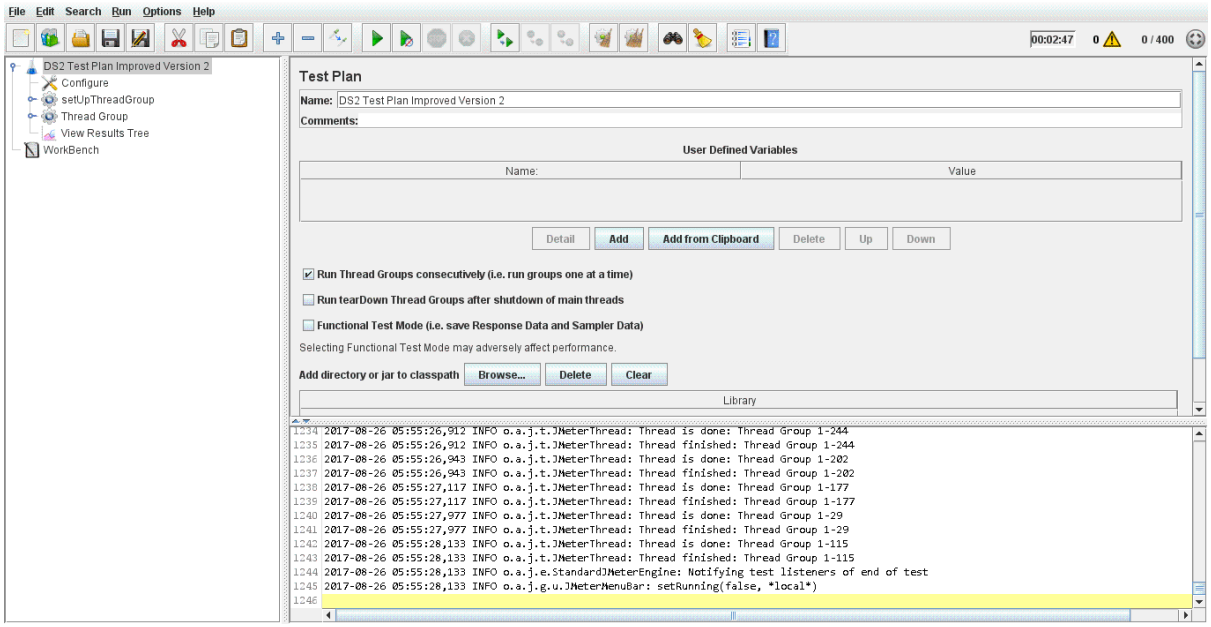
# DS2 JMeter Test Plan Design

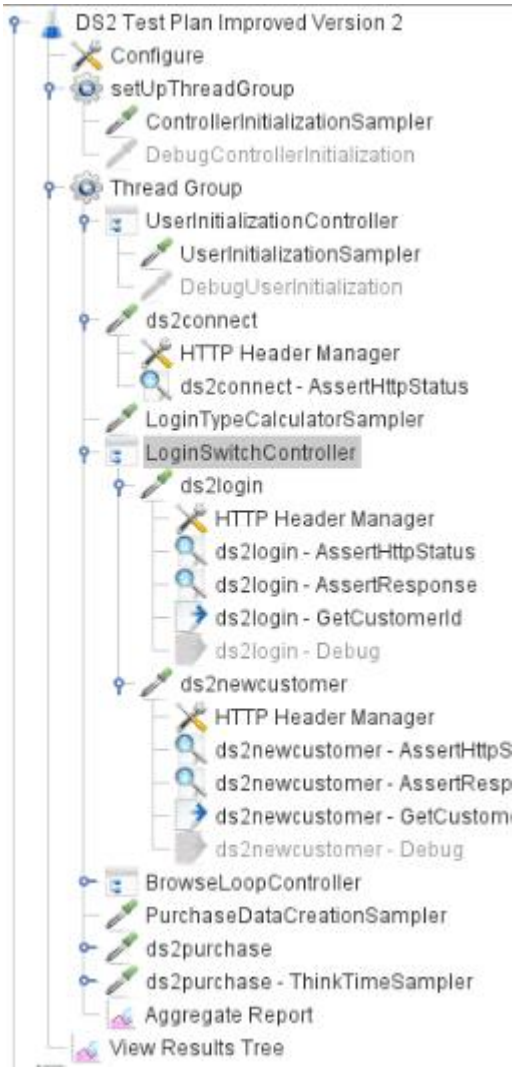
# Contents

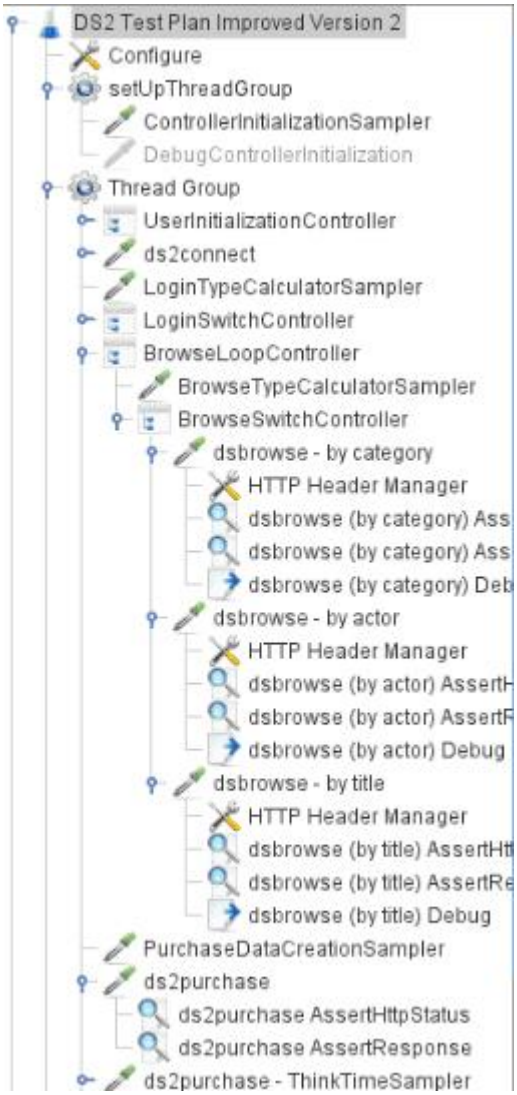
# 1 Test Plan




The top level is the Test Plan. The name of the Test Plan is DS2 Test Plan Improved Version 2. The Test Plan has following parts.

- Configure
- setUpThreadGroup
- ThreadGroup
- View Results Tree

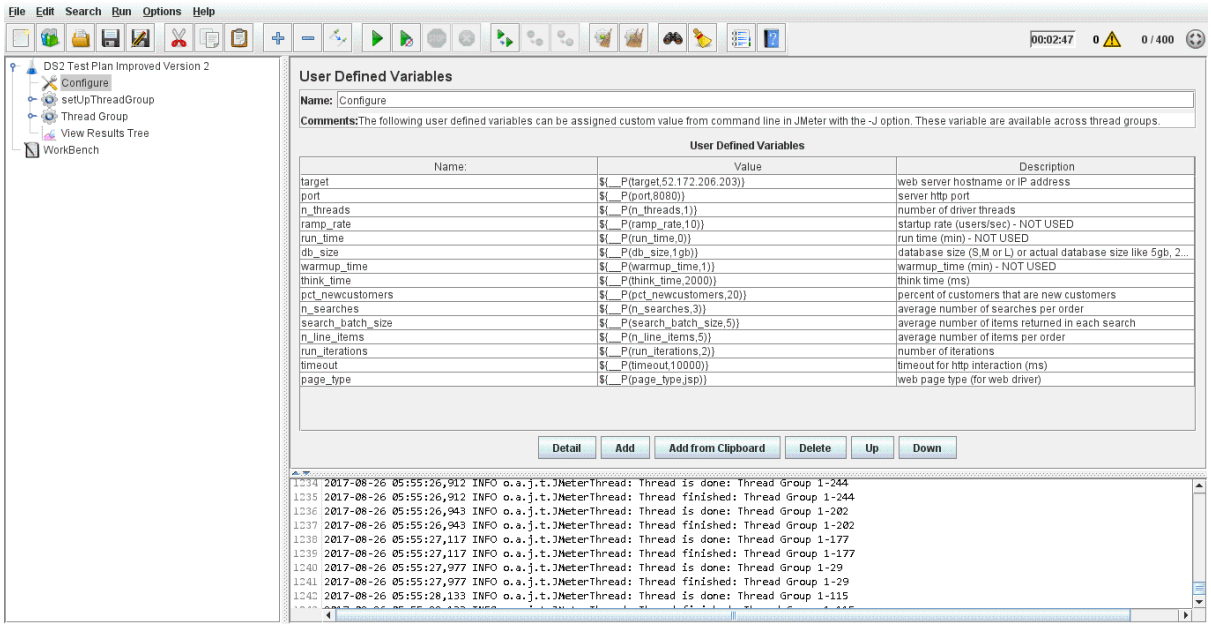



To ensure that setUpThreadGroup runs before ThreadGroup, the Run Thread Groups consecutively property of the Test Plan is enabled.

## 1.1 Configure

Configure has JMeter user defined variables. The user defined variables are provided with default values. The values can also be set from the command line using the -J option.

