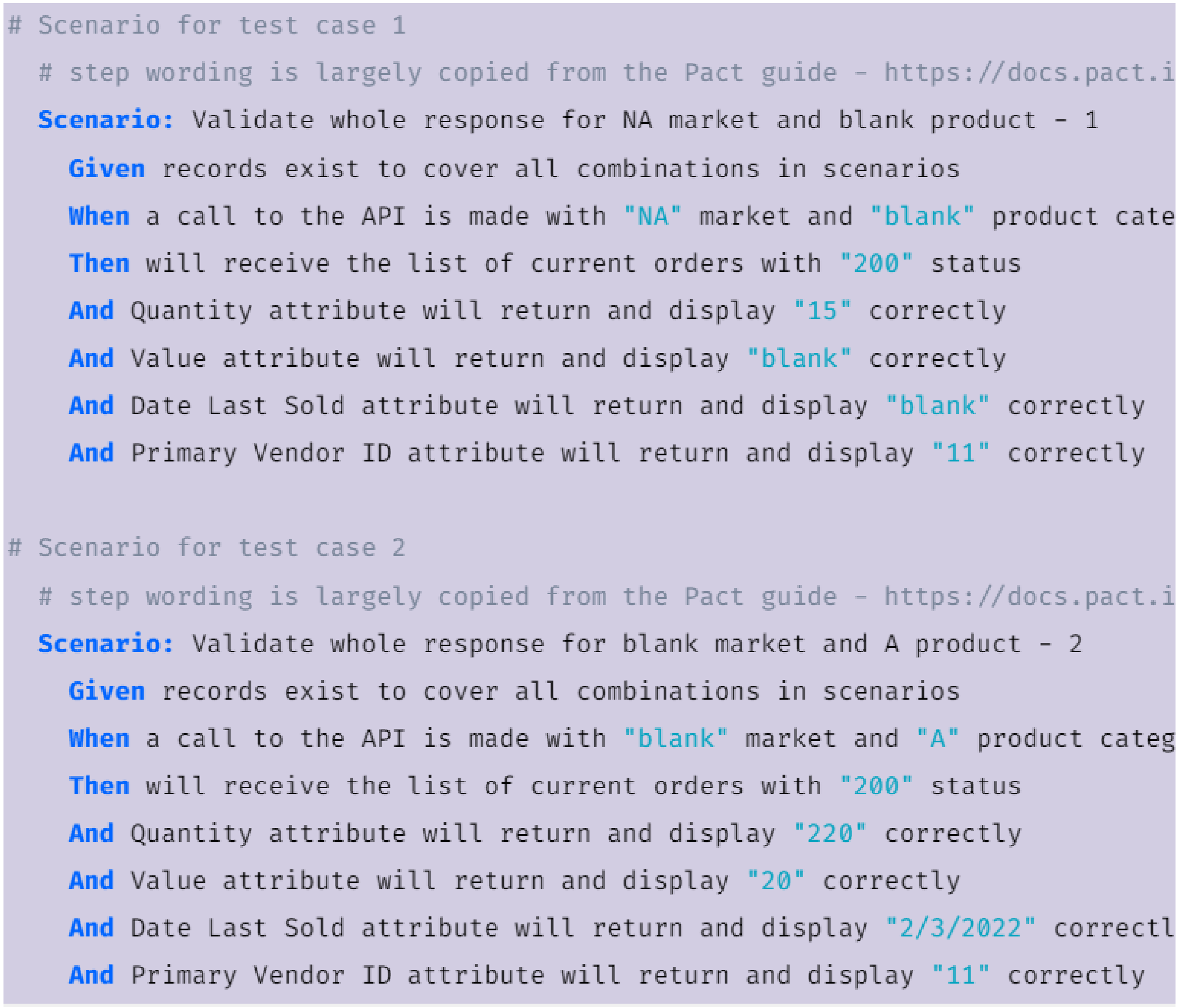
#	Request	Market	Product Category	Response	Quantity	Value	Date last sold	primary vendor ID	Status
1	******	NA	C	******	15 - whole	20 - whole	blank	11 - company itself	200
2	******	EU	A	******	220 - whole	100.1 - decimal	2/3/2022 - mm/dd/yyyy format	222 - partner	200
3	******	Asia	C	******	13.3 - partial	399.99 - decimal	3/2/2022 - dd/mm/yyyy format	3333 - unrelated 3rd party	200
4	******	blank	B	******	15 - whole	blank	blank	4444 - unrelated 3rd party	200
5	******	Asia	blank	******	220 - whole	100 - whole	3/2/2022 - dd/mm/yyyy format	222 - partner	200
6	*******	blank	C	*******	111.17 - partial	5000 - whole	2/3/2022 - mm/dd/yyyy format	11 - company itself	*200*
7	*******	NA	C	*******	13.3 - partial	blank	blank	222 - partner	*200*
8	*******	EU	B	*******	*15 - whole*	100.1 - decimal	3/2/2022 - dd/mm/yyyy format	11 - company itself	*200*
9	*******	Asia	A	*******	*220 - whole*	blank	blank	11 - company itself	*200*
10	*******	NA	blank	*******	*15 - whole*	399.99 - decimal	2/3/2022 - mm/dd/yyyy format	5555 - unrelated 3rd party	*200*

