

Testing with

HCL OneTest
Performance

and

HCL Commerce

Noureddine Brahimi

Senior Development Engineer, HCL Software

| TABLE OF CONTENTS | |
|---|----|
| SUMMARY | 2 |
| INTRODUCTION | 3 |
| DESIGN AND REUSE | 4 |
| 1 - Build tests (modules) for reuse | 4 |
| 2 - Global variables | 7 |
| 3 - Build a schedule | 7 |
| 4 - Clear cookie cache | 9 |
| 5 - Setup verification points against responses | 9 |
| HCL COMMERCE TEST SCENARIO: RANDOM BROWSE | 11 |
| Static browsing | 11 |
| Dynamic browsing | 17 |
| 1 - Add a custom code to pick one random top category | 23 |
| 2 - Add gson jar files to the project | 28 |
| 3 - Add a test on the output of the custom code | 30 |
| 4 - Add a loop | 31 |
| 5 - Add a custom code to pick a random sub-category | 32 |
| 6 - Add a products list page | 36 |
| 7 - Add a custom code to pick one product | 36 |
| 8 - Add a product details page | 38 |
| 9 - Add an ELSE to the first IF | 38 |
| 10 - Final script | 38 |
| 11 - Apply the 80/20 rule | 39 |
| PROBLEM DETERMINATION FOR ONETEST PERFORMANCE SCRIPTS | 41 |
| 1 - Examine references and substitutions in the test script | 41 |
| 2 - Set test behaviors when errors occur | 43 |
| CONCLUSION | 46 |

SUMMARY

This paper guides you through best practices for developing HCL OneTest Performance scripts and how to implement common scenarios for performance testing of HCL Commerce.

HCL OneTest Performance is a tool used to simulate website workloads to help measure application performance and identify bottlenecks.

The following is a step-by-step guide to go from a OneTest Performance novice to an expert.

The script design approach followed in this article will help you in your test coverage for e-Commerce applications and long-term re-use of the modules.

INTRODUCTION

E-commerce sites are widely used nowadays and it's about to become the new normal. Thus, the concern is now very high for web sites performance and reliability. To ensure that performance and reliability are well addressed, performance testing and monitoring should be conducted through tools such as HCL OneTest Performance against the application.

The HCL Commerce application is composed of several software applications and integrations and performance issues can be complex and multifaceted. The hardware resources used to deploy the application is another factor that needs to be considered as well.

Writing OneTest scripts for commerce requires not only understanding the basic steps of a simple shopping flow, like browsing to a product, adding it to the cart, checking it out and paying for it, but also the other flows that a shopper may go through while browsing on the storefront, like changing their shipping address, adding a new billing address, deleting items from the cart, and so on.

Once the script is written, you can define the volume of transactions to simulate real customer workload and it should be based on normal and peak business activity. This can be done in HCL OneTest by changing the schedule properties and using user groups, random selectors, and other features that will be discussed throughout this article.

Performance tests need to provide meaningful results that can be interpreted by stakeholders. Some relevant and common key performance indicators commonly used for e-Commerce are:



- ✓ Order throughput
- ✓ Response time
- ✓ Page views
- ✓ CPU utilization
- ✓ Cache hit ratio
- ✓ Database SQL activities

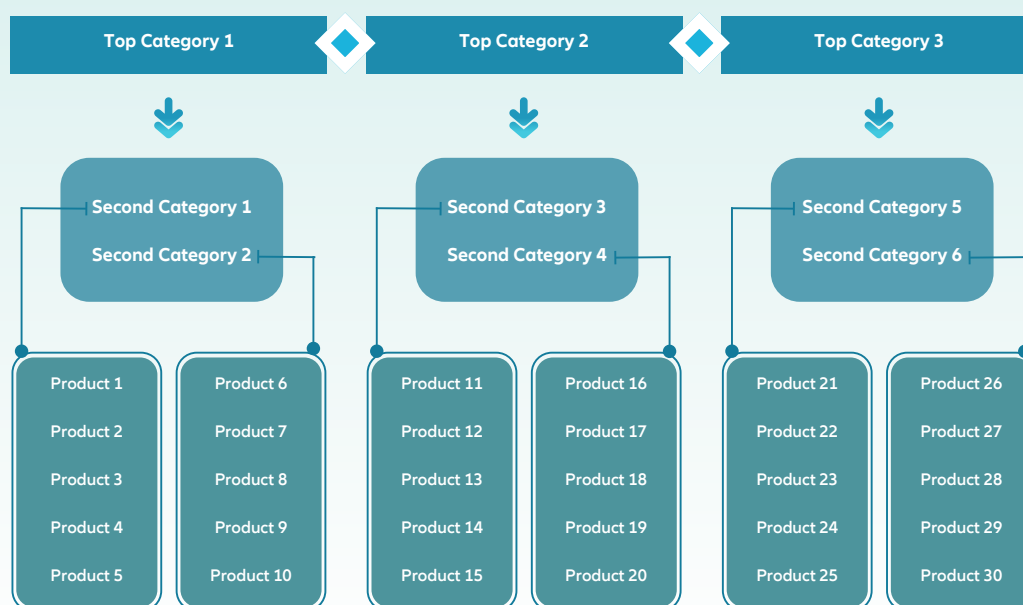
Orders processed per unit showcases the amount of revenue the solution can handle. The test workload can be increased to see if a greater order throughput is possible without exceeding constraints such as response time and hardware resources. Response time directly impacts customer satisfaction and has an inverse relationship to throughput. The number of page views is an indicator of how popular the website is regardless, if it is generating revenue or not. Some lower-level details such as CPU utilization, cache hit ratios and database SQL activities are of interest to architects and developers who need to know what to modify to achieve better performance. OneTest Performance provides reports which have many of these details. The reports can also be customized to help problem determination.

DESIGN AND REUSE

This section explains designing and building OneTest Performance scripts including:

1. Build tests (modules) for reuse.
2. Usage of global variables.
3. Build your schedule
4. Clear cookie cache
5. Setup verification points against responses

The HCL Commerce catalog that used in this paper is a balanced tree and has three first-level categories: each has two sub-categories, and each sub-category has five products as shown below.

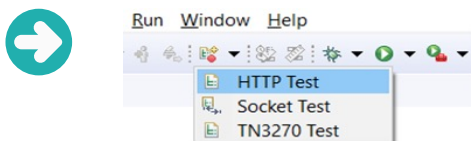


1 - Build tests (modules) for reuse

In the HCL OneTest Performance, almost all the time, building a performance script starts from recording the flow on the storefront.

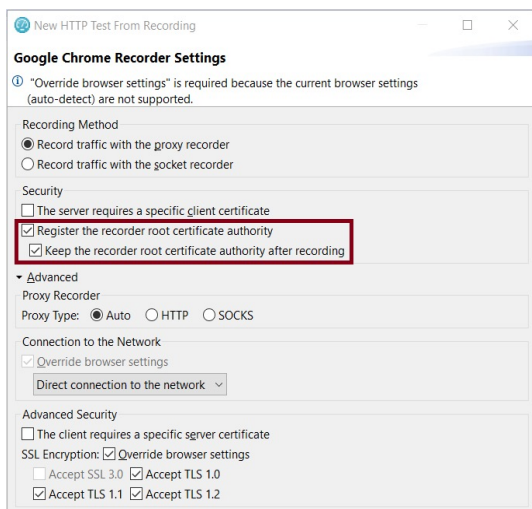
To record a flow against HCL Commerce React store:

- Select “New Test From Recording” button in the tool bar and pick “HTTP Test” as shown below



- Select the location and give the name of the recording
- Select the browser that you want to use

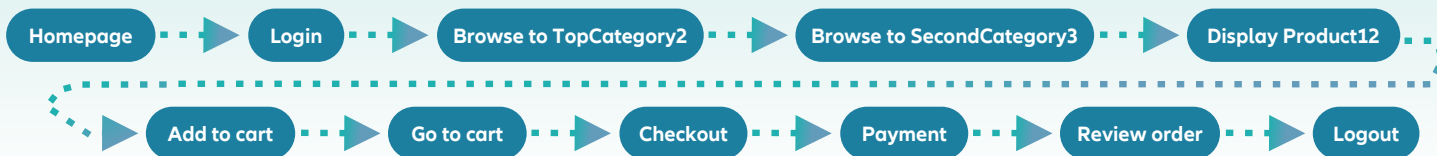
- Ensure that Register the recorder root certificate authority is selected as shown below:



and accept its installation after the above dialog box.

In this section, you will start by recording a registered user shopping flow session. Out of the recording, you will create reusable tests and use them to build a schedule.

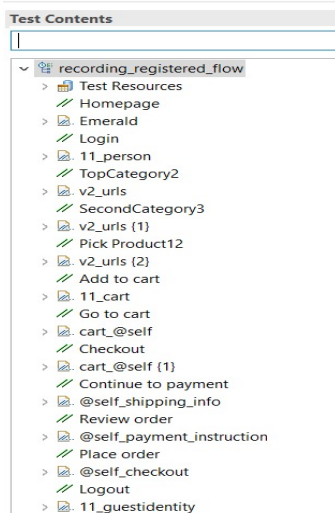
The recording goes through the following path



The test generated from the recording will look like the following:



Test - recording_registered_flow



The recording produces a set of pages that can be divided into a set of reusable tests or modules:

| Module to be created | Part of the recording that constitute the module |
|-----------------------------|---|
| Homepage | <ul style="list-style-type: none"> // Homepage > Emerald |
| Login | <ul style="list-style-type: none"> // Login > 11_person |
| BrowseAndPickProduct | <ul style="list-style-type: none"> // TopCategory2 > v2_urls // SecondCategory3 > v2_urls {1} // Pick Product12 > v2_urls {2} |
| AddToCart | <ul style="list-style-type: none"> // Add to cart > 11_guestidentity |
| GoToCart | <ul style="list-style-type: none"> // Go to cart > cart_@self |
| GoToCheckout | <ul style="list-style-type: none"> // Checkout > cart_@self {1} |
| GoToPayment | <ul style="list-style-type: none"> // Continue to payment > @self_shipping_info |
| ReviewOrder | <ul style="list-style-type: none"> // Review order > @self_payment_instruction |
| PlaceOrder | <ul style="list-style-type: none"> // Place order > @self_checkout |
| Logout | <ul style="list-style-type: none"> // Logout > 11_guestidentity |

The split need to be done carefully to produce modular tests. Adding comments during the recording helps in this task. If you want to add comments while recording, ensure annotation toolbar is installed before recording.

When you record a guest flow, you will notice that you will need to create a shipping address and thus, you may need to create a new module in addition to the one created above.

CreateNewAddress

- ✓ Create new address
- > person_@self
- > @self_contact

2 - Global variables

These tests will pass some information to each other. The data will be passed from one test to the next using global variables.

Global variables are set from Datasets, by direct variable assignments or by custom codes.

Setting all global variables in a separate test InitVariables is a good practice. By doing so, you will avoid changing variables' values inside each test when there is any need to update some variables' values. The changes need to happen only in InitVariables test and all the other modules will the new values.

The global variables that will be used throughout the whole flow are: hostname, store_id, catalog_id, and lang_id



- ▼ InitVariables
 - ▼ Test Resources
 - ▼ Test Variables
 - hostname = "www.your-e-commerce-site.com"
 - store_id = "11"
 - catalog_id = "11001"
 - lang_id = "-1"
 - > Server Access Configurations

The hostname is the URL of the storefront. The store_id and catalog_id can be fetched from the HCL Commerce database or provided by the e-Commerce site administrator. The lang_id is the default language ID, -1, which is English US.



Name:

Visible in:

The visibility of these global variables must be "All tests for this user" to make them global.

3 - Build a schedule

In previous section, you created a set of modules and in this section, you will group them in a schedule that will be used to run a test.

To organize things out, it's a good practice to create some folders to store schedules, modules, recording and even test results. Custom codes will be stored automatically under src folder which is created by default when you create the performance project Modularization.