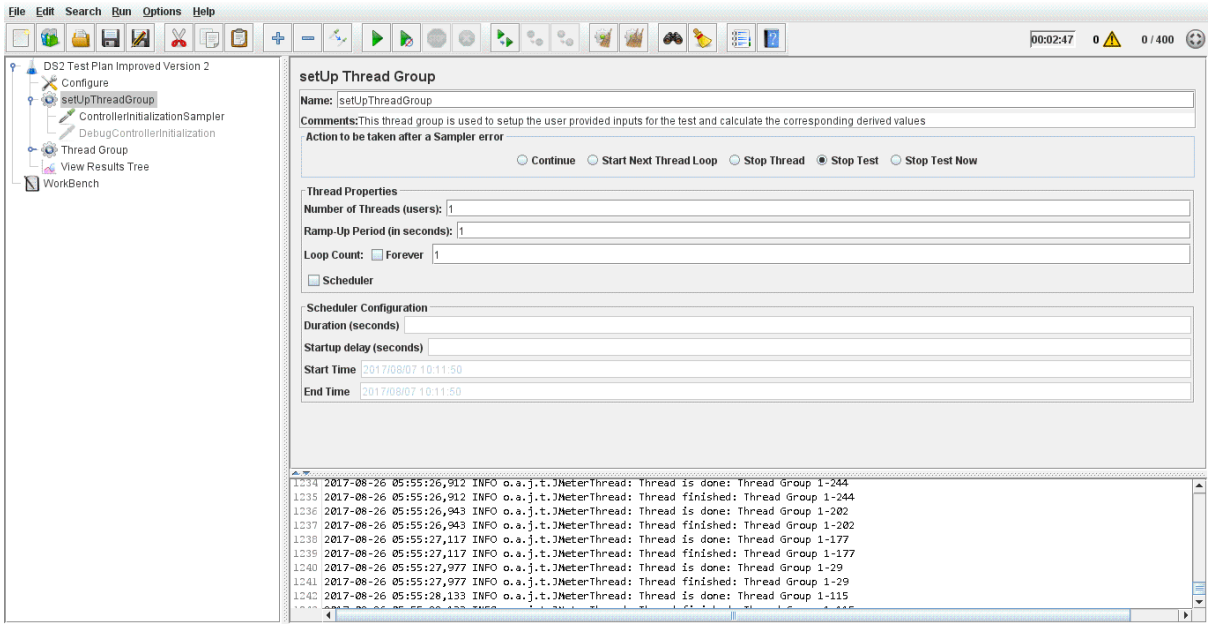



Following table provides the name, default value and description.

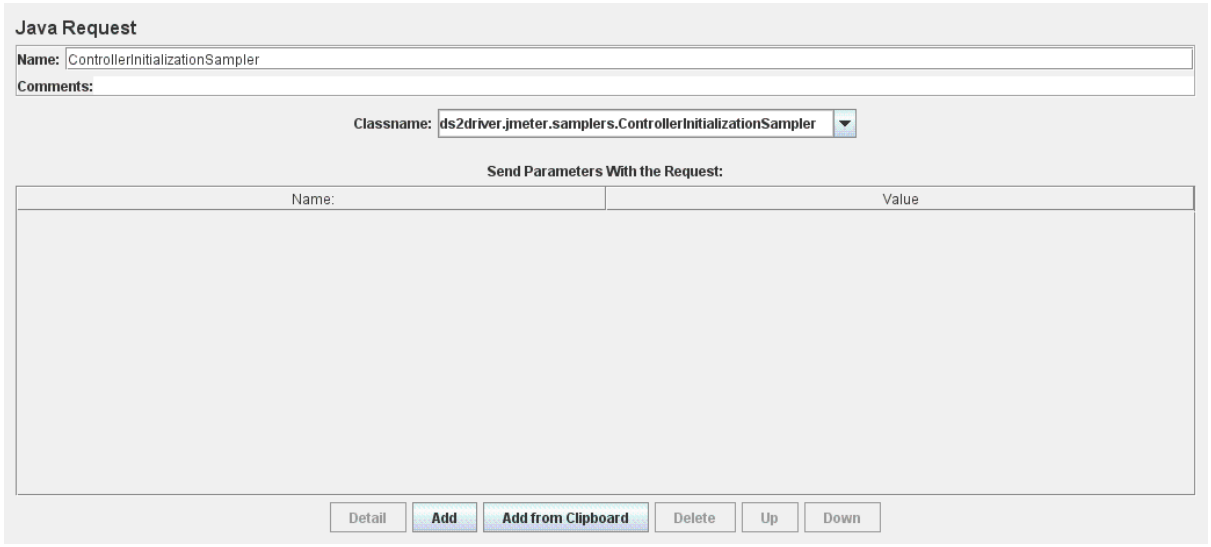
| Variable | Default Value | Description |
|---|---|---|
| target | - | |
| port | 8080 | |
| n_threads | 1 | |
| *ramp_rate* | 10 | |
| *run_time* | 0 | |
| db_size | 1gb | |
| *warmup_time* | 1 | |
| think_time | 2000 | |
| pct_newcustomers | 20 | |
| n_searches | 3 | |
| search_batch_size | 5 | |
| n_line_items | 5 | |
| *run_iterations* | 2 | |
| timeout | 10000 | |
| page_type | jsp | |

## 1.2 setUpThreadGroup

setUpThreadGroup initializes the objects required in the run using the input value specified at the start of run. It also if any of the required parameters are missing. The thread executes only once i.e. the Number of Threads (Users) and Loop Count is 1.




The logic is coded in *ControllerInitializationSampler* which is a custom Java sampler class.




The following values are checked.

- target
- port
- n_threads
- search_batch_size
- n_searches
- n_line_items
- pct_newcustomers
- db_size

The following values are internally derived using db_size.

- i_db_custom_size
- str_is_mb_gb
- ratio
- mult_cust_rows
- mult_ord_rows

- mult_prod_rows

The following values are calculated and stored for future use.

- customer_rows
- order_rows
- product_rows
- max_customer
- customer_rows

A singleton class *GlobalParametersSingleton* is initialized with following values.

- prod_array_size
- max_product
- max_customer
- pct_newcustomers
- n_searches
- search_batch_size
- n_line_items

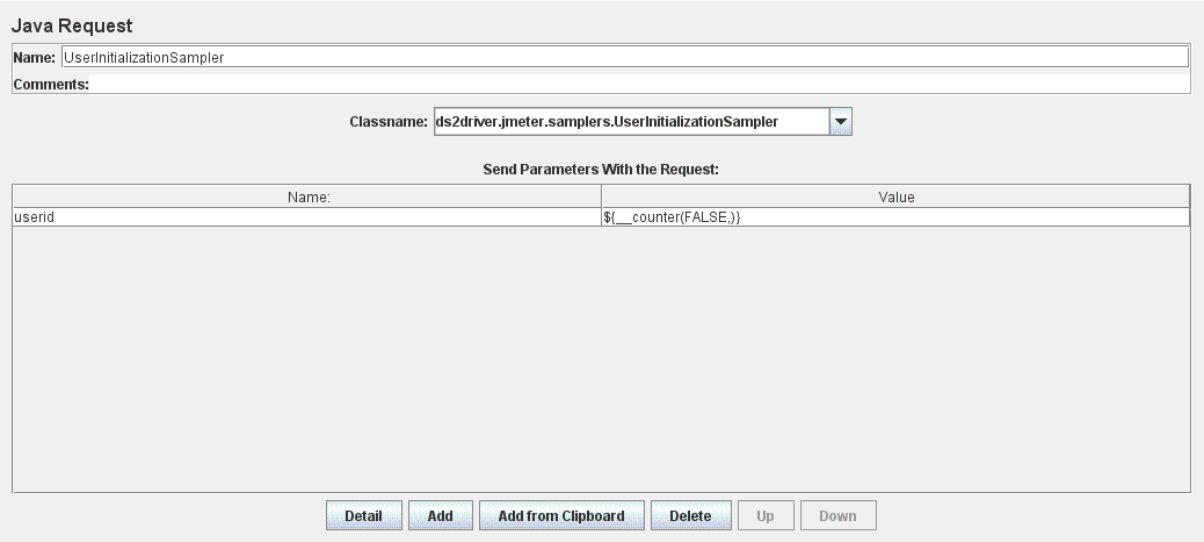
This class is designed to be thread safe. However, the class is called only by a single thread.

## 1.3 ThreadGroup

ThreadGroup contains the entire test sequence.

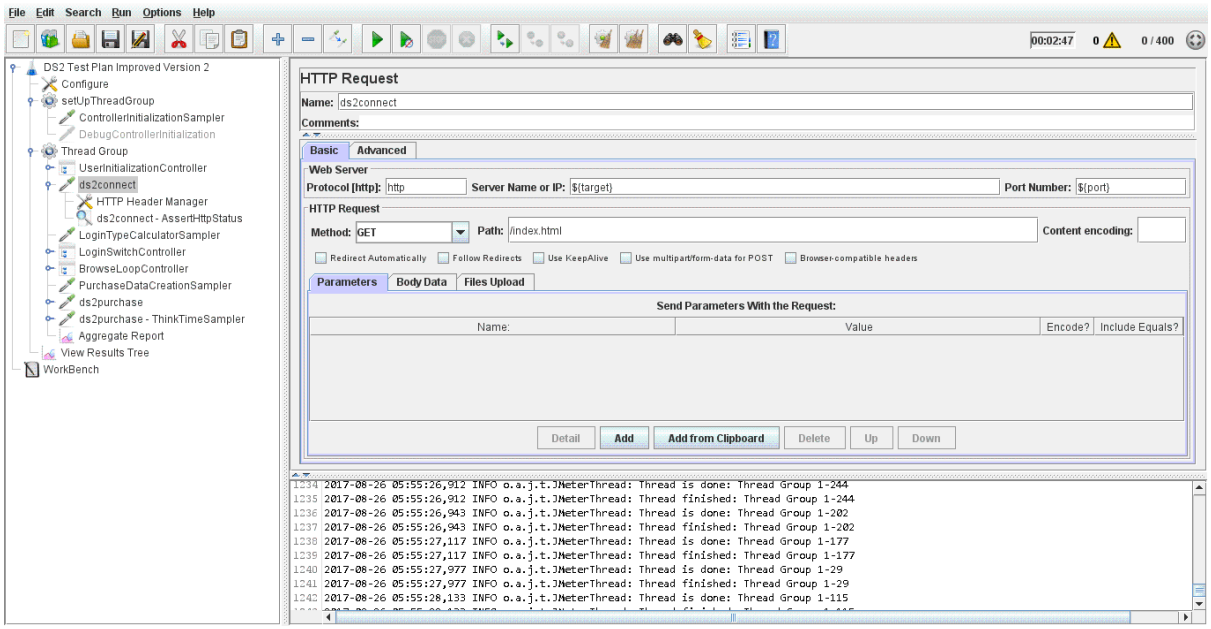
### 1.3.1 Step 1: UserInitializationController

The first step is a Once Only Controller named *UserInitializationController*. The controller has one custom Java sampler *UserInitializationSampler*. This sample executes only once per every thread i.e. first iteration of the loop. The sampler creates a User object (one object is created per thread) and initializes it with a running sequence number. The sequence number is injected using an out of box JMeter function counter. FALSE indicates, that the counter is common across threads.

