**Motability: Technical Test Solution**

**Creating the Initial Application**

Initial step: check the format, types of each data field, and values of the JSON containing the application data. Perform data field type and format checks.

* Check fields within the OLA application data for type correctness and format validity, for example:
* Check that the dealer code is a string
* Check that the customer first name is a string
* Check that customer date of birth is of date time format.

**See the ‘Pre-request Script’ tests in the ‘Initial Application’ query in the Postman solution for the sorts of format tests which could be performed on the initial input data.**

The OLA data must contain a field to indicate whether the customer has accepted DWP consent, ie is receiving the required disability benefit. This field could take several forms, eg a string of characters, but the simplest would be a Boolean true/false value. Two key checks to make would therefore be whether this field is of type Boolean (if this is the valid format) and that the value is correct, ie the field’s actual value is ‘true’.

Next step is to check the fields which specify the drivers to be added. These might be structured as follows:

* Driver name
* Driver address
* Driver date of birth
* Date licence was obtained
* and possibly a few others

Format and data type checks would be needed for these fields.

Most important would be to check the number of drivers is less than or equal to 3 – and at least 1, as there needs to be at least 1 driver per application

Vehicle adaptations are also specified. These might take a number of forms, so the testing will depend on how the data is structured and the form it takes. For example, specialised hand controls or wheelchair hoists. One possible way to structure the data would be an array which lists all the allowed adaptations for each driver, together with a Boolean which specifies whether a given driver requires that adaptation. Testing of this data might include checking that the data is in JSON format, and that the values for each adaptation are Boolean.

To test the estimated delivery date, again format checks to check that the data is of date/time format would be needed, plus and range checks specifying maximum or minimum values for the dates, eg if there is a maximum due date for delivery.

To test customer communication preferences, this would likely be an array from a list of specific choices, eg by letter or email. Testing could include checking that the value in this field, eg ‘email’, comes from the list of allowed values for the communication method.

**Other Tests on OLA Data**

As well as checks on the OLA data format, values and range checks, testing should include at a minimum, status code checks to ensure that the API calls were successful (ie 200 OK or 201 to prove the resource was successfully created), plus checks on response times for that API call to check that these were within permitted values.

**See the Pre-request Script in the** **‘DVLA Driver Eligibility’ Postman solution query for the JavaScript implementation of the tests on the application data after it has been supplemented by the addition of driver and vehicle adaptation data.**

**Business Rule Checks for Driver Eligibility**

The business rules could be translated into assertion statements in Postman, depending on the exact form of the category codes.

For the business rule examples below, pm.response denotes the response of the API which queries the DVLA database using the driver data from the application. See the Postman solution for the full code underlying the business rules.

**See the ‘Tests’ scripts in the ‘DVLA Driver Eligibility’ Postman solution query for the sorts of business rule tests which could be performed on the API response data.**

**NOTE**: In the Postman solution, the main query is the POST query to the DVLA database, called ‘**DVLA Driver Eligibility’**. The input data is validated in the pre-request script for this query, and the response data is validated in the ‘Tests’ scripts for this query. The business rules are implemented as part of the ‘Tests’ scripts. However I have added another query, ‘**Driver Eligibility API Response’**, whose purpose is to show the JSON structure of a possible format for the API response to the ‘DVLA Driver Eligibility’ query. The hypothetical response format has been added in to the body of the request for that query to show it in JSON format. No tests are run as part of ‘Driver Eligibility API Response’, it is only to suggest one possible format for the response data.

**Rule 1**

This rule could use two category codes from the API response, say ‘transmissionType’ for the transmission type of the ordered vehicle and ‘licenceTransmission’ for the driver’s licence transmission type, both of which can be either ‘MANUAL’ or ‘AUTOMATIC’. The rule for this test would essentially take the form:

    if(pm.response.transmissionType == "MANUAL") {

        pm.expect(pm.response.drivers[0].licenceTransmission).to.eql("MANUAL");

    };

**Rule 2**

Could use one category code, say ‘fullOrProvisional’, taken from the driver data. It would take the values ‘FULL’ or ‘PROVISIONAL’. The rule for this test would essentially take the form:

pm.expect(pm.response.drivers[0].fullOrProvisional).to.eql("FULL");

**Rule 3**

Could use one category code, say ‘isDriverDisqualified’, which can be true or false. The rule would be:

pm.expect(pm.response.drivers[0].isDriverDisqualified).to.eql(false);

**Rule 4**

Could use category codes such as ‘endorsements\_category\_G’, ‘endorsements\_category\_H’, ‘endorsements\_category\_I’ which give integer counts of the numbers of endorsements within the last 4 years in each category. The rule would be:

*const* recentEndorsementsCount = pm.response.drivers[0].endorsements\_category\_G +\_

                                    pm.response.drivers[0].endorsements\_category\_H +\_

                                    pm.response.drivers[0].endorsements\_category\_I;

    pm.expect(recentEndorsementsCount).to.be.below(5);

**Rule 5**

This would use the ‘endorsements\_category\_G’ category code, which gives an integer count of the numbers of Category G endorsements within the last 4 years. It would also use another code, ‘latest\_conviction\_date\_G’ to determine the latest conviction date. The rule would be:

    pm.expect(pm.response.drivers[0].endorsements\_category\_G < 2  || pm.response.drivers[0].latest\_conviction\_date\_G.getFullYear() < currentYear - 4);

**Rule 6**

This would use the ‘endorsements\_category\_F’ category code, which gives an integer count of the numbers of Category F endorsements within the last 4 years. It would also use another code, ‘latest\_conviction\_date\_F’ to determine the latest conviction date. The rule would be:

    pm.expect(pm.response.drivers[0].endorsements\_category\_F == 0 || pm.response.drivers[0].latest\_conviction\_date\_F.getFullYear() < currentYear - 4);