With data validation testing, there are usually fewer interactions/dependencies between the attributes. So, the mixed-strength setting with some 2-way on the request side and mostly 1-way on the response one is expected.

One script with a "Then" line per attribute would cover all scenarios:



```
# step wording is largely copied from the Pact guide - https://docs.
  Scenario: Validate whole response for <Market> market and <Product
    Given records exist to cover all combinations in scenarios
    When a call to the API is made with "<Market>" market and "<Prod
    Then will receive the list of current orders with "<Status>" sta
    And Quantity attribute will return and display "<Quantity>" corr
    And Value attribute will return and display "<Value>" correctly
    And Date Last Sold attribute will return and display "<Date last
    And Primary Vendor ID attribute will return and display "<primar
```



```
# Scenario for test case 1
  # step wording is largely copied from the Pact guide - https://docs.pact.i
  Scenario: Validate whole response for NA market and blank product - 1
    Given records exist to cover all combinations in scenarios
    When a call to the API is made with "NA" market and "blank" product cate
    Then will receive the list of current orders with "200" status
    And Quantity attribute will return and display "15" correctly
    And Value attribute will return and display "blank" correctly
    And Date Last Sold attribute will return and display "blank" correctly
    And Primary Vendor ID attribute will return and display "11" correctly

# Scenario for test case 2
  # step wording is largely copied from the Pact guide - https://docs.pact.i
  Scenario: Validate whole response for blank market and A product - 2
    Given records exist to cover all combinations in scenarios
    When a call to the API is made with "blank" market and "A" product categ
    Then will receive the list of current orders with "200" status
    And Quantity attribute will return and display "220" correctly
    And Value attribute will return and display "20" correctly
    And Date Last Sold attribute will return and display "2/3/2022" correctl
    And Primary Vendor ID attribute will return and display "11" correctly
```

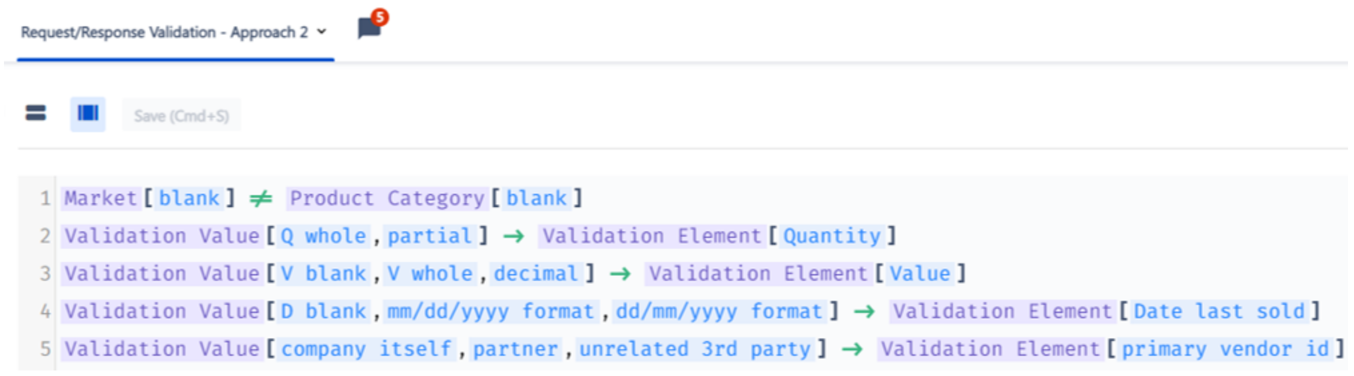
If there are attributes that don't have format variations, can't be blank, and therefore wouldn't become TCD parameters, the steps & validations for those would be hardcoded (i.e., without <> syntax) in the script.