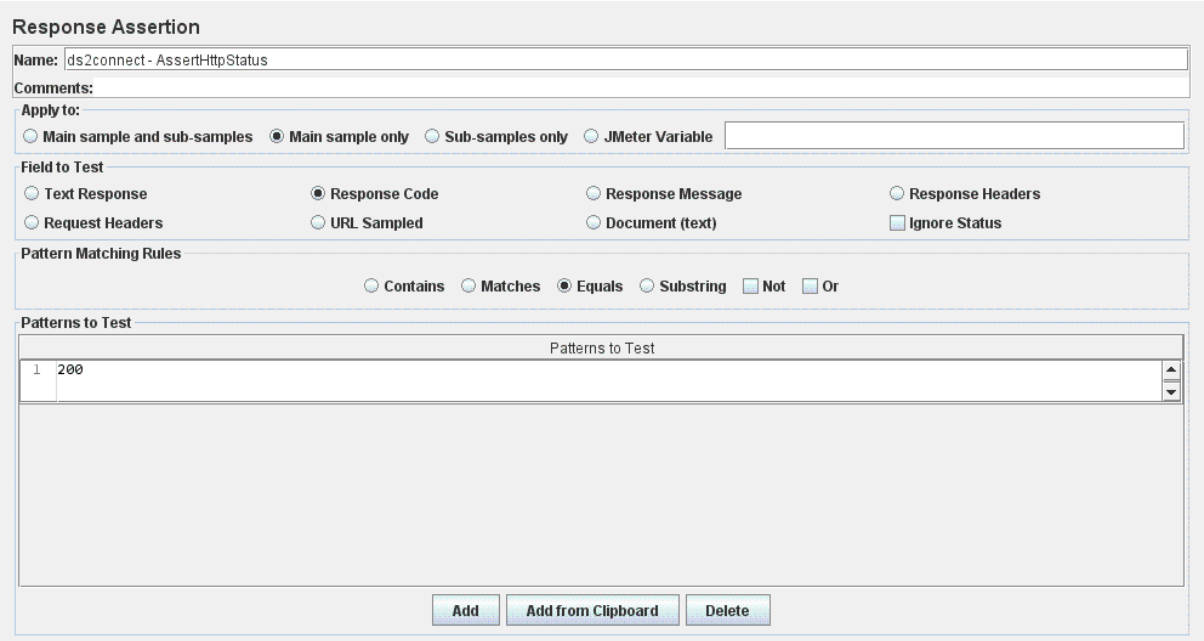


### 1.3.2 Step 2: ds2connect

The second step is an out of box JMeter HTTP request sampler named *ds2connect*. The sampler executes /index.html. The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent.

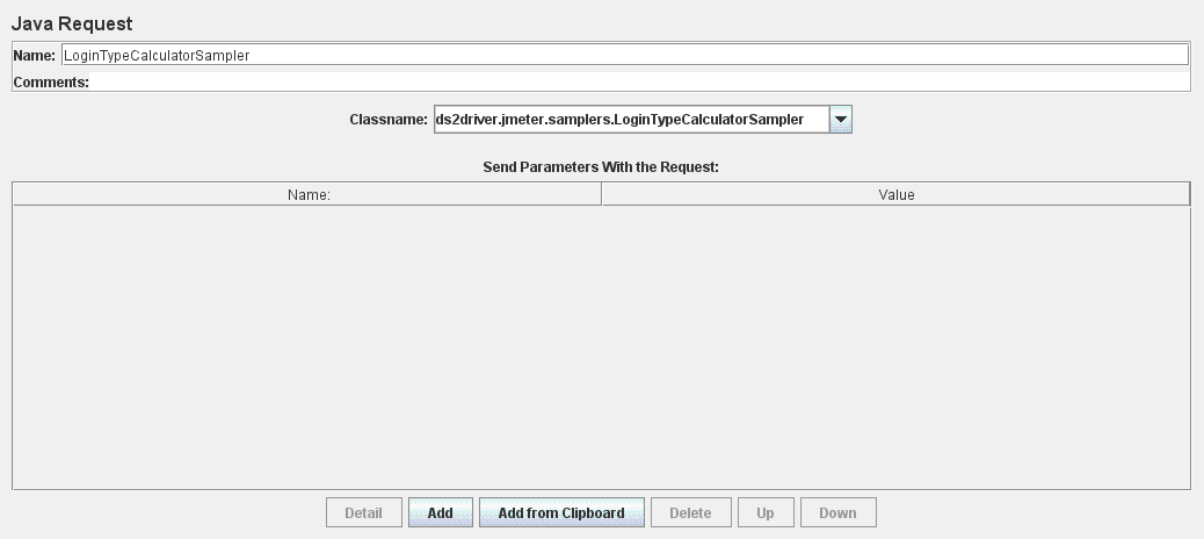



The http response code is checked to see if it is 200.



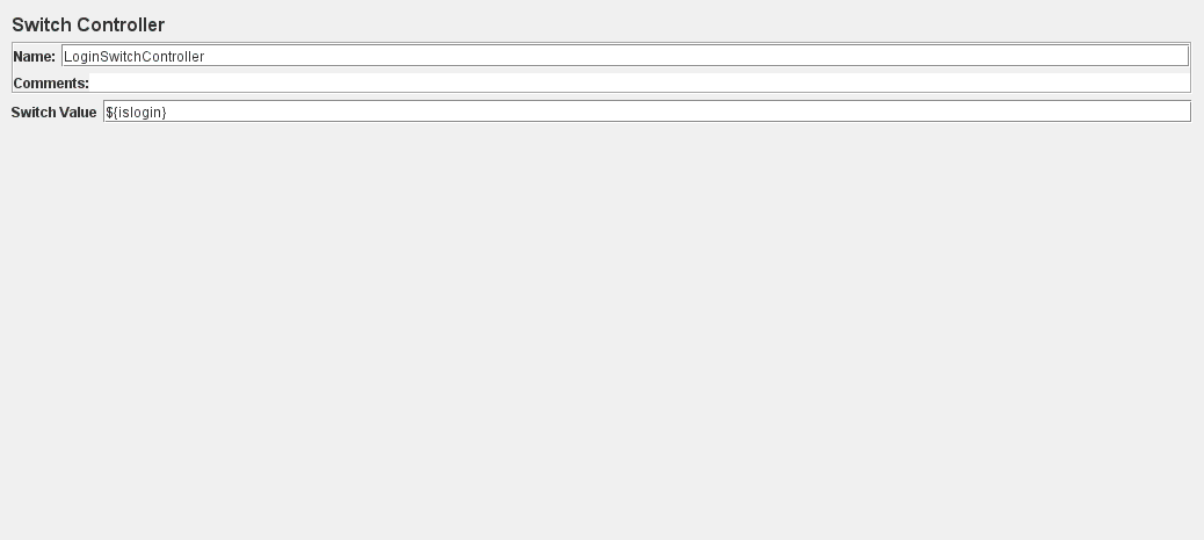

### 1.3.3 Step 3: LoginTypeCalculatorSampler

The third step is a custom Java request sampler named *LoginTypeCalculatorSampler*. The sampler calculates the request type using randomization logic and the pct_newcustomers value. It can be an existing DS2 customer login or registration of a new DS2 customer. The data for both request types is created using internal functions CreateLoginData and CreateLoginData respectively. The number of iterations to be used in Step 5 is also calculated and stored in variable *n_browse*. The data is stored in the form of JMeter variables. JMeter variables are thread specific.




### 1.3.4 Step 4: LoginSwitchController

In the fourth step, the *LoginSwitchController* either invokes *ds2login* or *ds2newcustomer* based on the value of request type calculated in the third step. The variable containing the request type is *islogin*.