- ▼ Modularization
 - > src
 - > Recording
 - > Schedules
 - ▼ Tests
 - AddToCart
 - Browse
 - CreateNewAddress
 - GoToCart
 - GoToCheckout
 - GoToPayment
 - Homepage
 - InitVariables
 - Login
 - Logout
 - PlaceOrder
 - ReviewOrder

After modularizing test scripts or modules as described earlier, you can sequence them in a schedule under transaction containers. You can create different sequences or transactions depending on what scenario is being simulated.

For example, you create a VU Schedule. Notice that it comes with one User Group, under which, you add a loop through the Add context menu or Add button. You add a Random Selector with 2 blocks, one for Guest Users and the other for Registered Users. Under each bloc, you create a transaction and add its corresponding modules as shown below:



- ▼ ShoppingFlow
 - ▼ User Group 1 (100%)
 - Loop (infinite): Loop1
 - ▼ Random Selector (2 blocks)
 - ▼ Registered User (1) (50%)
 - ▼ Transaction: Registered user
 - InitVariables
 - Homepage
 - Login
 - Browse
 - AddToCart
 - GoToCart
 - GoToCheckout
 - GoToPayment
 - PlaceOrder
 - ReviewOrder
 - Logout
 - ▼ Guest User (1) (50%)
 - ▼ Transaction: Guest user
 - InitVariables
 - Homepage
 - Browse
 - AddToCart
 - GoToCart
 - GoToCheckout
 - CreateNewAddress
 - GoToPayment
 - PlaceOrder
 - ReviewOrder

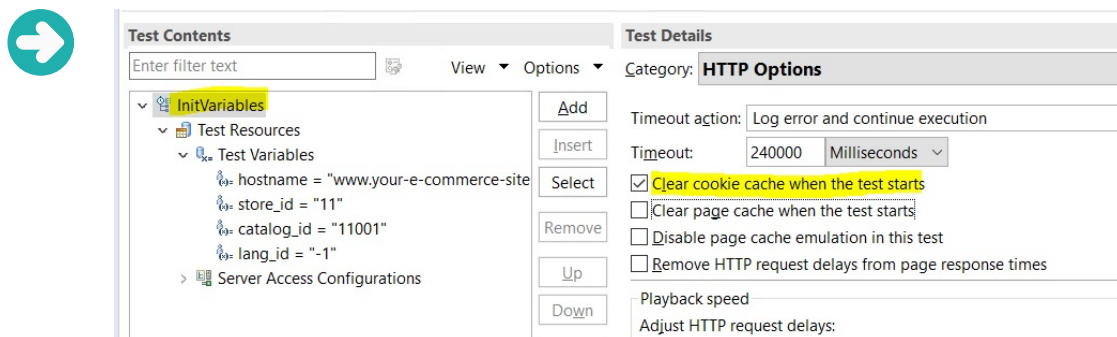
The Transaction blocks under each selector can be used in reporting. You can report a response time for guest or registered user shopping flow for example. They can also be used to define the rules when Verification Point fails as you will see later in this section

You can define more than one User Group in a schedule. You can assign a percentage or absolute quantity of virtual users to a User Group.

4 - Clear cookie cache

The very first test in a schedule should clear the cookie cache as it should simulate a fresh browser session for a user. Ensure all other modules do not clear the cookies to avoid having connection issues and users losing their cookies.

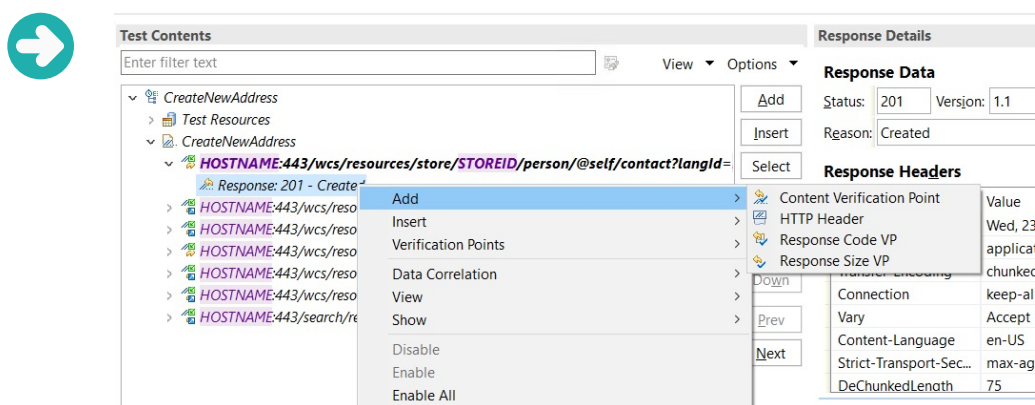
The cookie cache is cleared as shown below:



5 - Setup verification points against responses

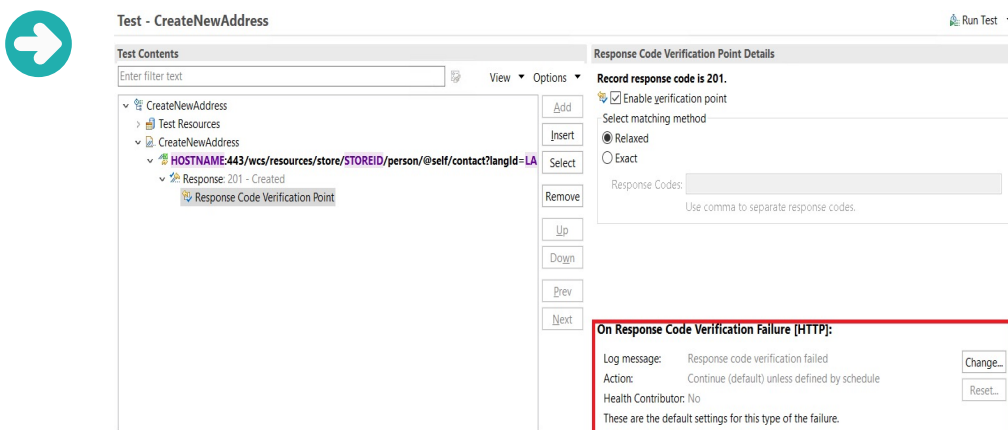
Verification points (VPs in short) are used to ensure a request is successful and its response is valid without errors or unexpected content and can be consumed without issues by either substituting some of the response's values in subsequent requests or using it in a custom code as an argument.

There are three types of verification points: response size, response code and response content. They can be created against a response through the Add context menu.

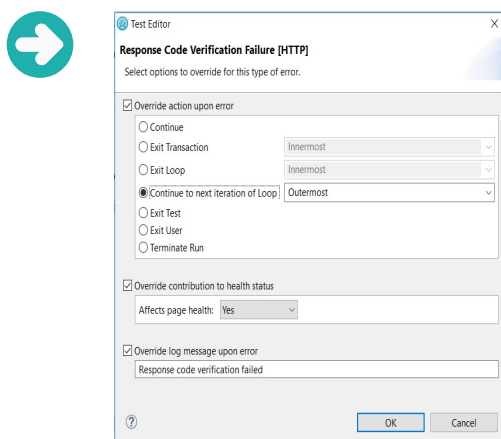


Once the VP is created, you can define what to do when the VP fails.

As an example, for the following Response Code VP, you can define the actions to do when the VP fails as highlighted below:



To force the virtual user to stop its session and go up to Loop1 as a new user in the schedule above, pick the following:



Verification points can be monitored during a run under the Summary tab. It will provide real-time information about the correctness of the run.

| Page Element Summary | |
|--|-----------|
| Counters | |
| Total Page Element VPs Passed | 1,091,269 |
| Total Page Element VPs Failed | 0 |
| Percent Page Element VPs Passed | 100.0% |
| Total Page Element VPs Inconclusive | 0 |
| Total Page Element VPs Error | 0 |
| Total Page Element Attempts | 3,155,475 |
| Total Page Element Hits | 3,155,466 |
| Total Page Elements Fresh In Cache, Not Sent to Server | 0 |
| Response Time For All Page Elements - Average | 24.87ms |
| Response Time For All Page Elements - Standard Deviation | 27.35ms |

HCL COMMERCE TEST SCENARIO: RANDOM BROWSE

This section describes the ways to implement browsing HCL Commerce catalog using HCL OneTest Performance.

Throughout this section, a full example from the design to the implementation of the test will be presented. The example will be based on the Emerald Out of the box store.

A goal of a random browse is to traverse the catalog hierarchy and land on a product page.

Browsing begins at top-level categories and continues to sub-categories until a product page is reached. Another way to browse to a product page is to use facets which allows the user to navigate directly to products. Faceted navigation basically filters the products by attributes such as color, size, type, etc and displays only the products that match the selected attributes.

To better utilize caching, the 80/20 rule is implemented.

This rule ensures that 20% of the catalog products are picked 80% of the time. This way, there will be many revisited catalog and product pages and thus the cache hit ratio will be more realistic. This rule works better when the catalog tree is well balanced, i.e. contain equal numbers of subcategories and products under each navigation path. It's recommended to make it so in testing environments.

In HCL OneTest Performance, there are two ways to implement random browsing: static which is data driven and dynamic browsing which is web crawling.

Static browsing

Static browsing uses a fixed and pre-defined number of requests. An example of that would be several requests to browse to the homepage, top category, sub-category and product pages.

Request parameters need to be substituted using actual category/product IDs and names. This means that the IDs and names (categories and products) need to be known ahead of time before test execution. Datasets can be used to store the IDs and names, so they can be used to substitute the requests' parameters.

Based on the catalog tree mentioned above, one dataset can be used for top categories, another one for second categories and another one for products. All of them have 2 columns, one for the names and one for the IDs.

TopCategoriesDataset

| | name | id |
|---|--------------|----|
| 1 | TopCatgeory1 | 1 |
| 2 | TopCategory2 | 2 |
| 3 | TopCategory3 | 3 |

SubCategoriesDataset

| | name | id |
|---|---------------------|----|
| 1 | SecondCatego ry1 | 4 |
| 2 | SecondCatego ry2 | 5 |
| 3 | SecondCatego ry3 | 6 |
| 4 | SecondCatego ry4 | 7 |
| 5 | SecondCatego ry5 | 8 |
| 6 | SecondCatego ry6 | 9 |

ProductsDataset

| | name | id |
|---|-----------|-----|
| 1 | Product1 | 1 |
| 2 | Product2 | 2 |
| 3 | ... | ... |
| 4 | ... | ... |
| 5 | Product29 | 29 |
| 6 | Product30 | 30 |

From the recording, the top category, sub category and product details will have the following requests. The screenshots show how the variables highlighted in green are substituted from the datasets TopCategoriesDataset, SecondCategoriesDataset and ProductsDataset.



▼ TopCategory

- > `hostname:443/search/resources/api/v2/urls?identifier=TOPCATEGORYNAME&storeId=STOREID&langId=LANGID`
- > `hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&identifier=TOPCATEGORYNAME&storeId=STOREID&langId=LANGID`
- > `hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&parentCategoryId=CATEGORYID&storeId=STOREID&langId=LANGID`

| | name | id |
|---|--------------|----|
| 1 | TopCatgeory1 | 1 |
| 2 | TopCategory2 | 2 |
| 3 | TopCategory3 | 3 |

▼ SecondCategory

- > `hostname:443/search/resources/api/v2/urls?identifier=SECONDCATEGORYNAME&storeId=STOREID&langId=LANGID`
- > `hostname:443/search/resources/api/v2/products?categoryId=CATEGORYID&contractId=CONTRACTID¤cy=USD&limit=12&offset=0&storeId=STOREID&langId=LANGID`

| | name | id |
|---|---------------------|----|
| 1 | SecondCatego ry1 | 4 |
| 2 | SecondCatego ry2 | 5 |
| 3 | SecondCatego ry3 | 6 |
| 4 | SecondCatego ry4 | 7 |
| 5 | SecondCatego ry5 | 8 |
| 6 | SecondCatego ry6 | 9 |



Product Details

- hostname:443/search/resources/api/v2/urls?identifier=PRODUCTNAME&storeId=STOREID&langId=LANGID
- hostname:443/search/resources/api/v2/products?catalogId=CATALOGID&contractId=CONTRACTID&partNumber=PRODUCTNAME&storeId=STOREID&langId=LANGID
- hostname:443/wcs/resources/store/STOREID/associated_promotion?q=byProduct&qProductId=PRODUCTID&langId=LANGID
- hostname:443/search/resources/api/v2/products?categoryId=CATEGORYID&contractId=CONTRACTID¤cy=USD&storeId=STOREID&langId=LANGID
- hostname:443/wcs/resources/store/STOREID/inventoryavailability/PRODUCTID&langId=LANGID

| | name | id |
|---|---------------------|----|
| 1 | SecondCatego zy1 | 4 |
| 2 | SecondCatego zy2 | 5 |
| 3 | SecondCatego zy3 | 6 |
| 4 | SecondCatego zy4 | 7 |
| 5 | SecondCatego zy5 | 8 |
| 6 | SecondCatego zy6 | 9 |

| | name | id |
|----|-----------|----|
| 1 | Product1 | 1 |
| 2 | Product2 | 2 |
| 3 | Product3 | 3 |
| 4 | Product4 | 4 |
| 5 | Product5 | 5 |
| 6 | Product6 | 6 |
| 7 | Product7 | 7 |
| 8 | Product8 | 8 |
| 9 | Product9 | 9 |
| 10 | Product10 | 10 |

The static browse that will be implemented is that few users browse to a home page and then leave, few others would continue to one of the top categories and leave and another group of users would continue up to one of the sub-categories and leave and the last group would go to the product page and leave.