Approach 2 – Attribute per test case

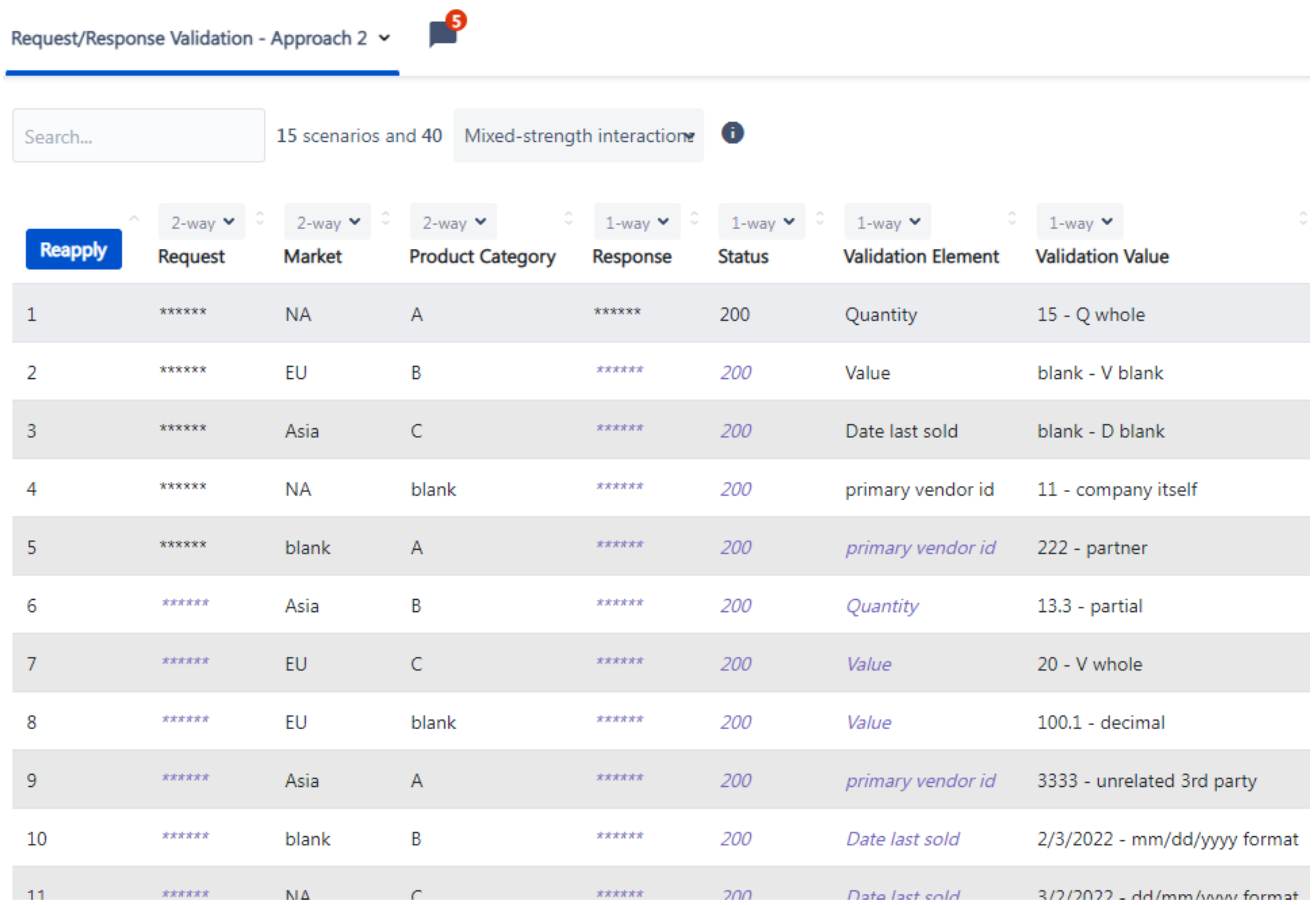
To enable this in TCD, we will "transpose" our thinking from Approach 1. We will have a pair of parameters – "Validation Element" (the list of all non-status response attributes we need to check) and "Validation Value" (the list of all non-status response values we need to check).

```
Request [ ****** ]
Market [ blank , NA , EU , Asia ]
Product Category [ blank , A , B , C ]
Response [ ****** ]
Status [ 200 ]
Validation Element [ Quantity , Value , Date last sold , primary vendor id ]
Validation Value [ Q whole , partial , V blank , V whole , decimal , D blank , mm/dd/yyyy format , dd/mm/yyyy format , company itself ,
```

Then we will constrain each Validation Element only to the relevant Value.