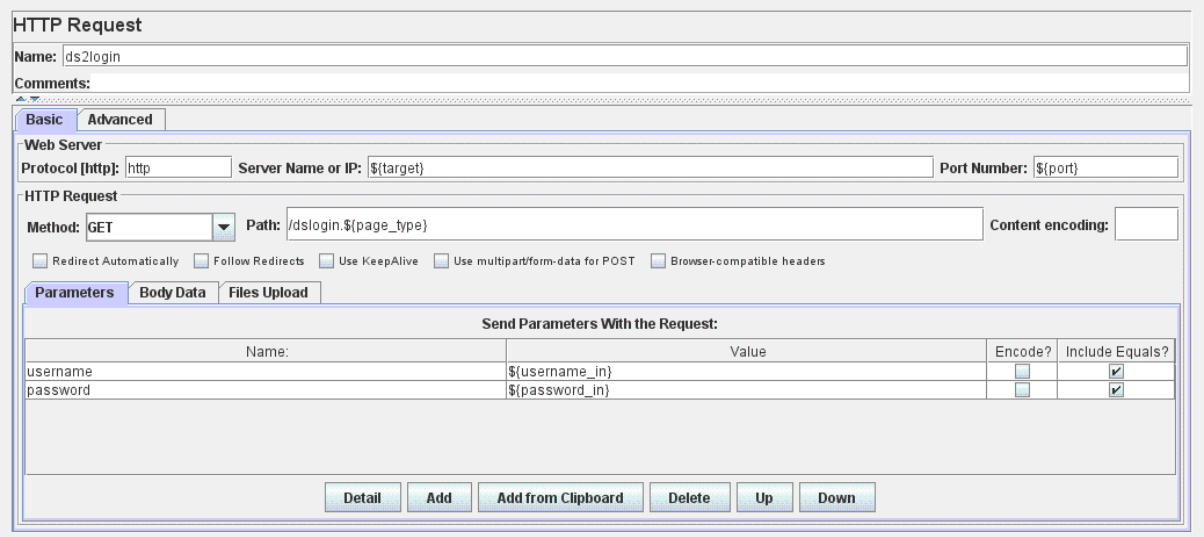



#### *1.3.4.1 Step 4.A: ds2login*

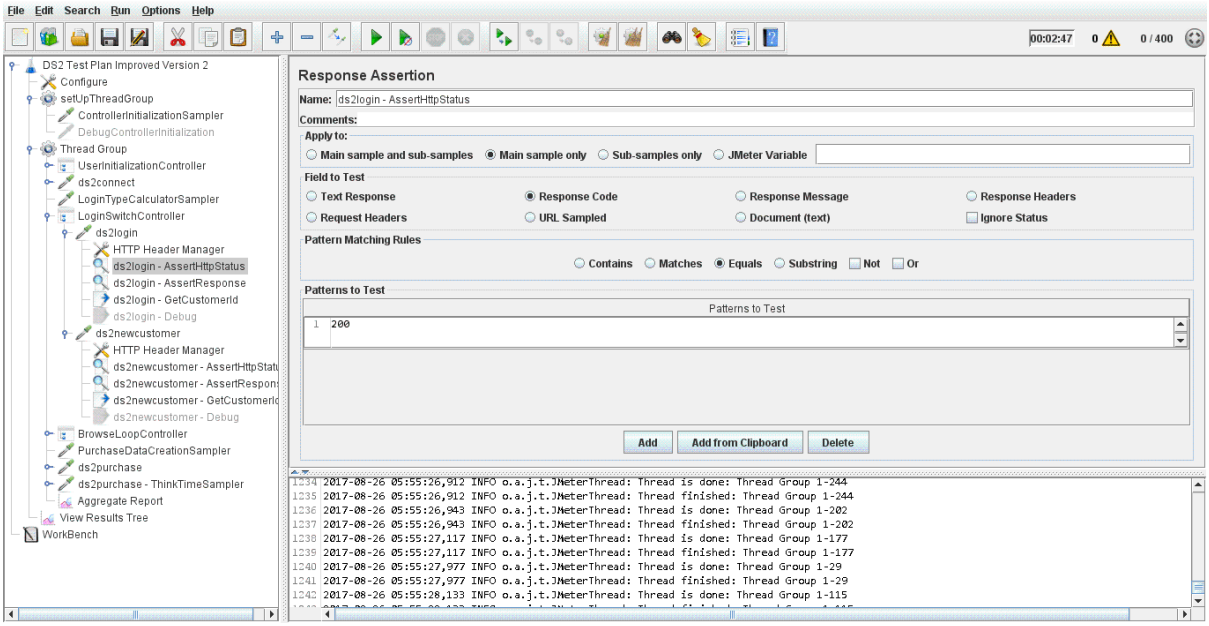
The step is an out of box JMeter HTTP request sampler named *ds2login*. The sampler executes /dslogin.jsp.

The query parameters are sent the JMeter out of box option (Send Parameters With the Request). The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent. The query parameters are as follows.

- username
- password

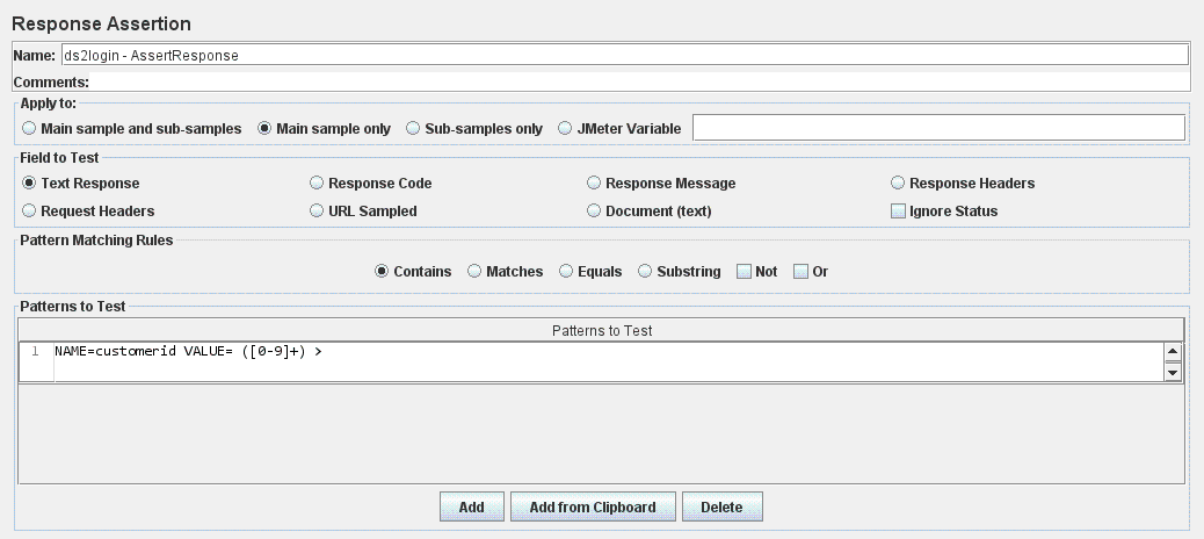



First the http response code is checked to see if it is 200.

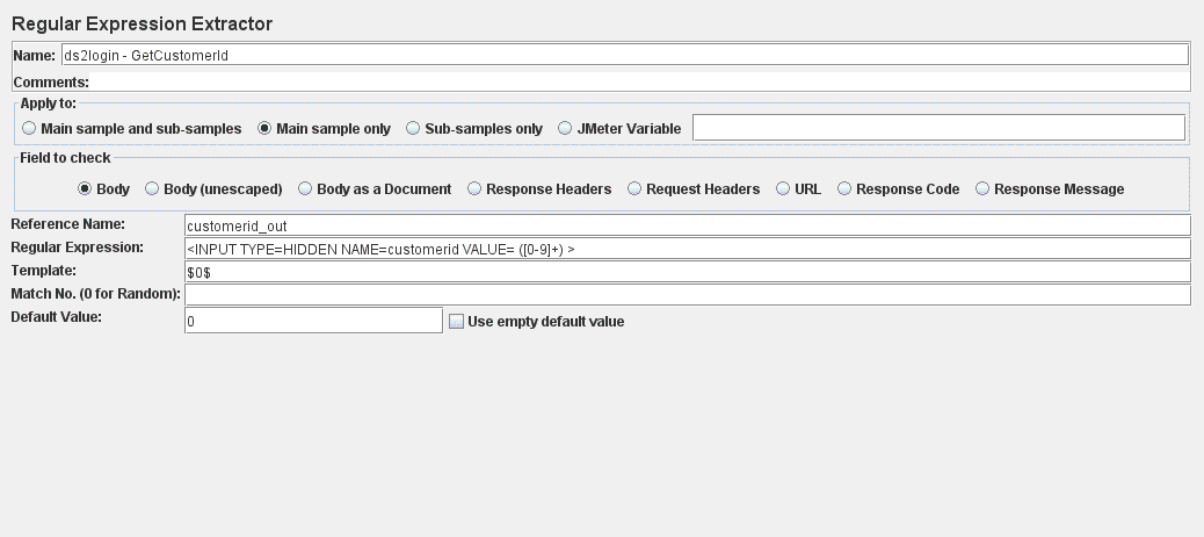



Next response text is checked to see if the following regular expression matches.

NAME=customerid VALUE= ([0-9]+) >




Last the value of customer id is extracted using a regular expression extractor and stored in *customerid_out*.




### *1.3.4.2 Step 4.B: ds2newcustomer*

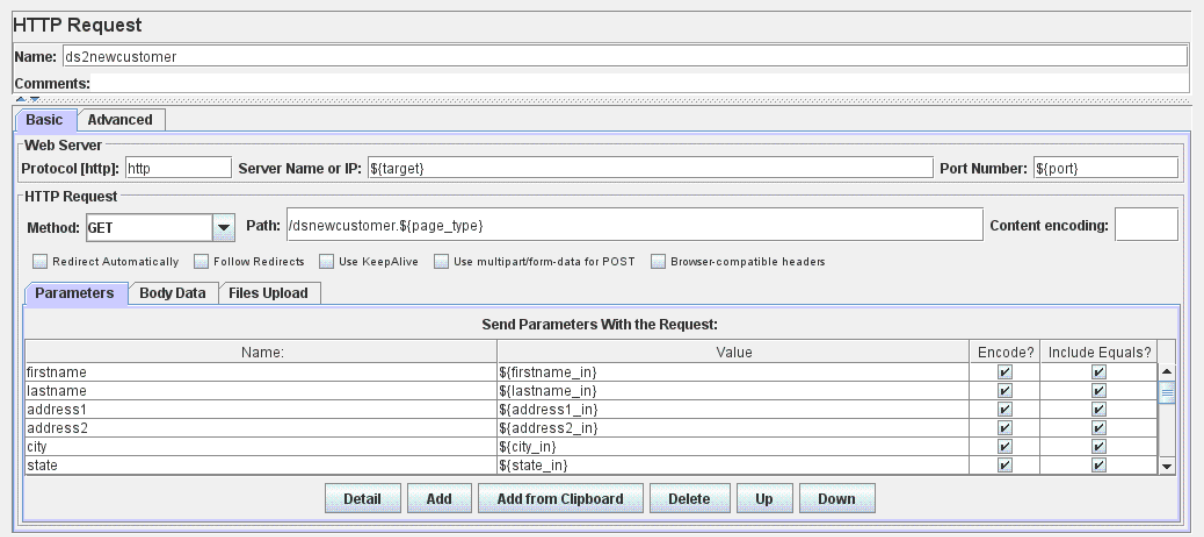
The step is an out of box JMeter HTTP request sampler named *ds2newcustomer*. The sampler executes /dsnewcustomer.jsp.

The query parameters are sent the JMeter out of box option (Send Parameters With the Request). The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent. The query parameters are as follows.

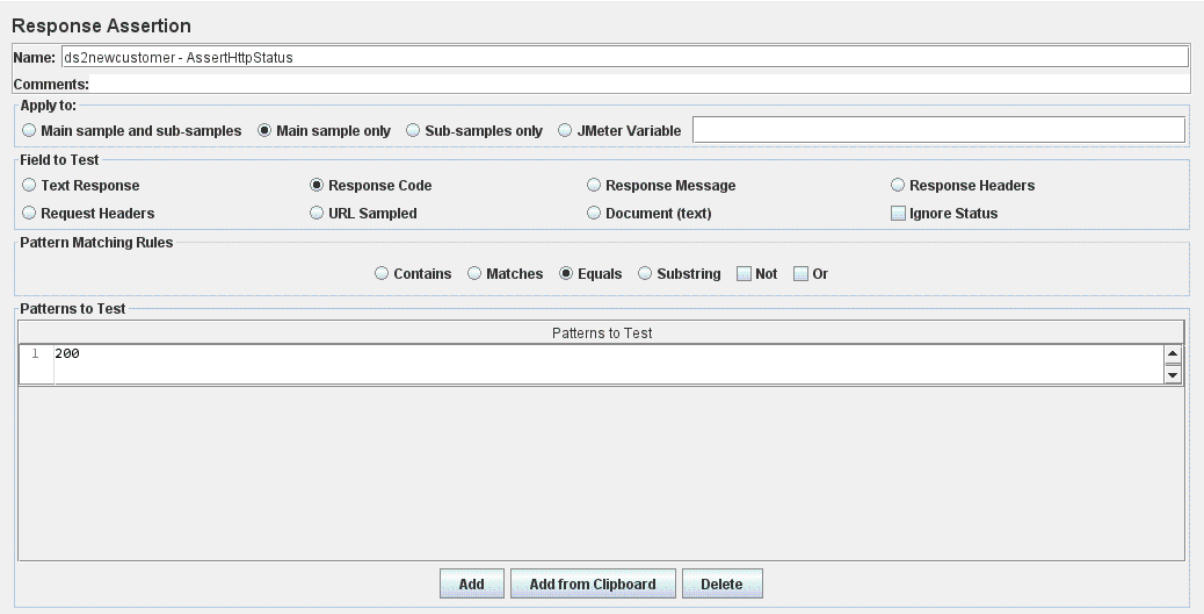
- firstname
- lastname
- address1
- address2
- city
- state
- zip
- country
- email
- phone

- creditcardtype
- creditcard
- ccexpmon
- ccexpryr
- username
- password
- age
- income
- gender

Encoding ensures unsafe characters like spaces in text values get passed.