To implement this behavior, random selector will be used like the following:

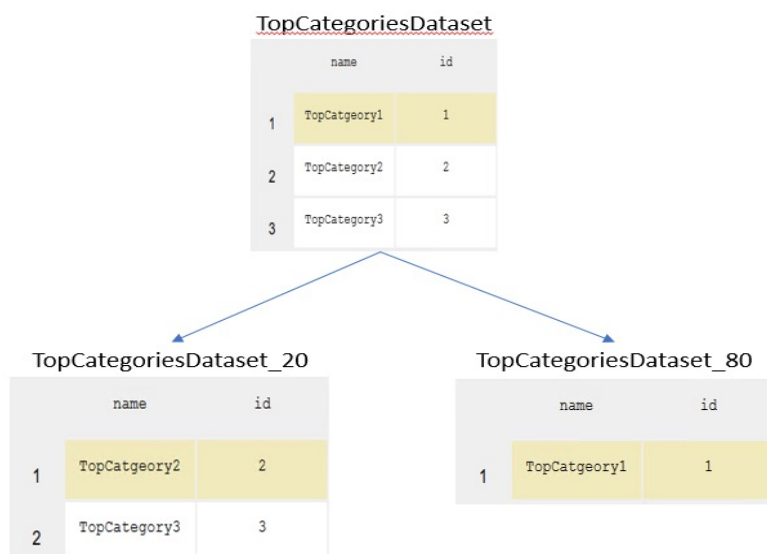


cBrowse

- ast Resources
- andom Selector (4 blocks)
 - Homepage (1) (25%)
 - Homepage
 - TopCategoryDisplay (1) (25%)
 - Homepage
 - TopCategory
 - SecondCategoryDisplay (1) (25%)
 - Homepage
 - TopCategory
 - SecondCategory
 - ProductDisplay (1) (25%)
 - Homepage
 - TopCategory
 - SecondCategory
 - ProductDetails

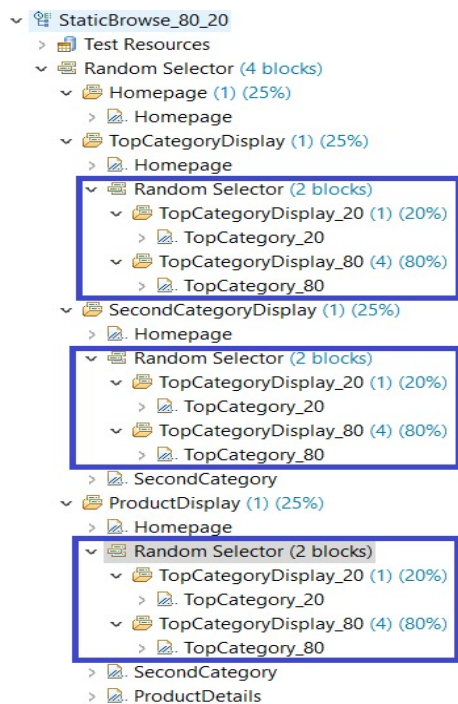
To implement the 80/20 rule, you need to make 20% of top categories are picked 80% of the time. Since there are 3 top categories (TopCategory1, TopCategory2 and TopCategory3), you will make sure that 80% of the time the 1st top category TopCategory1 is picked and the 20% remaining of the time, one of the 2 remaining top categories (TopCategory2 and TopCategory3) are picked randomly.

One way to implement this is to split TopCategoriesDataset to two datasets, TopCategoriesDataset_80 and TopCategoriesDataset_20, one will contain only TopCategory1 and the other contains the remaining top categories respectively.



As you used Random Selectors above with equal weights, you can use the same thing with 80% weight to pick TopCategory1 and 20% weight to pick one of the remaining top categories. You need also to copy TopCategory test to TopCategory_20 that picks its data from TopCategoriesDataset_20 and TopCategory_80 that picks its data from TopCategoriesDataset_80.

The script become like the following:



TopCategory page was updated to become the highlighted part of the script above.

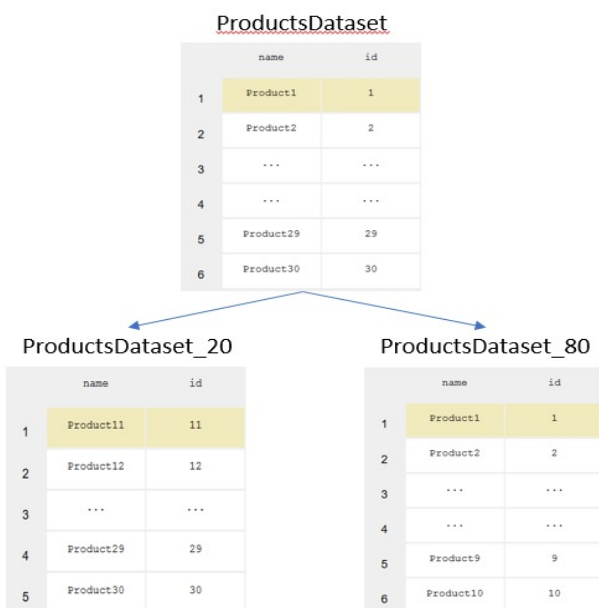
This static browsing, as it is implemented now by this script, can make a virtual user browse to a top category and then to a second category that doesn't belong to this top category and then to a product that doesn't belong to this sub category. Because of this behavior, the 80/20 rule is not respected.

For this rule to be respected with our catalog described at the beginning of this section, TopCategory1 should be picked 80% of time as well as everything under this top category.

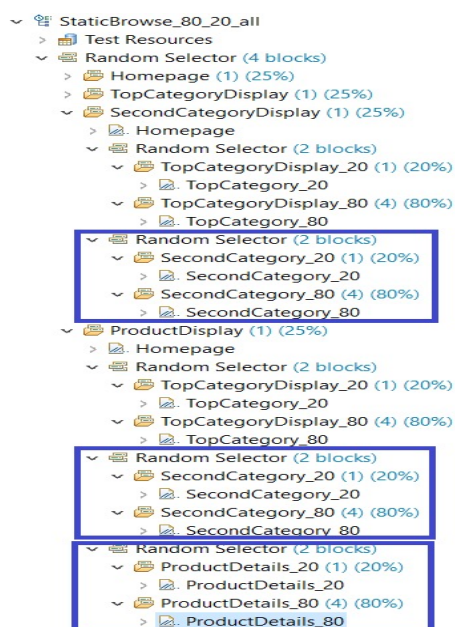
To implement this and respect the 80/20 rule, you will need to split the sub categories and products Datasets and add Random Selectors in the script the same way it was done with browsing to top categories above.

After splitting the datasets SubCategoriesDataset and ProductsDataset, they will look like the following:





You also need to split the pages SecondCategory and ProductDetails and link them to the 2 new datasets. The script will become like the following:



SecondCategory and ProductDetails pages were updated to become the highlighted part of the script above.

The big advantage of static browsing is the ease of implementation. The tester doesn't need to create any regular expression to get a product or a category ID for example and no need to implement any custom code to start with. It's a very quick way to get response time for browsing categories and products.

The disadvantages of static browsing can be summarized in these points:

- 1 - In static browsing, the chosen categories to the end product may not be truly linked.
- 2 - Static browsing is linked to the data which means changing the data implies changing the script. By adding a new category or deleting one, the tester needs to add or delete blocks in the script and by adding new products; the tester will need to update the Datasets.

These disadvantages make the script not portable as it cannot be used to test another commerce catalog without updates.

Dynamic browsing

In this kind of dynamic browsing, there is no need to know the catalog hierarchy, the products or category IDs to build your script. This kind of browsing avoids all the disadvantages of the previous static browsing.

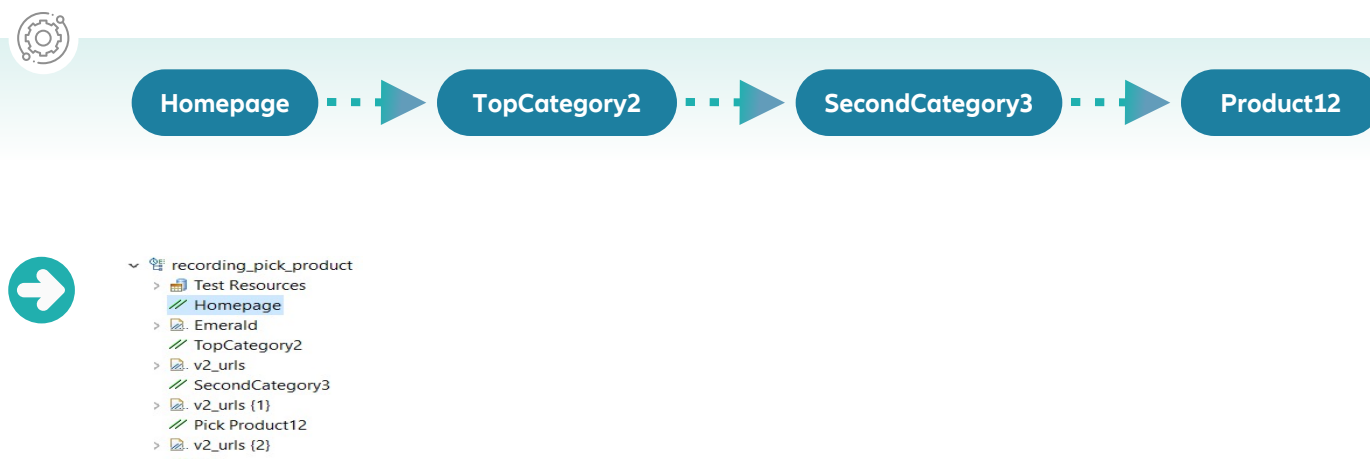
The way of doing dynamic browsing is to pick randomly one top category and browse to it. Get all its sub-categories and pick one of them randomly. Repeat this process until products list page is reached and where one random product is also picked randomly.

Throughout this dynamic browsing, the same catalog defined in the second section of this article will be used.

In this section, you will get step by step from a recording to a complete dynamic browse module

As mentioned before, building a script almost always starts with a recording.

The following is a recording of a guest browse



To make this guide easy to follow, under each page, extra requests will be deleted like JavaScript files, json, espot, and picture requests. Only the necessary requests for each page will be kept.

Some needed variables will be created. They will be used to substitute requests' parameters.

The page names will also be updated.