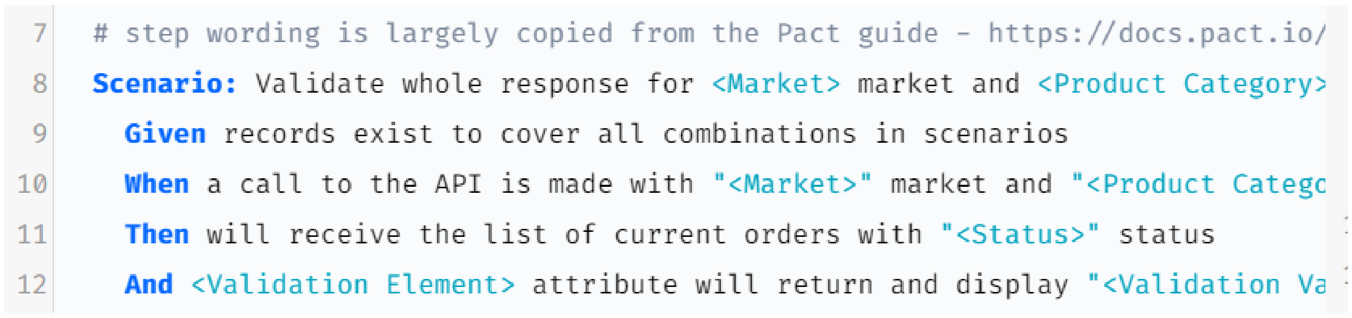


```
Market [ blank ] ≠ Product Category [ blank ]
Validation Value [ Q whole , partial ] → Validation Element [ Quantity ]
Validation Value [ V blank , V whole , decimal ] → Validation Element [ Value ]
Validation Value [ D blank , mm/dd/yyyy format , dd/mm/yyyy format ] → Validation Element [ Date last sold ]
Validation Value [ company itself , partner , unrelated 3rd party ] → Validation Element [ primary vendor id ]
```


Request/Response Validation - Approach 2
Search...
15 scenarios and 40
Mixed-strength interactions
Reapply

	2-way	2-way	2-way	1-way	1-way	1-way	1-way
	Request	Market	Product Category	Response	Status	Validation Element	Validation Value
1	******	NA	A	******	200	Quantity	15 - Q whole
2	******	EU	B	******	*200*	Value	blank - V blank
3	******	Asia	C	******	*200*	Date last sold	blank - D blank
4	******	NA	blank	******	*200*	primary vendor id	11 - company itself
5	******	blank	A	******	*200*	*primary vendor id*	222 - partner
6	******	Asia	B	******	*200*	*Quantity*	13.3 - partial
7	******	EU	C	******	*200*	*Value*	20 - V whole
8	******	EU	blank	******	*200*	*Value*	100.1 - decimal
9	******	Asia	A	******	*200*	*primary vendor id*	3333 - unrelated 3rd party
10	******	blank	B	******	*200*	*Date last sold*	2/3/2022 - mm/dd/yyyy format
11	******	NA	C	******	*200*	*Date last sold*	3/2/2022 - dd/mm/yyyy format

A single script with a single "Then" line would cover all scenarios because the key wording is dynamically tied to the TC table:



```
# step wording is largely copied from the Pact guide - https://docs.pact.io/
Scenario: Validate whole response for <Market> market and <Product Category>
  Given records exist to cover all combinations in scenarios
  When a call to the API is made with "<Market>" market and "<Product Catego
  Then will receive the list of current orders with "<Status>" status
  And <Validation Element> attribute will return and display "<Validation Va
```

Decision points

Approach 1 – Pros:

- If there is any validation dependency between response attributes, this approach has a higher chance of catching defects.
- Less vulnerable to setup costs per TC (i.e., in an absurd example, if each test requires a unique API token that costs $1000, then executing a test per response profile is much cheaper than a test per attribute).

Approach 1 – Cons:

- More complex and less flexible execution-wise (i.e., more steps to get to the end of the scenario).
- More vulnerable to test data availability (if the request is sent against the real database or the mock that was built only based on the production sample, the "free" combinations that Test Case Designer algorithm generates may result in "record not found" too often).

To no surprise, the points below are the mirror reflection of the above.

Approach 2 – Pros:

- Quicker and more flexible execution of “componentized” TCs (i.e., if only 1 API response attribute changes, you don’t need to re-execute all the steps to get to that one).
- Less vulnerable to test data availability (if a valid standalone attribute value is not present in the mock/real database, that’s probably not a good sign and should be solved separately).

Approach 2 – Cons:

- Will have a much lower chance of catching any interaction defects (i.e., if Value is not retrieved correctly only when an unrelated 3rd party vendor is involved).
- More vulnerable to setup costs per TC (higher total setup cost in the “$1000 per token” example above).

“

"Number of tests” as a metric becomes irrelevant in this comparison since the number of steps per test and the corresponding execution time /effort are too different.

Conclusion

Hopefully, this article has demonstrated how Test Case Designer can be applied to use cases where n-way interactions are no longer the priority. The speed of scenario generation and the one-to-many scripting move into the spotlight, so the tool can still deliver benefits in either approach.

Extra consideration: shared TCD model can serve as another collaboration artifact between the consumer and the provider, which could allow for uncovering mismatching expectations much faster.