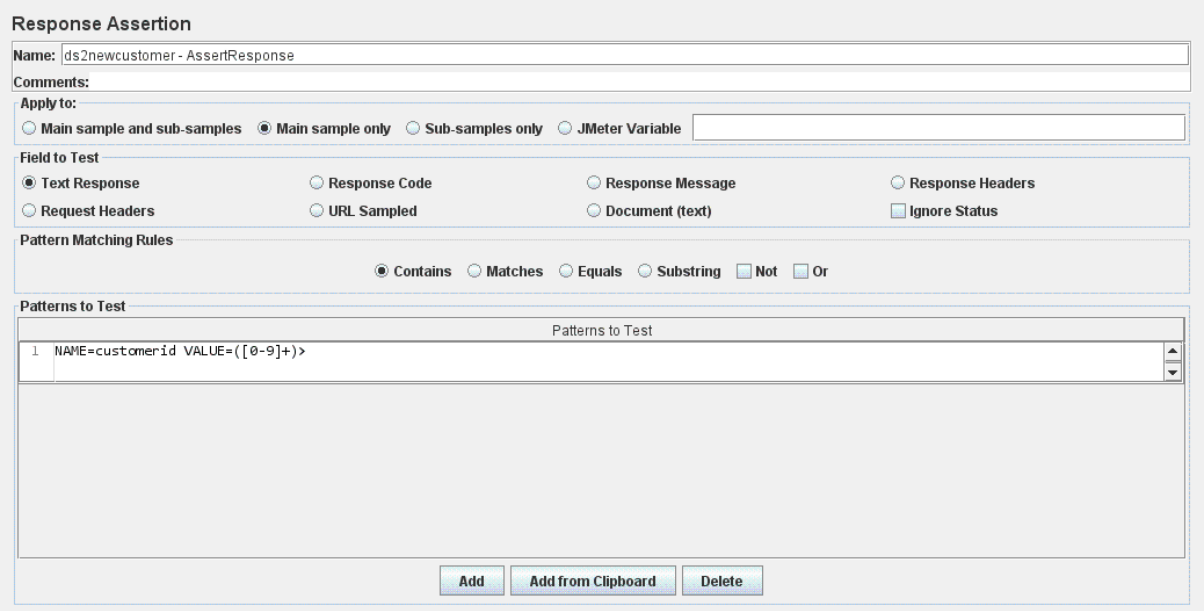

First the http response code is checked to see if it is 200.



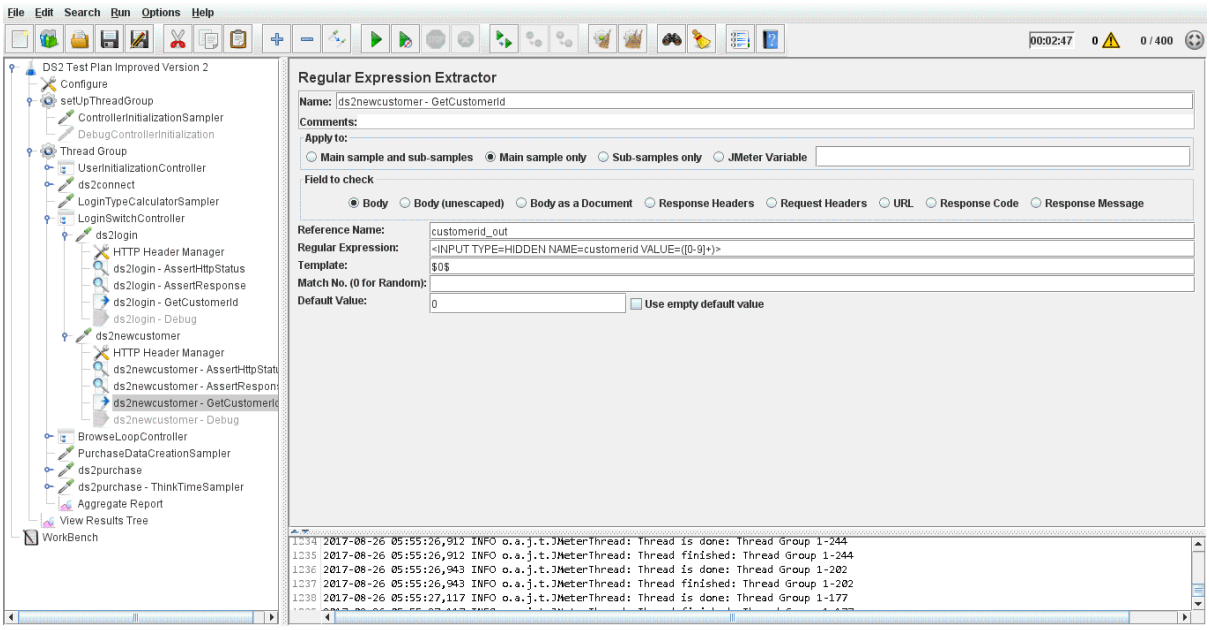

Next response text is checked to see if the following regular expression matches.

**Note: this is slightly different from the expression for ds2login.**

NAME=customerid VALUE=([0-9]+)>

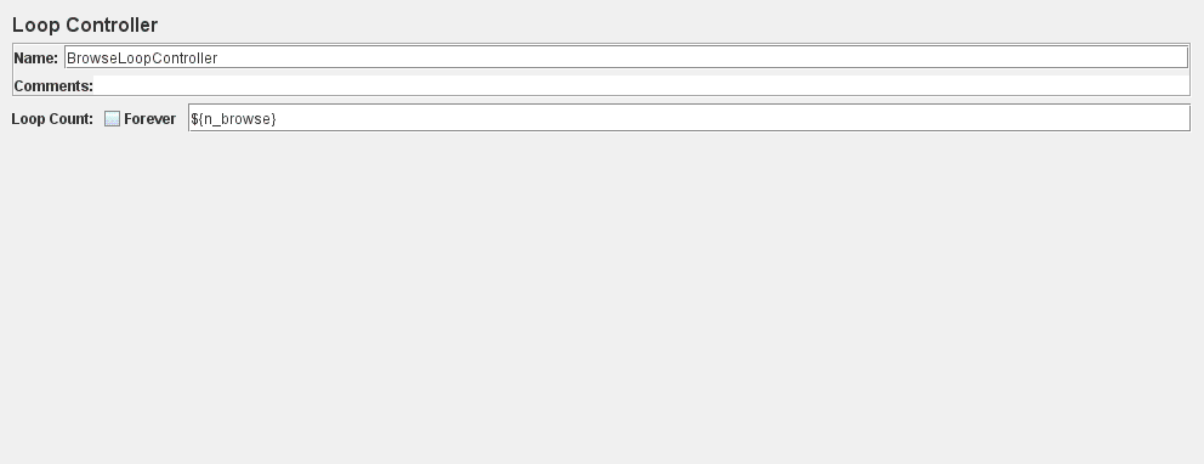



Last the value of customer id is extracted using a regular expression extractor and stored in *customerid_out*. The JMeter assigned real variable name is *customerid_out_g1*.



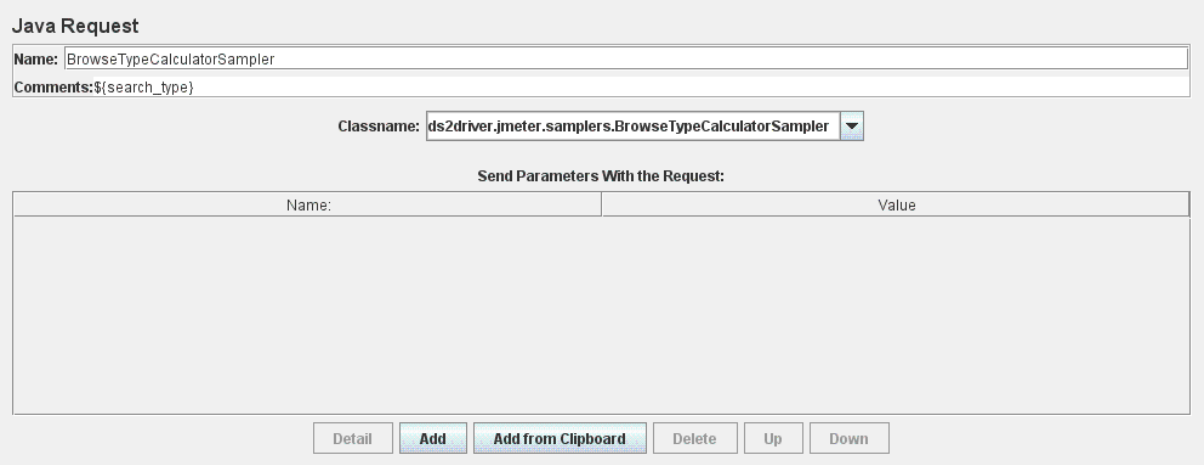

### 1.3.5 Step 5: BrowseLoopController

In the fifth step, the *BrowseLoopController* iterates *n_browse* times (calculated in Step 4).