A snippet of the script looks like the following:



```

DynamicBrowsing_1bis
├── Test Resources
│   ├── Test Variables
│   │   ├── hostname = ""
│   │   ├── store_id = ""
│   │   ├── catalog_id = ""
│   │   ├── lang_id = ""
│   │   ├── contract_id = ""
│   │   ├── category_name = ""
│   │   ├── category_id = ""
│   │   ├── product_name = ""
│   │   └── product_id = ""
│   ├── Server Access Configurations
│   └── Homepage
│       ├── hostname:443/Emerald/
│       ├── hostname:443/wcs/resources/store/11/contract?q=eligible&langId=-1
│       └── hostname:443/search/resources/api/v2/categories?contractId=-11005&depthAndLimit=11,11&storeId=11&langId=-1
└── TopCategory
    ├── hostname:443/search/resources/api/v2/urls?identifier=topcategory2&storeId=11&langId=-1
    └── hostname:443/search/resources/api/v2/categories?contractId=-11005&identifier=TopCategory2&storeId=11&langId=-1

```

After renaming the requests' parameters (uppercase) and substituting all them using the corresponding created variables, the script will look like the following



```

DynamicBrowsing_1bis1
├── Test Resources
│   ├── Test Variables
│   │   ├── hostname = ""
│   │   ├── store_id = ""
│   │   ├── catalog_id = ""
│   │   ├── lang_id = ""
│   │   ├── contract_id = ""
│   │   ├── category_id = ""
│   │   ├── category_name = ""
│   │   ├── product_id = ""
│   │   └── product_name = ""
│   ├── Server Access Configurations
│   └── Homepage
│       ├── hostname:443/Emerald/
│       ├── hostname:443/wcs/resources/store/STOREID/contract?q=eligible&langId=LANGID
│       └── hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&depthAndLimit=11,11&storeId=STOREID&langId=LANGID
└── TopCategory
    ├── hostname:443/search/resources/api/v2/urls?identifier=TOPCATEGORYNAME&storeId=STOREID&langId=LANGID
    ├── hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&identifier=TOPCATEGORYNAME&storeId=STOREID&langId=LANGID
    ├── hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&parentCategoryId=TOPCATEGORYID&storeId=STOREID&langId=LANGID
└── SecondCategory
    ├── hostname:443/search/resources/api/v2/urls?identifier=SECONDCATEGORYNAME&storeId=STOREID&langId=LANGID
    └── hostname:443/search/resources/api/v2/products?categoryId=SECONDCATEGORYID&contractId=CONTRACTID&currency=USD&limit=12&offset=0&storeId=STOREID&langId=LANGID
ProductDetails
├── hostname:443/search/resources/api/v2/urls?identifier=PRODUCTNAME&storeId=STOREID&langId=LANGID
├── hostname:443/search/resources/api/v2/products?catalogId=CATALOGID&contractId=CONTRACTID&partNumber=PRODUCTNAME&storeId=STOREID&langId=LANGID
├── hostname:443/wcs/resources/store/STOREID/associated_promotion?q=byProduct&qProductId=PRODUCTID&langId=LANGID
└── hostname:443/wcs/resources/store/STOREID/inventoryavailability/PRODUCTID?langId=LANGID

```

Variable `contract_id` will be initialized using the response of the 2nd request, under Homepage.

The response of that request contains the contract id.

- First, you need to create a reference to the id



The screenshot shows the HCL OneTest Performance interface. On the left, a tree view displays the test structure, including 'DynamicBrowsing_1bis1', 'Test Resources', 'Test Variables', 'Server Access Configurations', and 'Homepage'. The 'Homepage' folder is expanded, showing a list of requests. The 'Response: 200 - OK' is selected. On the right, the 'Response Headers' tab is active, displaying a table of headers:

| Header Name | Value |
|-------------------|--------------------|
| Date | Tue, 22 Dec 2020 2 |
| Content-Type | application/json |
| Transfer-Encoding | chunked |
| Connection | keep-alive |

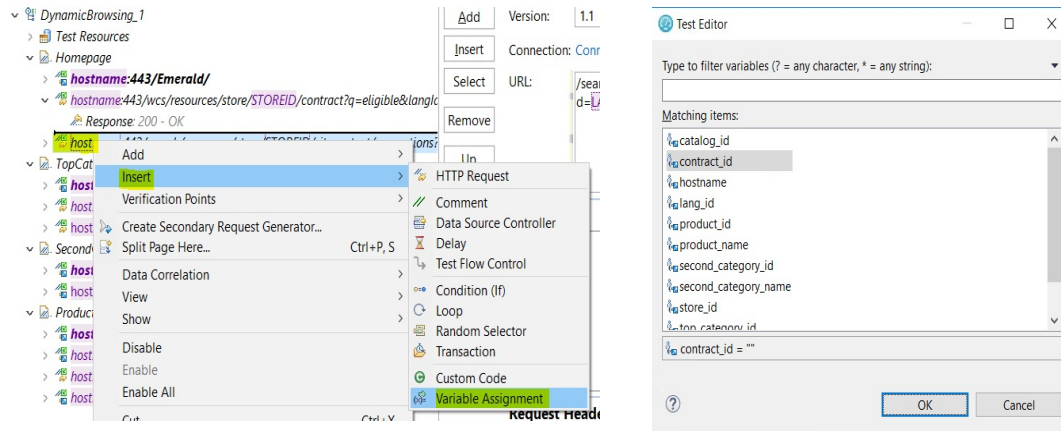
Below the headers, the 'Content' tab is active, showing the response body. A context menu is open over the response body, with the 'Create Reference' option highlighted.

and give it a name



The screenshot shows the 'Properties for Reference' dialog box. The 'Reference' tab is active. The 'Name' field is set to 'contract_id'. The 'Regular Expression' field is set to '\d{1,6}'. The 'Occurrence' section is set to 'Specific occurrence number: 1'. The 'Preview' tab is active, showing the response body with the value '11005' highlighted. The 'Apply and Close' button is visible at the bottom.

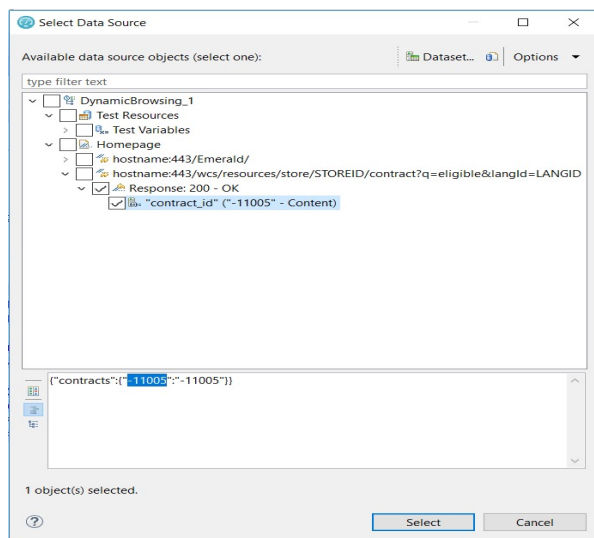
- Second you add a variable assignment right after the request and pick contract_id variable



- Set the value of contract_id from Data Source Value



- Pick the contract_id reference that you defined earlier and click Select



- Now each substitution using contract_id in subsequent requests will use the right contract id from the response content.



If you run this test, it will always go through the same path. To update this test and make it dynamic and random, you need to follow the following steps to transform the recording above to a dynamic browsing:

A pseudo code of the recording that you did above:



is shown in the following steps (R stands for Recording)

R1 - Browse to the Homepage

R2 - Browse to top category TopCategory2

R3 - Browse to second category SecondCategory3

R4 - Pick product Product12 and show its details.

To write a general script that browses any Commerce catalog, the above R steps of the recording need to be transformed:

- Step R1 is kept.
- Step R2 is updated to get one random category from the list of all top categories
- Step R3 is updated by adding a loop where one random sub-category is picked under the top category already picked in previous step and then check
 - o If the current category has sub-categories. Loop again
 - o Else, products list page is reached. Break out of the loop.
- Pick a random product from the list of products

This table summarizes the previous steps to transform a browse from a recording to a dynamic pseudo script (R stands for Recording and S stands for Script in the table):

| <i>Recording steps</i> | <i>Pseudo code of a dynamic script</i> |
|------------------------|---|
| R1 – Homepage | S1 – Browse to the Homepage |
| R2 – TopCategory2 | S2 – Pick one top category from the list of top categories |
| R3 – SecondCategory3 | S3 – Loop <ul style="list-style-type: none"> S4 – Browse to category picked S5 – Get the list of sub-categories S6 – If the list is not empty <ul style="list-style-type: none"> S7 – Pick one random sub-category and browse to it S8 – Else <ul style="list-style-type: none"> S9 – BreakLoop |
| R5 – Product12 | S10 – Products List Page S11 – Pick one random product from the list and display it |

The following pseudo-code is the same as the one on the right side of the previous table with some additions that will be explained a bit later. It is written similarly to HCL OneTest Performance scripts.

| <i>PseudoCode1 - Dynamic browse module script</i> |
|---|
| <pre> 1 Page: Browse to Homepage 2 CustomCode: PickOneTopCategory() 3 If category_picked not empty 4 Loop (infinite) 5 Page: Browse to Category 6 CustomCode: GetOneNextCategory 7 If We're on a Products List page 8 Breakloop 9 Page: Browse to Products List Page 10 CustomCode: PickOneProduct 11 Page: Browse to Product 12 Else 13 Display error and Exit </pre> |

The same way a pseudo-code of the recording is transformed to a pseudo-code script, the recorded script will be transformed to a dynamic browsing script.

Throughout the remaining part of this section, the above numbered lines of PseudoCode1 will be referenced to show the current line that is going to be updated/transformed:

1 - Add a custom code to pick one random top category

As shown in the PseudoCode1, right after the Homepage, at step 2, you will create a custom code to fetch the list of top categories and pick one randomly.

You get the list of top categories from the last request under Homepage.

```
GET /search/resources/api/v2/categories?contractId=CONTRACTID&depthAndLimit=11%2C11&storeId=STOREID&langId=LANGID
```

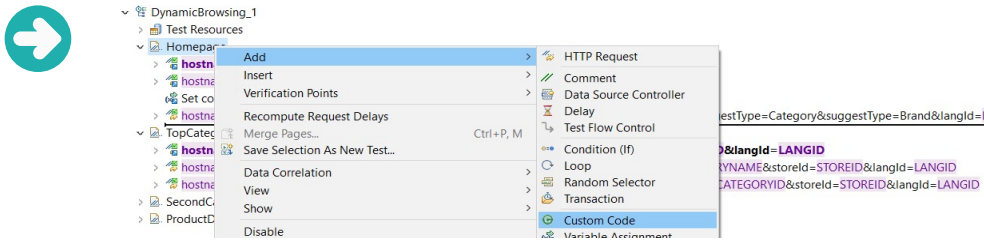
A snippet of the JSON response content of this request looks like this (children and links sections are collapsed since we don't need them now):



```
{
  "contents": [
    {
      "seo": {
        "href": "/topcategory1"
      },
      "name": "TopCategory1",
      "identifier": "TopCategory1",
      "storeId": "11",
      "sequence": "1.0",
      "children": [
        {
          "id": "1",
          "links": {

```

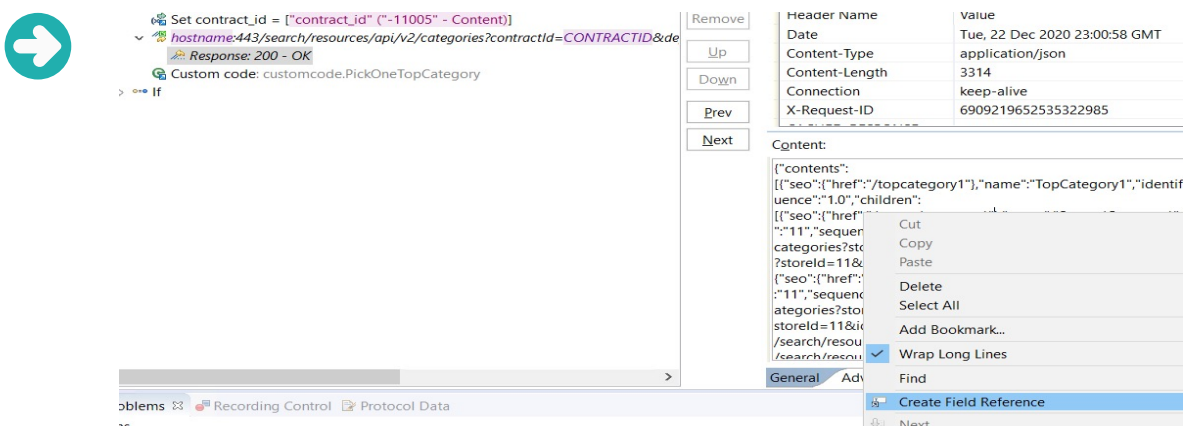

Create the custom code like the following – since the custom code will be created under Homepage, right click the title Homepage as shown below:



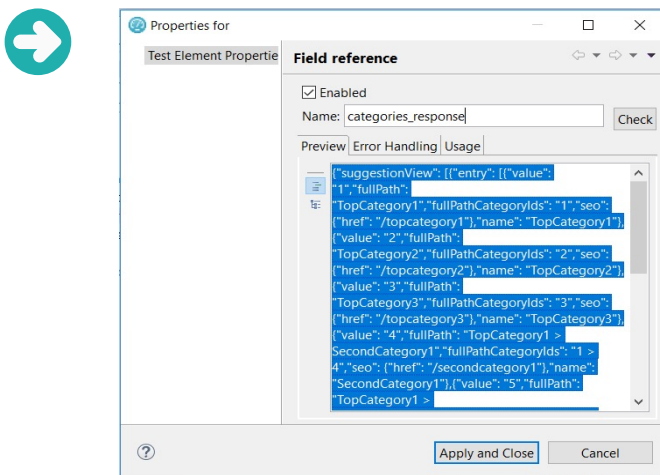
The custom code parses the request's JSON response content and gathers top category ids and names in a list and pick one random top category from that list.

The response content of the request will be made an argument to the custom code.

First you need to create a field reference from the response content:

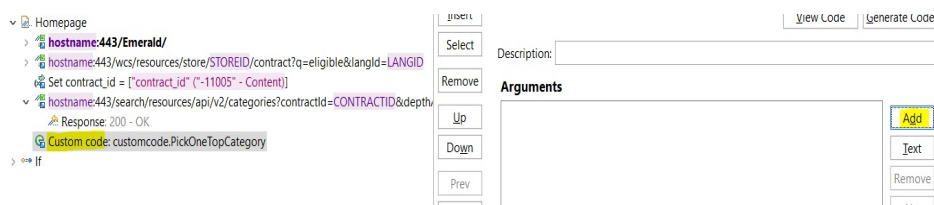


Give it a name and click Apply and Close button:

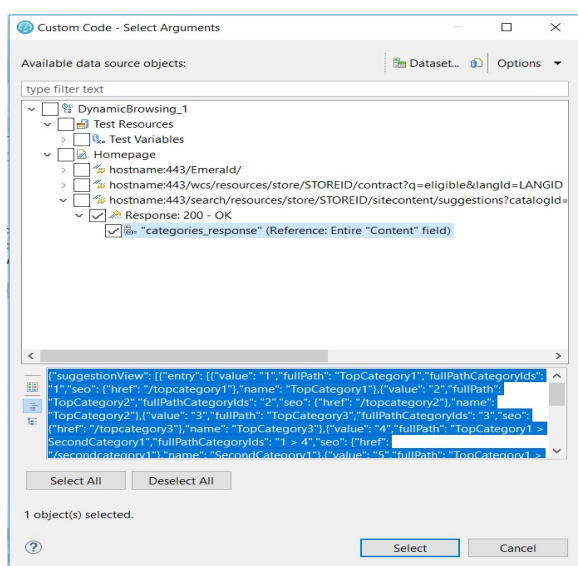


Once the response is set a field reference, you can reference it as an argument to the custom code.

To do so select the custom code and click on Add button:



On the dialog box, select the field reference you just defined above and click on Select button:



Once this is done, the field reference will show up in the list of arguments of the custom code.

The category id value is assigned to variable `category_id` and name is assigned to `category_name`. These variables need to be created in the module variables and their visibility set to "All test for this user"

If a top category is picked, 0 is returned, otherwise, -1 is returned.

The custom code uses google gson library to parse JSON response content and get the category id (value in the JSON response) and name.

The complete code:

```
package customcode;

import java.util.Iterator;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonStreamParser;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

public class PickOneTopCategory implements com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    public PickOneTopCategory() {}

    public String exec(ITestExecutionServices tes, String[] args) {

        String json_str = args[ 0 ];

        JsonStreamParser p = new JsonStreamParser(json_str);

        JsonElement e;

        if ( p.hasNext() ) {

            e = p.next();

        }

        else {

            tes.getTestLogManager().reportMessage( "Error in the request's response" );

            return "-1";

        }

        JsonObject suggestionView = e.getAsJsonObject();

        JsonArray OneSuggestionView = suggestionView.get( "suggestionView" ).getAsJsonArray();

        JsonElement elem;

        Iterator<JsonElement> iterator = OneSuggestionView.iterator();

        String id, name;

        String ids = "", names = "";

        elem = iterator.next();

        JsonObject entry = elem.getAsJsonObject();
```

```

JSONArray OneEntry = entry.get( "entry" ).getAsJSONArray();

JsonObject json_object;

Iterator<JsonElement> entry_iterator = OneEntry.iterator();

while ( entry_iterator.hasNext() ) {

    elem      = entry_iterator.next();

    json_object = elem.getAsJsonObject();

    id = json_object.get( "id" ).getString();

    name = json_object.get( "name" ).getString();

    ids += id + ",";

    names += name + ",";

}

if ( names.length() == 0 ) {

    tes.getTestLogManager().reportMessage( "No top categories found" );

    return "-1";

}

// Delete the last comma

ids = ids.substring( 0, ids.length() - 1 );

names = names.substring( 0, names.length() - 1 );

String [] a_ids = ids.split( "," );

String [] a_names = names.split( "," );

Int i = (int) Math.floor( Math.random() * a_ids.length );

tes.getTestLogManager().reportMessage( "i = " + String.valueOf( i ) );

tes.setValue( "category_name", ITestExecutionServices.STORAGE_USER, a_names[ i ] );

tes.setValue( "category_id", ITestExecutionServices.STORAGE_USER, a_ids[ i ] );

tes.getTestLogManager().reportMessage( "Category picked = " + a_names[ i ] );

return "0";

}

}

```

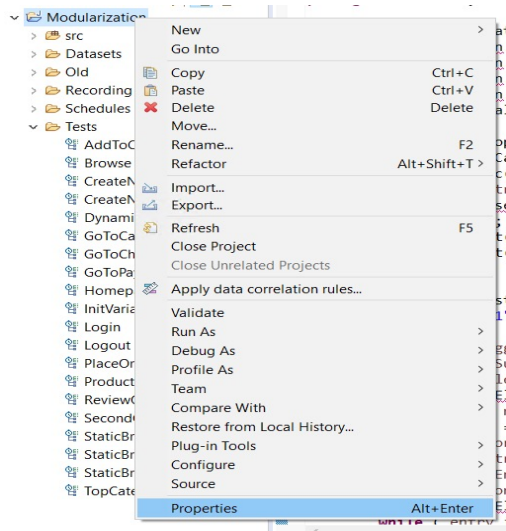
2 - Add gson jar files to the project

You will notice that there are errors in the custom code complaining about the imported external library

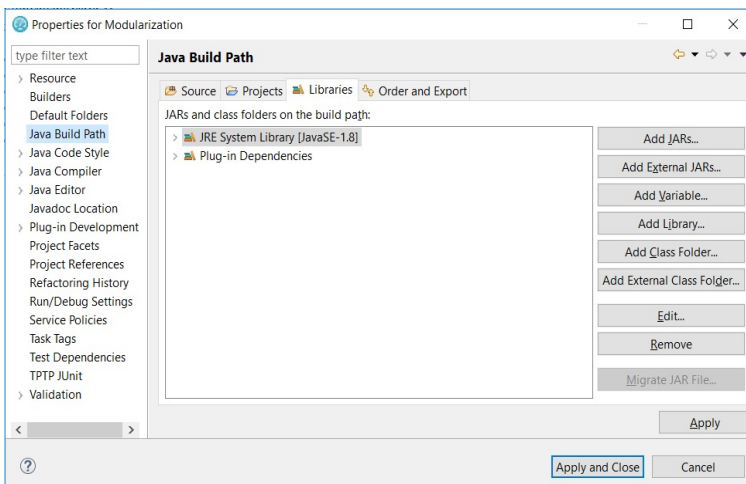


```
import java.util.Iterator;  
import com.google.gson.JsonArray;  
import com.google.gson.JsonElement;  
import com.google.gson.JsonObject;  
import com.google.gson.JsonStreamParser;
```

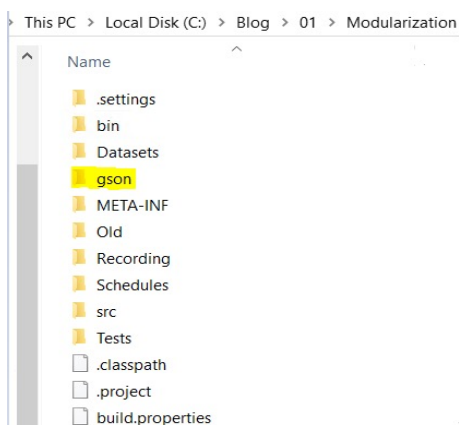
- To add this library, you need to download its jar files to a local folder:



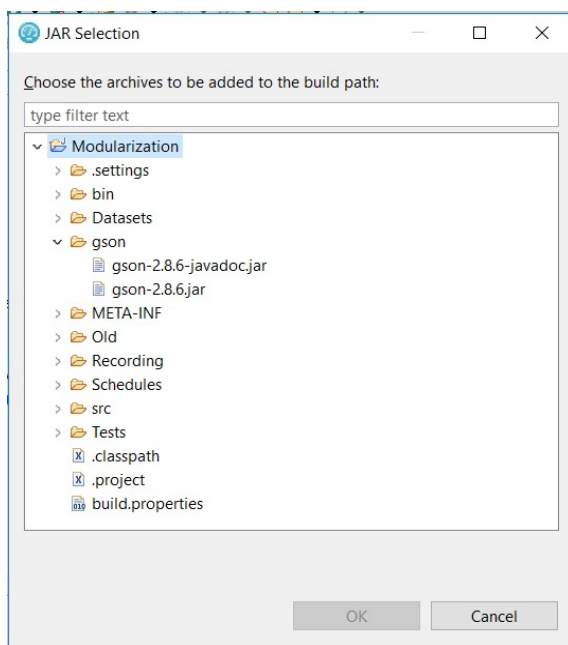
- Right click on the project name and choose Properties
- On the dialog box, choose Java Build Path on the right side, then choose Libraries tab,



and finally click on the button Add JARs if you created the folder under the workspace project or click on Add External JARs if you created the folder outside the workspace:



As you can see, the folder highlighted was created under the workspace, so the button Add JARs will be chosen:



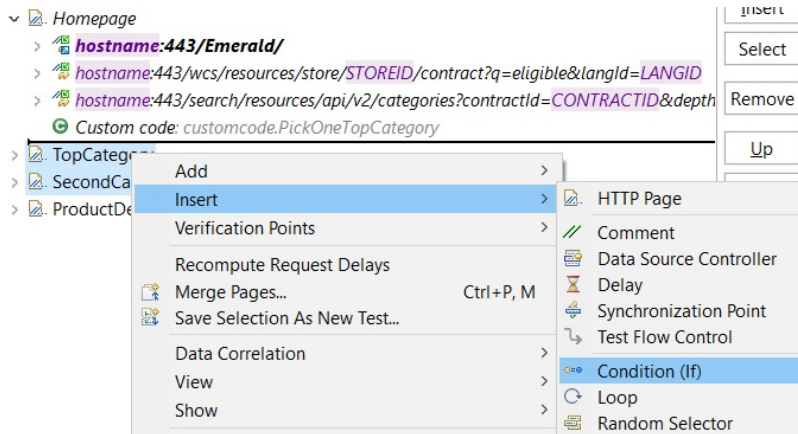
Expand the gson folder, select the jar files, and click on OK button.