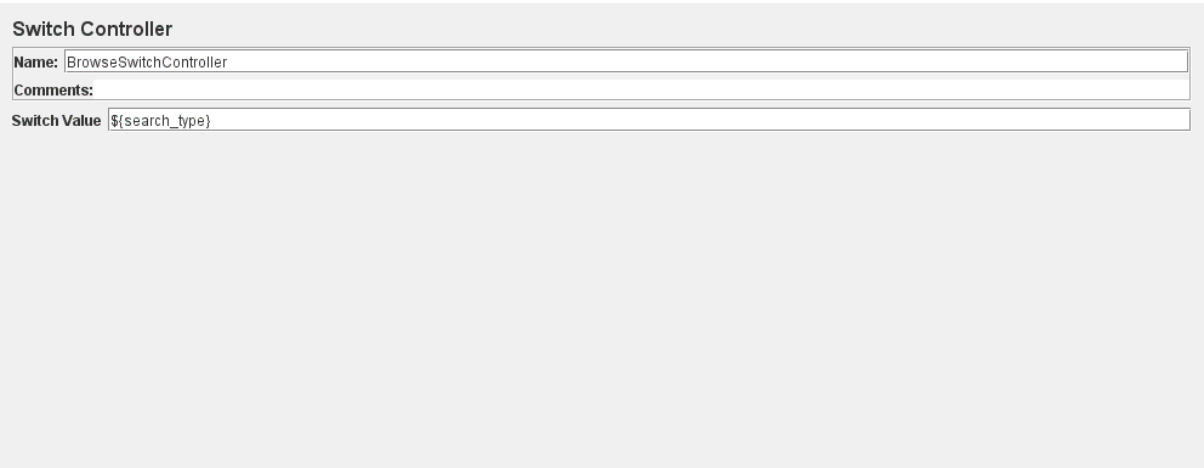

### 1.3.5.1 Step 5A: BrowseTypeCalculatorSampler

The *BrowseTypeCalculatorSampler* is a custom Java request sampler. The sampler calculates the browse type using randomization logic and the pct_newcustomers value. It can be browse by category, actor or title. The data for all browse types is created using internal functions CreateActor and CreateTitle. The data is stored in the form of JMeter variables. JMeter variables are thread specific.




### 1.3.5.2 Step 5B: BrowseSwitchController

In this step, the *BrowseSwitchController* either invokes *dsbrowse – by category, dsbrowse – by actor or dsbrowse – by title* based on the value of browse type calculated in the previous step. The variable containing the request type is *search_type*.

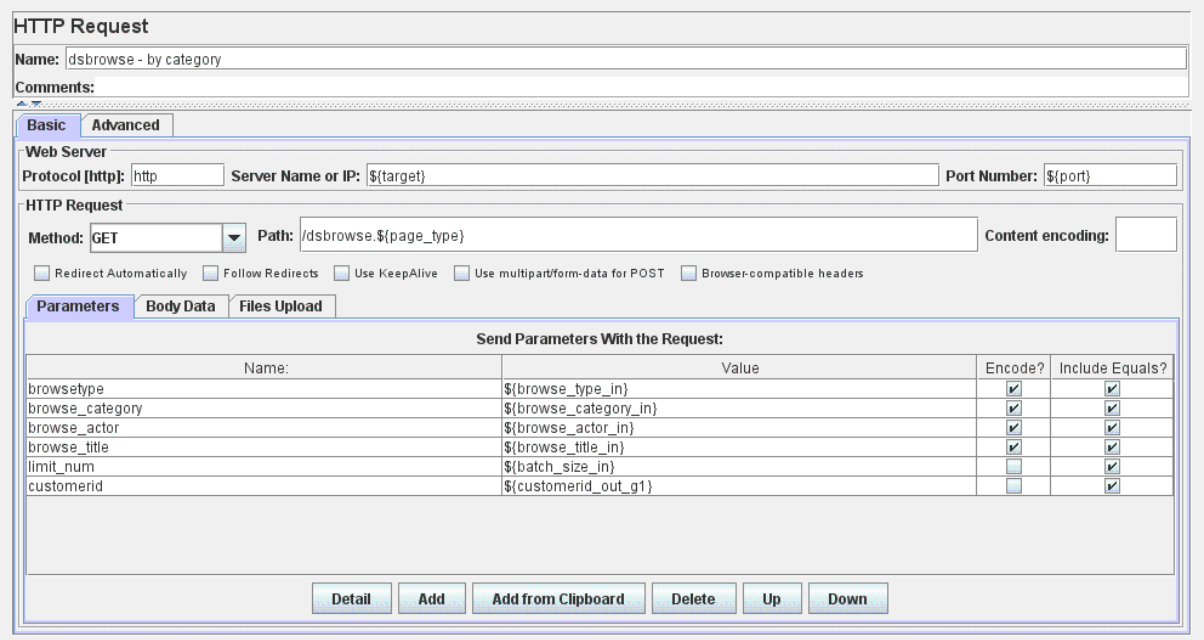


#### 1.3.5.2.1 Step 5B1: dsbrowse - by category

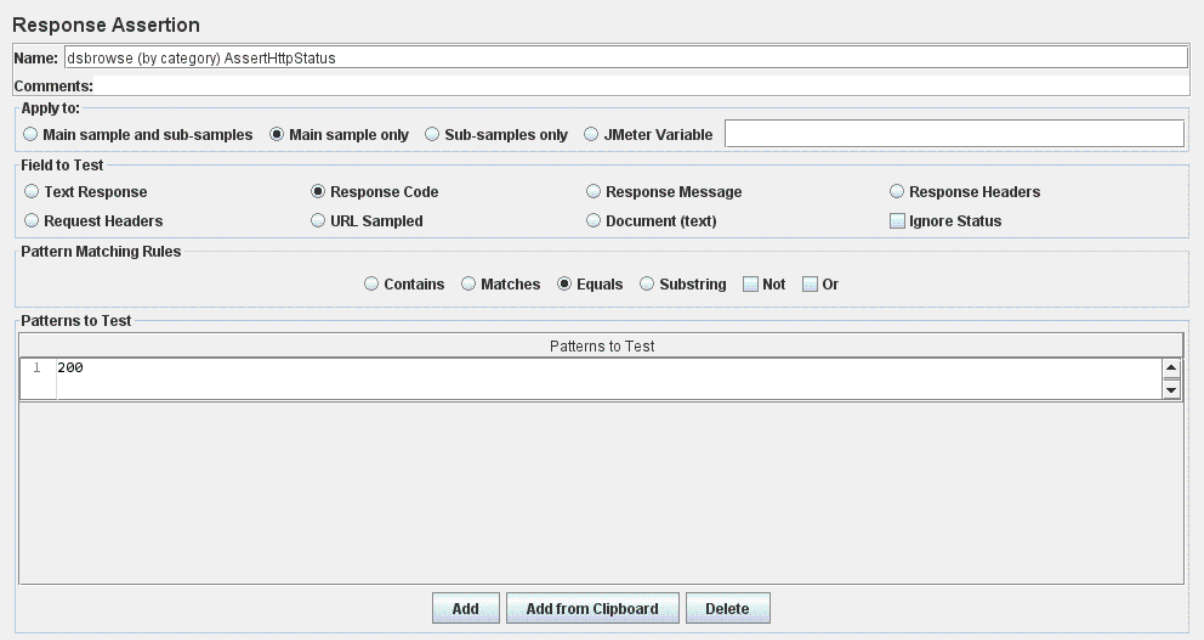
The step is an out of box JMeter HTTP request sampler named *dsbrowse – by category*. The sampler executes /dsbrowse.jsp.

The query parameters are sent the JMeter out of box option (Send Parameters With the Request). The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent. The query parameters are as follows.

- browse_type
- browse_category
- browse_actor
- browse_title
- limit_null
- customer_id

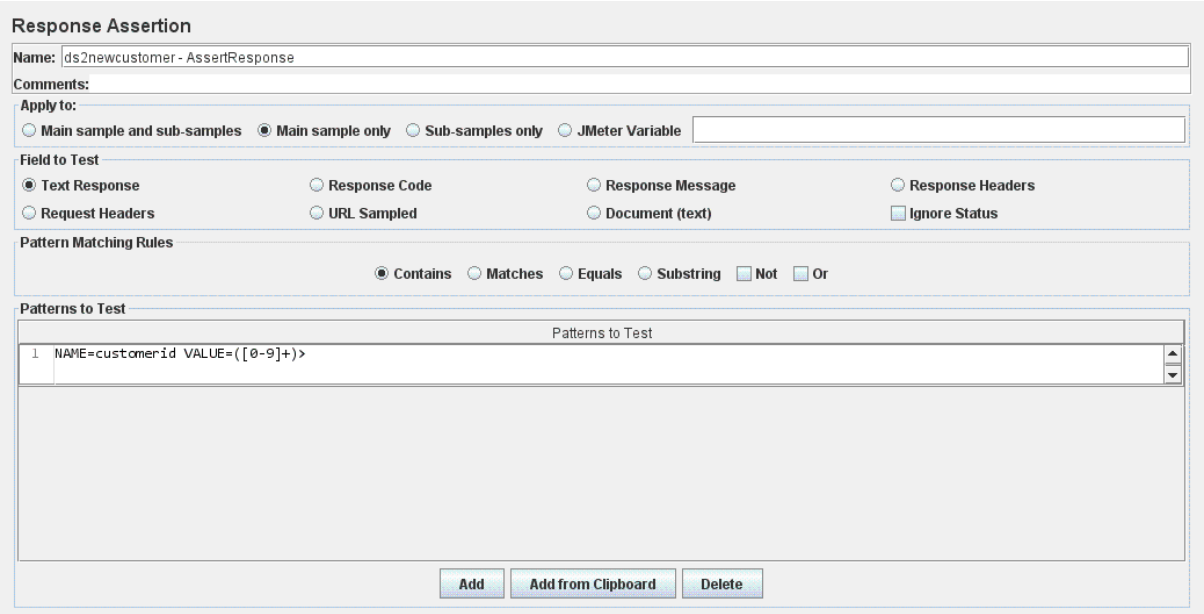



First the http response code is checked to see if it is 200.




Next response text is checked to see if the following regular expression matches.