- The jar files will show up in the list of libraries. Click on Apply and Close button

3 – Add a test on the output of the custom code

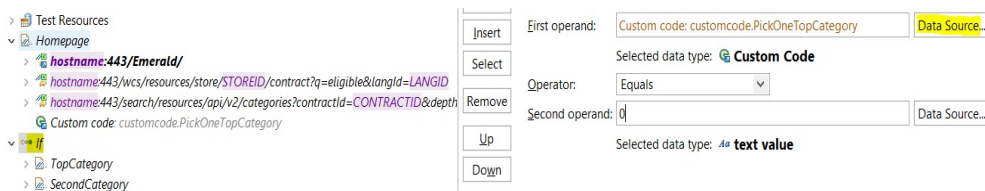
You will apply step 3 in PseudoCode1, there is a test on the output of the custom code. To do so you need to follow those steps:

- Select TopCategory and SecondCategory pages and right click and select Condition (If) command on the contextual menu, then click Yes on the dialog box. Once this is done, the 2 pages will be under the If:



- Once the IF is created, you need to define its first and second operand values.

The 1st operand is the output of the custom code. You need to click the highlighted Data Source button and select the custom code. The 2nd operand is 0. This means that the if the custom code returns zero, then go through the 2 pages under the If.



4 – Add a loop

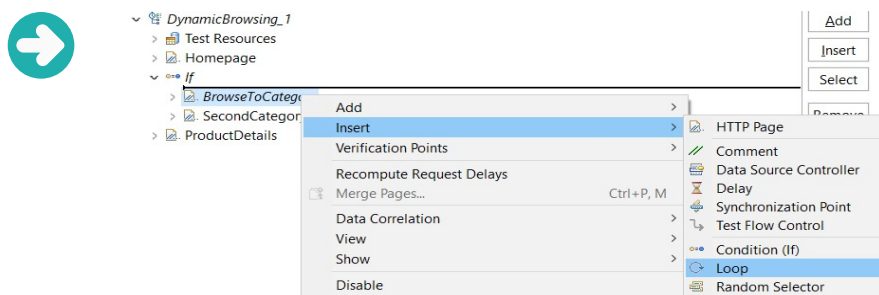
You will apply step 4 and 5 in the PseudoCode1:

- Since the catalog tree has only 2 levels, TopCategory page provides details about the top category picked and its child categories, while SecondCategory page, in addition to providing some details about the second category with the same request as TopCategory page (url?identifier), it displays the list of products under that category.

You will then update the TopCategory page in the following steps and keep SecondCategory page to be used a bit later

Change the name of the page to BrowseToCategory to make it more general.

- Insert a loop by selecting the page and right-clicking as shown below



Make the loop duration as Infinite on the right side of the view.

5 - Add a custom code to pick a random sub-category

The update that will be done here is related to step 6 in PseudoCode1 and it's creating a custom code to get a random sub-category and loop again or exit the loop if a products list page is reached.

- The response content of the first request under BrowseToCategory page has a field that shows that the next page is either a category page or a products list page as shown below.

Shown below for a category page



Down

Prev

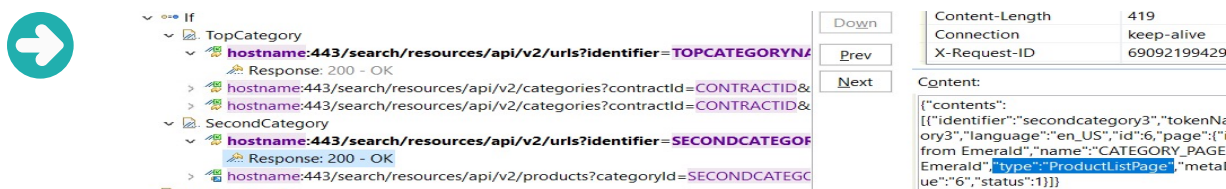
Next

| | |
|-------------------|------------|
| Transfer-Encoding | chunked |
| Connection | keep-alive |
| X-Request-ID | 690921984 |

Content:

```
{
  "contents": [
    {
      "identifier": "topcategory2",
      "tokenName": "CategoryToken",
      "tokenExternal": "pCategory2",
      "language": "en_US",
      "id": "2",
      "page": {
        "imageDescription": "Image for TopCategory2 from Emerald",
        "name": "CATEGORY_PAGE",
        "title": "TopCategory2 | Emerald",
        "type": "CategoryPage",
        "metaDescription": "",
        "metaKeywords": "",
        "tokenValue": "307445734561",
        "status": "1"
      }
    }
  ]
}
```

Shown below for a products list page



Down

Prev

Next

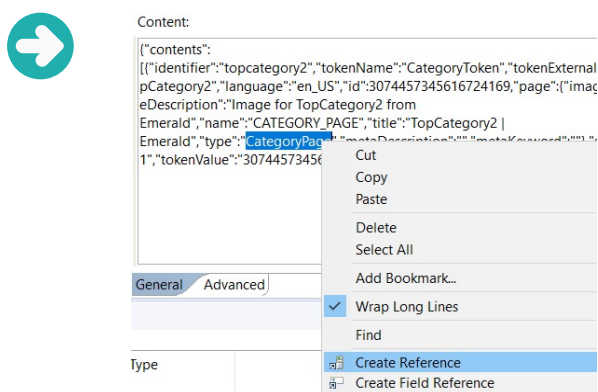
| | |
|----------------|-------------|
| Content-Length | 419 |
| Connection | keep-alive |
| X-Request-ID | 69092199429 |

Content:

```
{
  "contents": [
    {
      "identifier": "secondcategory3",
      "tokenName": "CategoryToken",
      "tokenExternal": "pCategory3",
      "language": "en_US",
      "id": "6",
      "page": {
        "imageDescription": "Image for TopCategory2 from Emerald",
        "name": "CATEGORY_PAGE",
        "title": "TopCategory2 | Emerald",
        "type": "ProductListPage",
        "metaDescription": "",
        "metaKeywords": "",
        "tokenValue": "307445734561",
        "status": "1"
      }
    }
  ]
}
```

This field will be used to decide whether to loop again or exit the loop.

- You need to create a reference in the response content on that field by selecting it and right-clicking and choosing Create Reference on the contextual menu.



Content:

```
{
  "contents": [
    {
      "identifier": "topcategory2",
      "tokenName": "CategoryToken",
      "tokenExternal": "pCategory2",
      "language": "en_US",
      "id": "2",
      "page": {
        "imageDescription": "Image for TopCategory2 from Emerald",
        "name": "CATEGORY_PAGE",
        "title": "TopCategory2 | Emerald",
        "type": "CategoryPage",
        "metaDescription": "",
        "metaKeywords": "",
        "tokenValue": "307445734561",
        "status": "1"
      }
    }
  ]
}
```

General

Advanced

Type

Cut

Copy

Paste

Delete

Select All

Add Bookmark...

Wrap Long Lines

Find

Create Reference

Create Field Reference

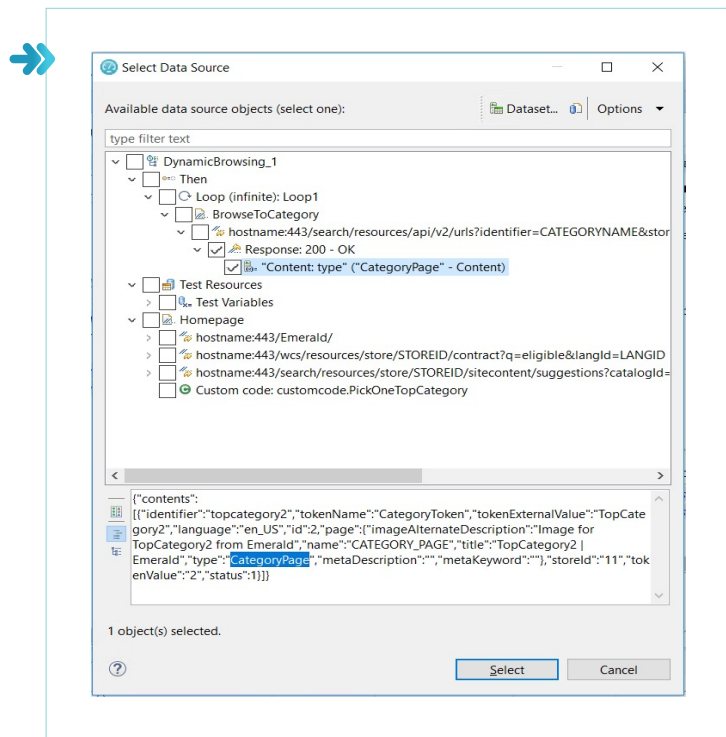
Click on Apply and Close on the dialog box.

- You need to insert an IF right after the 1st request of BrowseToCategory page to test on the field you just create a reference for the same way you did before

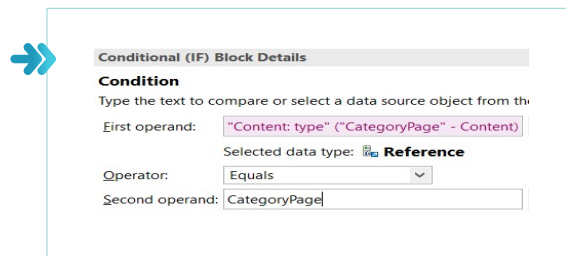
Select the 2 last requests, right-click and choose Insert, then Condition (If), then Yes on the dialog box.

Same way here you need to define the 1st and the 2nd operand of the IF test.

1st operand



2nd operand



This means that if the field is equal to CatalogPage, then continue and run the 2 last requests.

- You need to add an ELSE to the previous IF to add a flow in case the field is not equal CatalogPage (ie equal to ProductListPage).

Select the If, right-click, Add, then choose Condition (If) - ELSE Block

Select the Else, right-click, Add, then choose Test Flow Control

Select the Test Flow Control and select Exit Loop for Innermost

The result will look like this:



By doing this you're defining the following flow:

If the field is equal to CatalogPage

Then Get sub-categories by running the 2 last request

Else Exit the loop as we're on the products list page

- You can now add the custom code to fetch all sub-categories and pick one of them randomly.

The custom code should be under the section Then and it should use the response content of the 2nd request under Then's section like the following:



Same way here too, you need to create a field reference for the highlighted response above and use it as an argument to the custom code.

The highlighted response content looks like the following:

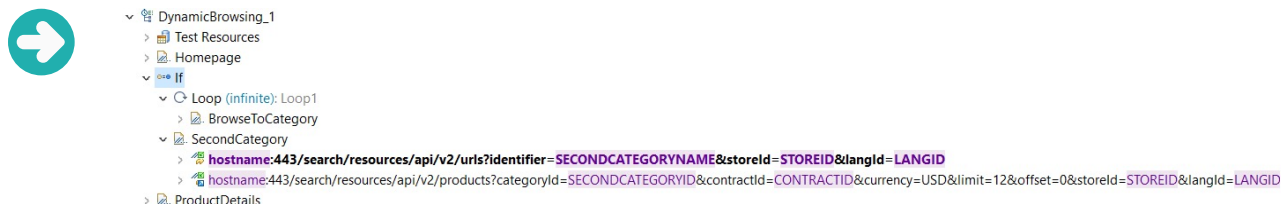
```
{
  "contents": [
    {
      "seo": {
        "href": "/secondcategory3"
      },
      "name": "SecondCategory3",
      "identifier": "SecondCategory3",
      "storeId": "11",
      "sequence": "1.0",
      "id": "6",
      "links": {
        "parent": {
          "href": "/search/resources/api/v2/categories?storeId=11&id=2"
        },
        "self": {
          "href": "/search/resources/api/v2/categories?storeId=11&id=6"
        }
      }
    },
    ....
  ]
}
```

So, the custom code content should be almost the same as the previous one. You should look for names and ids only in the response.

6 - Add a products list page

You will implement step 9 in the PseudoCode1 which is adding a products list page after exiting from the loop.

The current module is like the following:



Once the loop is exit, the category leaf should have been reached and the products should be displayed.

SecondCategory page contains a request to display the products and contain one more request which is not needed as it relates to the categories. This page will be updated to be the products list page and it should be under the If since we cannot get to a products list if a top category is not found.



7 - Add a custom code to pick one product

You will implement step 10 in PseudoCode1 to create a custom code to pick one random product.

The response content of the request under ProductsList page looks like the following:

```

{
  "contents": [
    {
      "hasSingleSKU": false,
      "buyable": "true",
      "parentCatalogGroupID": "/2/6",
      "name": "Product11",
      "shortDescription": "Short description of product11",
      "seo": {
        "href": "/product11"
      },
      "partNumber": "Product11",
      "storeID": "11",
      "price": [
        {
          "usage": "Display",
          "description": "L",
          "currency": "USD",
          "value": "90.17"
        },
        {
          "usage": "Offer",
          "contractId": "-11005",
          "description": "I",
          "currency": "USD",
          "value": "88.17"
        }
      ],
      "type": "product",
      "id": "11"
    },
    .....
  ]
}

```

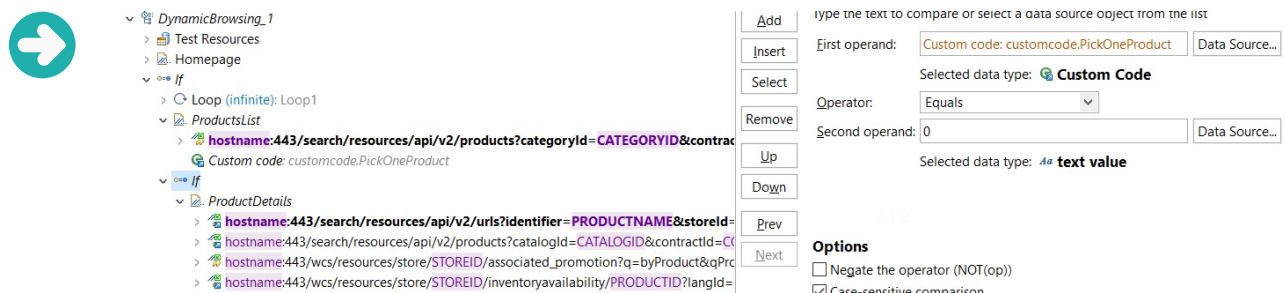
So, the custom code will be almost the same as the custom code for categories you created earlier. Notice that in ProductDetails page, only product name and id are used, so in the custom code you need to search for the name and id only and set product_name and product_id at the end of the custom code.

You also need to create a field reference for the response highlighted above.

8 - Add a product details page

This is implementing step 11 in the PseudoCode1 to add product details page. You will need to move ProductDetails page up to put it as part of the IF.

To avoid any issue, you should add a test on the custom code output to be sure that you found at least one product and set the 1st operand of the IF to the custom code and the 2nd operand to 0. This ensures that you don't display a product details unless the products list is not empty and you picked one product:



The screenshot shows the test flow editor with the following structure:

- DynamicBrowsing_1
 - Test Resources
 - Homepage
 - Loop (infinite): Loop1
 - ProductsList
 - hostname:443/search/resources/api/v2/products?categoryId=CATEGORYID&contractId=CONTRACTID
 - Custom code: customcode.PickOneProduct
 - If
 - ProductDetails
 - hostname:443/search/resources/api/v2/urls?identifier=PRODUCTNAME&storeId=STOREID
 - hostname:443/search/resources/api/v2/products?catalogId=CATALOGID&contractId=CONTRACTID
 - hostname:443/wcs/resources/store/STOREID/associated_promotion?q=byProduct&qProductId=PRODUCTID
 - hostname:443/wcs/resources/store/STOREID/inventoryavailability/PRODUCTID?langId=LANGID

The 'If' block configuration is shown on the right:

- Type the text to compare or select a data source object from the list
- First operand: Custom code: customcode.PickOneProduct (Data Source...)
- Selected data type: Custom Code
- Operator: Equals
- Second operand: 0 (Data Source...)
- Selected data type: text value
- Options:
 - ☐ Negate the operator (NOT(op))
 - ☒ Case-sensitive comparison

9 - Add an ELSE to the first IF

This is step 12 and 13 in PseudoCode1.

If the 1st custom code PickOneTopCategory doesn't found a top category, the script should stop. To implement this behavior, right-click the IF and choose Add Condition If - ELSE Block.

Right-click the Else and Add Test Flow Control.

10 - Final script

The whole flow of the module will look like the following:



The final test flow script is as follows:

- DynamicBrowsing_1
 - Test Resources
 - Homepage
 - hostname:443/Emerald/
 - hostname:443/wcs/resources/store/STOREID/contract?q=eligible&langId=LANGID
 - Set contract_id = ["contract_id" ("11005" - Content)]
 - hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&depthAndLimit=11,11&storeId=STOREID&langId=LANGID
 - Custom code: customcode.PickOneTopCategory
 - If
 - Then
 - Loop (infinite): Loop1
 - BrowseToCategory
 - hostname:443/search/resources/api/v2/urls?identifier=CATEGORYNAME&storeId=STOREID&langId=LANGID
 - If
 - Then
 - hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&identifier=CATEGORYNAME&storeId=STOREID&langId=LANGID
 - hostname:443/search/resources/api/v2/categories?contractId=CONTRACTID&parentCategoryId=CATEGORYID&storeId=STOREID&langId=LANGID
 - Custom code: customcode.GetOneNextCategory
 - Else
 - Test Flow Control
 - ProductsList
 - hostname:443/search/resources/api/v2/products?categoryId=CATEGORYID&contractId=CONTRACTID¤cy=USD&limit=12&offset=0&storeId=STOREID&langId=LANGID
 - Custom code: customcode.PickOneProduct
 - If
 - ProductDetails
 - hostname:443/search/resources/api/v2/urls?identifier=PRODUCTNAME&storeId=STOREID&langId=LANGID
 - hostname:443/search/resources/api/v2/products?catalogId=CATALOGID&contractId=CONTRACTID&partNumber=PRODUCTNAME&storeId=STOREID&langId=LANGID
 - hostname:443/wcs/resources/store/STOREID/associated_promotion?q=byProduct&qProductId=PRODUCTID&langId=LANGID
 - hostname:443/wcs/resources/store/STOREID/inventoryavailability/PRODUCTID?langId=LANGID
 - Else
 - Test Flow Control

11 - Apply the 80/20 rule

Before going into the details of implementing dynamic browsing, let's see how to implement the 80/20 rule in this case (See its definition at the beginning of the section).

One way to implement the 80/20 rule is using the following formula

$$f(x) = \text{floor}(N * x^7)$$

where:

x is a random generated number with uniform distribution $0 \leq x < 1$

N is the number of items to be browsed.

This formula will be used to get one random top category like the following:

Inside a custom code, if the top categories will be in an array named `topCategoryList` then the formula to pick a random category respecting the 80/20 rule will look like this:

```
i = (int)(Math.floor(topCategoryList.size() * Math.pow(Math.random(), 7)));
```

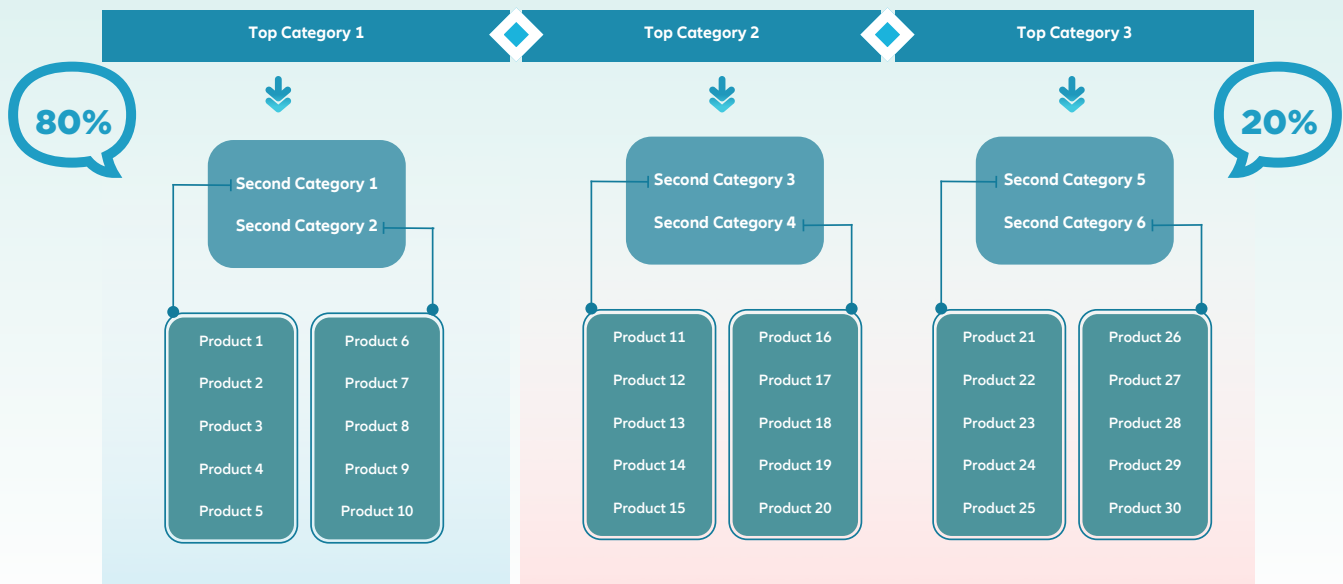
Where:

`topCategoryList.size() = N`

`Math.random() = x`

i is the index in the array of top category list `topCategoryList`

Using the above formula, `TopCategory1` will be visited 80% of the time which makes the user pick 80% of the time products `Product1` to `Product10` as shown in the screenshot below:



The same formula (to apply the 80/20 rule) can be applied for sub-categories and for products if the number of sub-categories and/or products is high.

To apply this rule, you need to change the following line in the custom code `GetOneTopCategory`:

```
int i = (int) Math.floor( Math.random() * a_ids.length );
```

By this line

```
int i = (int) Math.floor( a_ids.length * Math.pow( Math.random(), 7 ) );
```

For the sake of testing, you can add a new global variable `enable_80_20_rule`, that can take 'yes' or 'no', defined in `InitVariables` module described in the previous section. This variable should be set as a parameter to the custom code and used to test either to apply the rule or not. It can make it easy to test with and without the 80/20 rule. The code becomes:

```
int i;  
  
if ( enable_80_20_rule == "yes" ) {  
    i = (int)( Math.floor( a_ids.length * Math.pow( Math.random(), 7 ) ));  
}  
  
else if ( apply_80_20_rule == "no" ) {  
    i = (int) Math.floor( Math.random() * a_ids.length );  
}  
  
else {  
    return "-1";  
}
```

Remarks

Only browsing is being implemented. Checkout was not considered. It can be implemented following the same techniques above.

The substitution of requests headers was not mentioned.

PROBLEM DETERMINATION FOR ONETEST PERFORMANCE SCRIPTS

This section presents some guidelines and strategies to help in problem determination when hitting errors in running OneTest Performance scripts.

Before running a test script, it is advisable to first manually check that the scenarios are functionally working. If running a manual test of the scenario passes but running the OneTest Performance script fails, then there may be a mismatch of requests or bad data being used in the script.

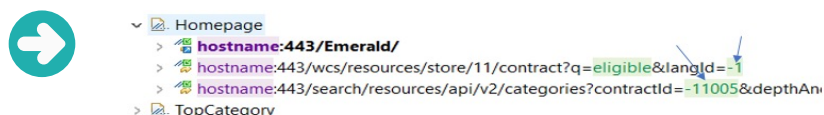
Re-recording the scenario and comparing it to the original script is required to find any differences. This is recommended when older test scripts are used to test a server with updated store pages or underlying application code.

1 - Examine references and substitutions in the test script

In test scripts, errors can occur when request parameters are dynamic but are not substituted with the right values. References are these values and they are extracted from response content using a regular expression. They replace hardcoded parameters in requests.