(<H2>Search Results<\/H2>|<H2>No DVDs Found<\/H2>)



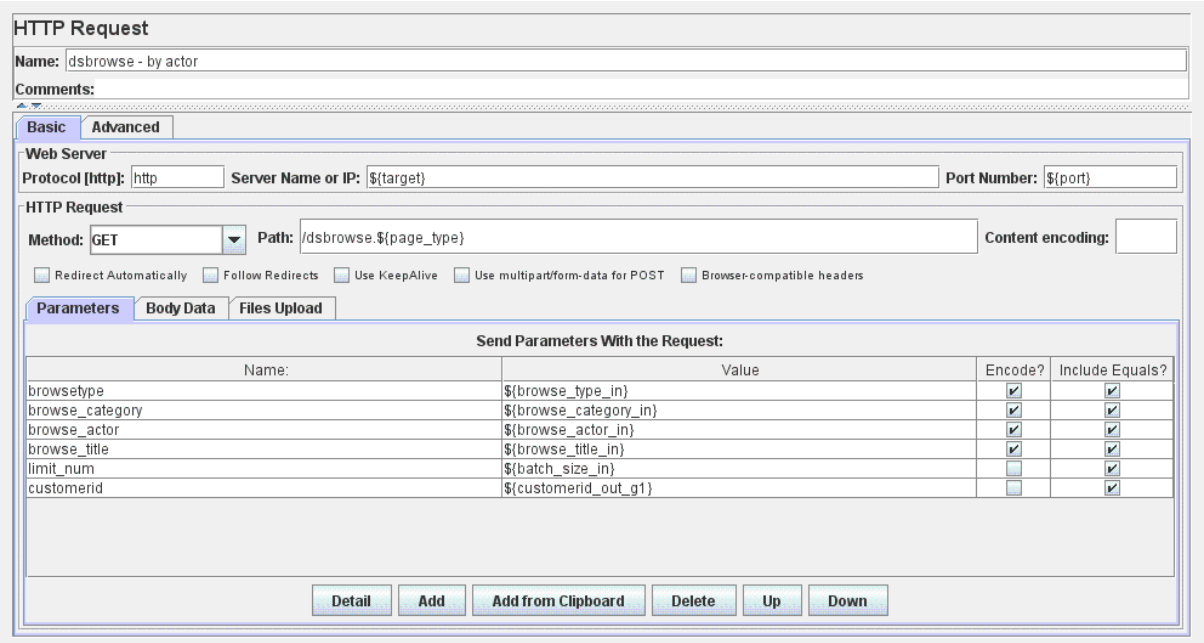

#### 1.3.5.2.2 Step 5B2: dsbrowse - by actor

The step is an out of box JMeter HTTP request sampler named *dsbrowse – by actor*. The sampler executes /dsbrowse.jsp.

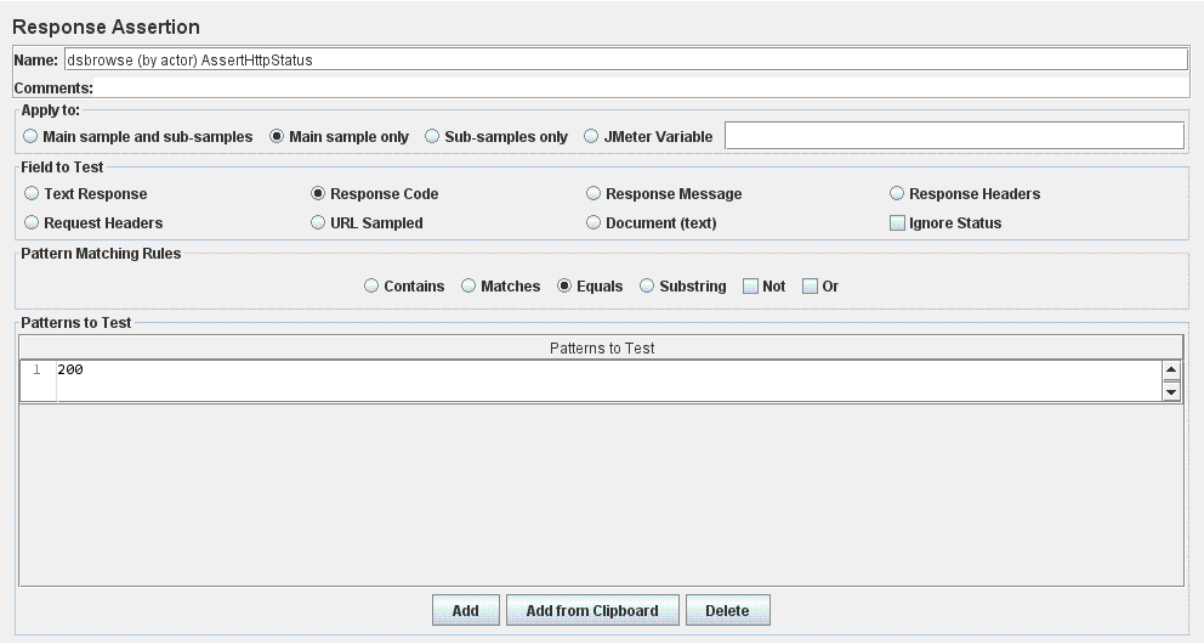
The query parameters are sent the JMeter out of box option (Send Parameters With the Request). The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent. The query parameters are as follows.

- browse_type
- browse_category
- browse_actor
- browse_title

- limit_null
- customer_id

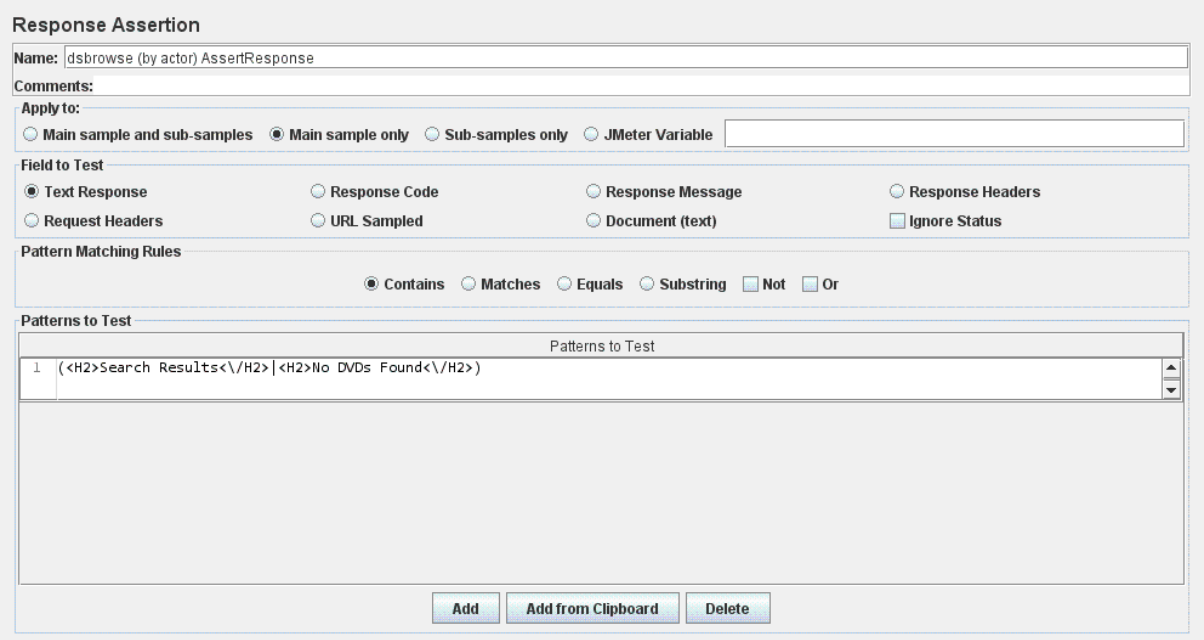



First the http response code is checked to see if it is 200.




Next response text is checked to see if the following regular expression matches.

(<H2>Search Results<\/H2>|<H2>No DVDs Found<\/H2>)