OneTest Performance provides automatic data correlation which means it finds references and makes substitutions in the test without the user having to do anything. However manual intervention is recommended to control some substitutions. Automatic data correlations often use imprecise regular expressions which can fail to find references for substitutions. If some dynamic request parameters are not substituted and left with hardcoded values, then this may cause an invalid request.

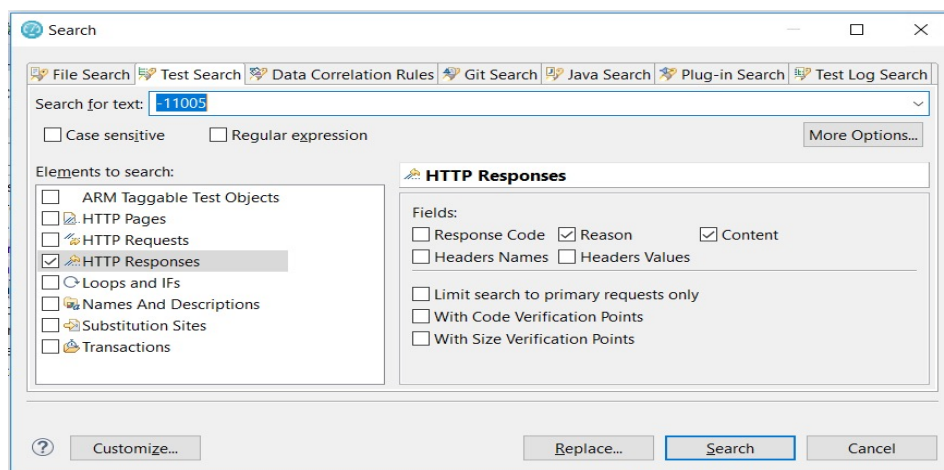
An example of that is the requests under Homepage right at the beginning of Dynamic browse section. It's also copied again here



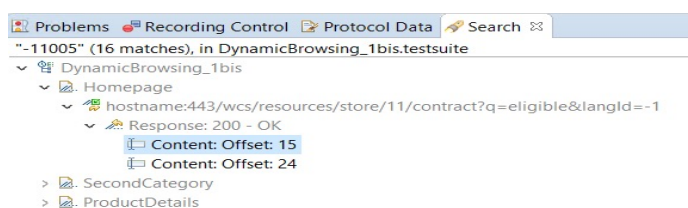
The contract id wasn't correlated correctly. It was highlighted in light green. It would've been highlighted in light red otherwise.

One way to fix data correlation is to use the test search feature in OneTest Performance to find where the value is coming from:

- Select the value that you want to correlate in the URL field and use search tool (CTRL+H or Menu Search > Search). The value will be automatically put in the Search for text field.
- Since the contract value should be coming from a response content of another requests called before the current one, ensure HTTP Responses checkbox is checked so the search will be in response contents.



- The result will be shown in the Search console and the first occurrence will be highlighted.



- Double-click on the 1st occurrence to see where this response content is located vs the request that you want to correlate its value. If the response content is from a request that was called before the contract's request, then it can be used to get the value from. This is the case actually as shown below:

The screenshot shows the HCL OneTest Performance interface. On the left, a tree view displays the test suite 'DynamicBrowsing_1bis' with steps like 'Homepage', 'ProductDetails', and 'SecondCategory'. A specific response is highlighted: 'hostname:443/search/resources/api/v2/categories?contractId=-11005&depthAnd'. The right pane shows the 'Response Headers' and 'Content' sections. The 'Content' section displays a JSON snippet: `{"contracts":[{"contractId": "-11005"}]}`. Below the main view, a search bar shows '11005' (16 matches) in 'DynamicBrowsing_1bis.testsuite'. The bottom pane shows the 'Content: Offset: 15[visited]' and 'Content: Offset: 24'.

- You can add a variable `contract_id` and set it from the value in this response content, which is what was done earlier when building the dynamic browse script.

2 - Set test behaviors when errors occur

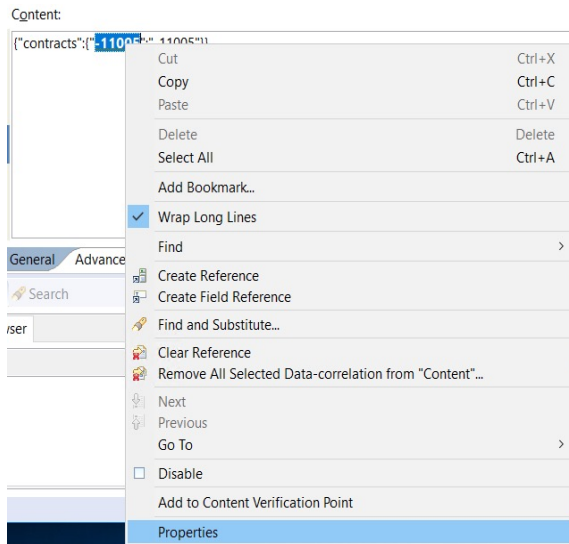
Scripts can be configured to act when an error occurs. The error can be a timeout, failing a content verification point or not finding a reference such as `contract_id`. By default, a message is logged and the rest of the test continues. However, in some cases the test should stop. OneTest Performance can set error handling at the schedule level and at the script level. This is configured by going into the Advanced tab for a schedule or test's properties:

The screenshot shows the 'Error Handling' configuration in HCL OneTest Performance. The left pane shows the test suite 'DynamicBrowsing_1' with steps like 'Homepage', 'ProductList', and 'ProductDetails'. The right pane shows the 'Error Handling' configuration. A table lists various error conditions and their actions. The 'Page Title Verification Point Failure' condition is selected, and its action is 'Continue (default) unless defined by schedule'. The 'Affects...' column shows 'Yes (de...)' and the 'Message' column shows 'Page title verification f...'. The 'Hide unselected conditions' checkbox is checked. The bottom pane shows the 'General' and 'Advanced' tabs.

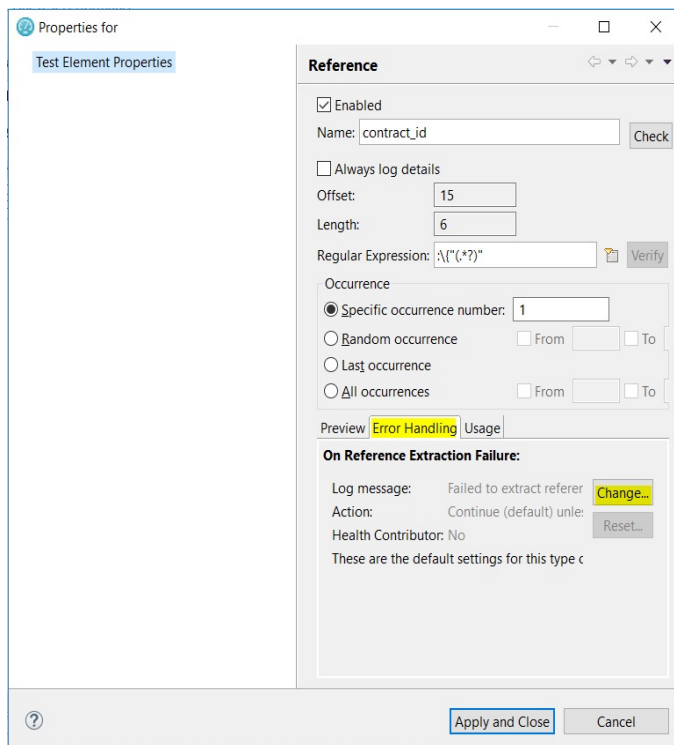
| Condition | Action | Affects ... | Message |
|---|--|-------------|------------------------------|
| <input checked="" type="checkbox"/> Page Title Verification Point Failure | Continue (default) unless defined by schedule | Yes (de...) | Page title verification f... |
| <input type="checkbox"/> Response Code Verification Failure... | Continue (default) unless defined by schedule | Yes (de...) | Response code verifica... |
| <input type="checkbox"/> Response Size Verification Failure ... | Continue (default) unless defined by schedule | Yes (de...) | Response size verificat... |
| <input type="checkbox"/> Content Verification Point Failure | Continue (default) unless defined by schedule | Yes (de...) | Content Verification Po... |
| <input type="checkbox"/> Connection Failure | Continue (default) unless defined by schedule | Yes (de...) | Connection failed (def... |
| <input type="checkbox"/> Authentication Failure | Continue (default) unless defined by schedule | Yes (de...) | Authentication failed (...) |
| <input type="checkbox"/> End of Dataset reached | Exit User (default) unless defined by schedule | Yes (de...) | End of dataset reached... |
| <input type="checkbox"/> Reference Extraction Failure | Continue (default) unless defined by schedule | Yes (de...) | Failed to extract refere... |
| <input type="checkbox"/> Substitution Failure | Continue (default) unless defined by schedule | Yes (de...) | Substitution failed (def... |
| <input type="checkbox"/> Server Timeout | Continue (default) unless defined by schedule | Yes (de...) | Timeout (default) unles... |
| <input type="checkbox"/> Custom Verification Point Failure | Continue (default) unless defined by schedule | Yes (de...) | Custom Verification Poi... |
| <input type="checkbox"/> Custom Code Alert | Continue (default) unless defined by schedule | No (de...) | Custom Code reported... |
| <input type="checkbox"/> Custom Code Exception | Exit User (default) unless defined by schedule | Yes (de...) | Custom Code reported... |

The response content of the request to get the contract must contain a value for contractId. If this reference extraction fails, then the script should stop because subsequent requests require the contractId as a parameter and thus will fail:

- You can change the default behavior if contract id reference extraction fails by using the contractId reference properties:



- And click Change button:



- Depending on how the test is structured and if it is within a loop or transaction, choose an appropriate exit option. In this example, continue to next iteration of outermost loop.



Test Editor

Reference Extraction Failure

Select options to override for this type of error.

☒ Override action upon error

☐ Continue

☐ Exit Transaction Innermost

☐ Exit Loop Innermost

☒ Continue to next iteration of Loop Outermost

☐ Exit Test

☐ Exit User

☐ Terminate Run

☒ Override contribution to health status

Affects page health: Yes

☒ Override log message upon error

Failed to extract reference

OK Cancel

By controlling and managing the first occurrence of errors, subsequent errors are avoided and do not complicate problem determination.

CONCLUSION

This paper describes how to use OneTest Performance to write test scripts for HCL Commerce.

It presents a step-by-step guidance on how to implement browsing an e-Commerce storefront using two techniques, static and dynamic. It shows some techniques used in problem determination as well.

Some best practices mentioned are:

- ✓ Comment while recording
- ✓ Designing modular tests
- ✓ Making use of verification points and error handling
- ✓ Modeling behavior using 80/20 rule in browsing to increase cache hits
- ✓ Building schedules using loops, random selectors and transaction containers

About HCL OneTest Performance

HCL OneTest Performance allows for quickly executing performance tests that analyze the impact of load on applications.

Testers can validate the scalability of web and server-based applications, identify the presence and cause of system performance bottlenecks, while tracking performance changes between versions.

HCL OneTest Performance enables performance testers to easily create realistic workloads that model daily user activities, while eliminating last-minute performance problems. Scalable through bare metal and Docker agents, provisioning a performance test platform has never been easier.

With HCL's DevOps integration, bringing performance quality metrics to delivery pipelines is simple and effective.

About HCL Commerce

HCL Commerce is designed to adapt to your business strategy, not the other way around, helping your commerce experiences lead the pack instead of having to follow the herd with a templated approach. With HCL Commerce you can support the individuality of your brands, as well as pivot to address current and future business needs quickly and elegantly so you can grow with confidence and deliver a consistently inspiring experience.

About HCL Software

HCL Software is a division of HCL Technologies (HCL) that operates its primary software business. We develop, market, sell, and support more than 20 product families in the areas of Customer Experience, Digital Solutions, Secure DevOps, Security and Automation.

For more information, visit hcltechsw.com.

Copyright ©2021. All rights reserved. No materials from this case study can be duplicated, copied, republished or reused without the written permission from HCL Software. The information and insights contained in this case study reflect research and observations made by HCL Software.