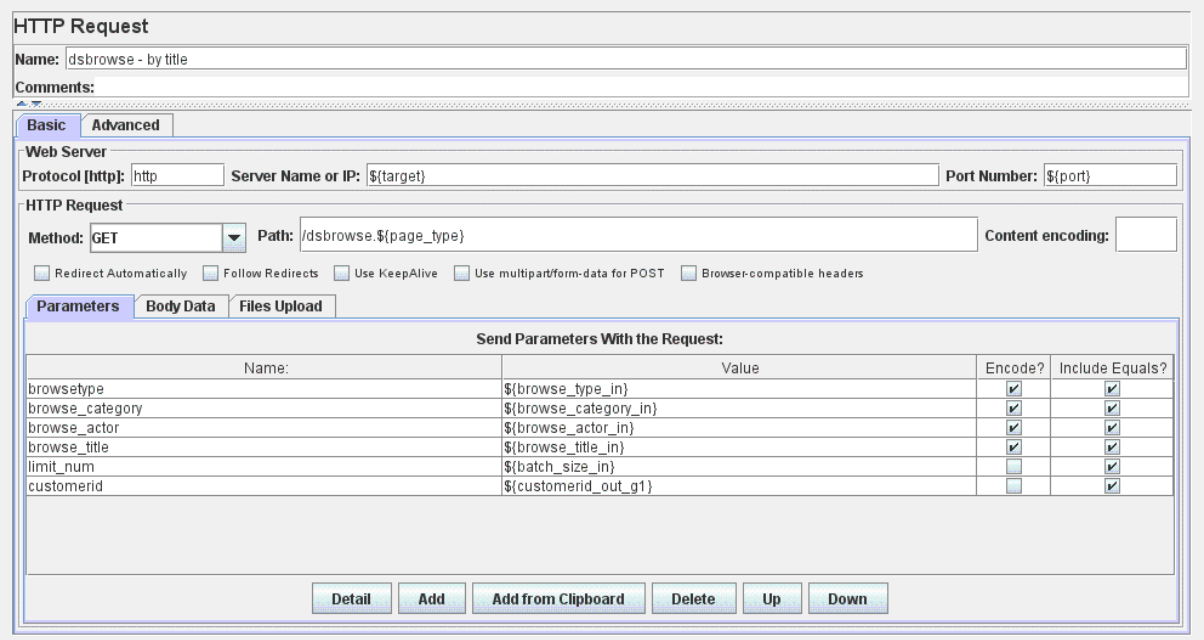



#### 1.3.5.2.3 Step 5B3: dsbrowse - by title

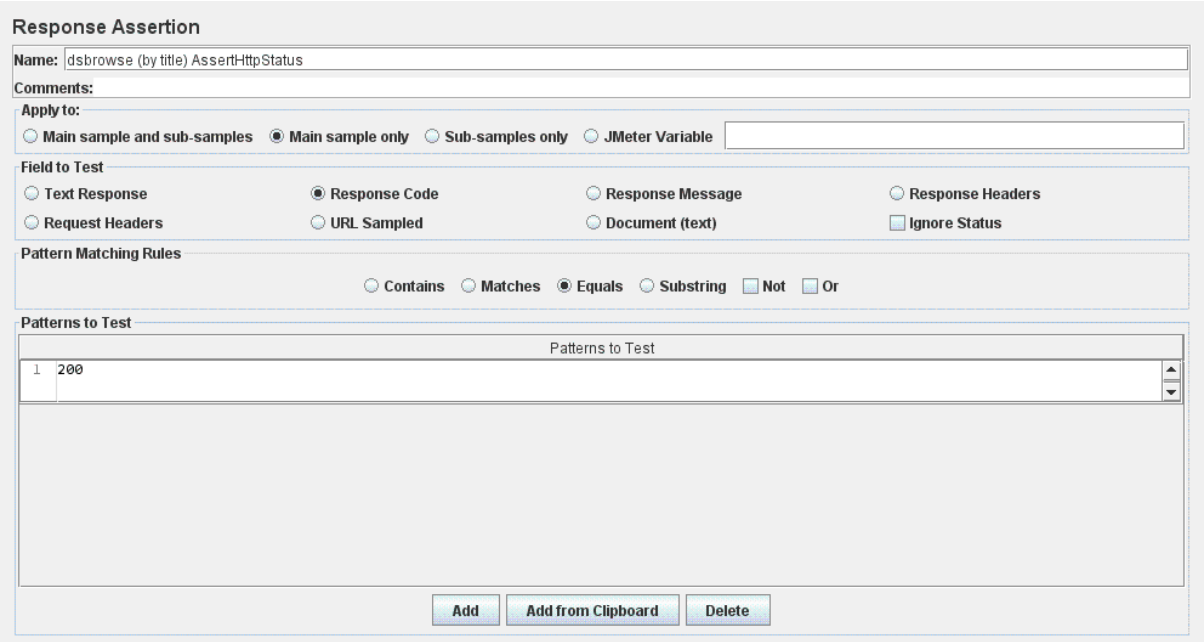
The step is an out of box JMeter HTTP request sampler named *dsbrowse – by title*. The sampler executes /dsbrowse.jsp.

The query parameters are sent the JMeter out of box option (Send Parameters With the Request). The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent. The query parameters are as follows.

- browse_type
- browse_category
- browse_actor
- browse_title
- limit_null
- customer_id

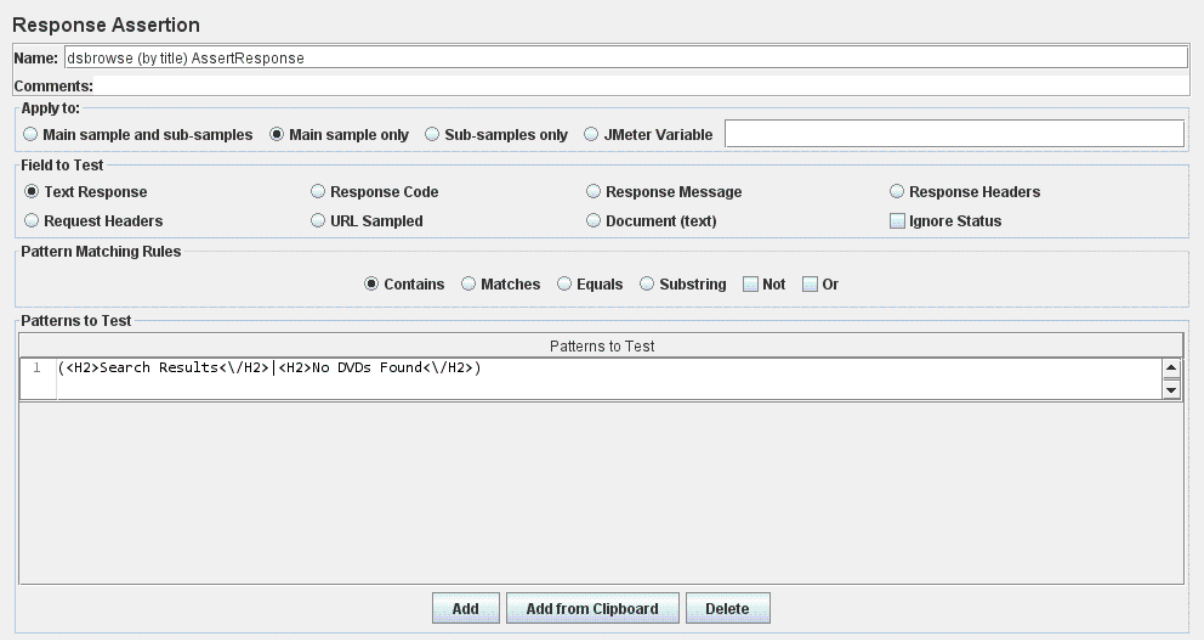



First the http response code is checked to see if it is 200.




Next response text is checked to see if the following regular expression matches.

(<H2>Search Results<\/H2>|<H2>No DVDs Found<\/H2>)

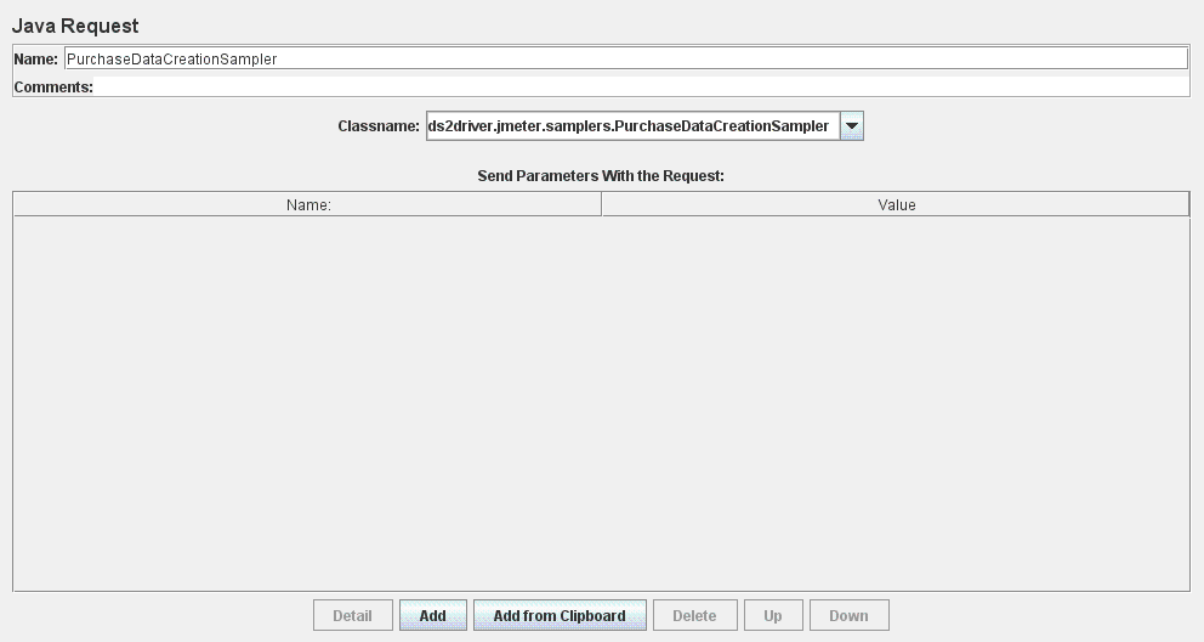



### 1.3.6 Step 6: PurchaseDataCreationSampler

The sixth step is a custom Java request sampler *PurchaseDataCreationSampler*. The sampler calculates the items and respective quantities to be purchased using randomization logic.

- Randomize number of cart items with average n_line_items
- For each cart item randomly select product_id using weighted prod_array

The data is stored in the form of JMeter variable *purchaseitemsqty*. JMeter variables are thread specific.

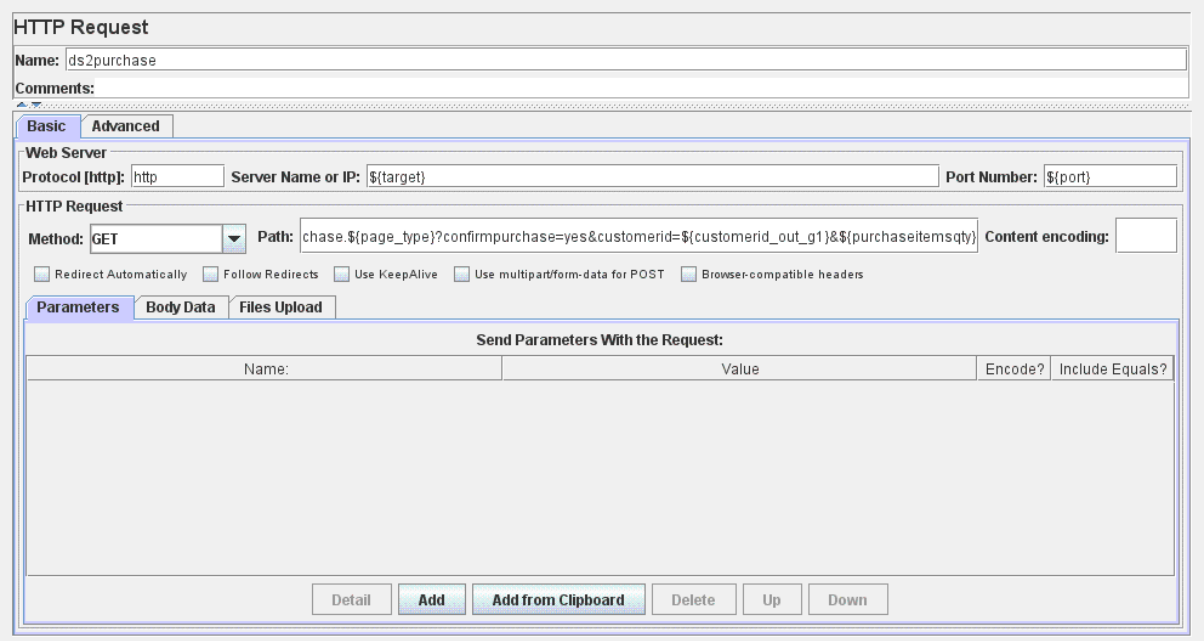



### 1.3.7 Step 7: ds2purchase

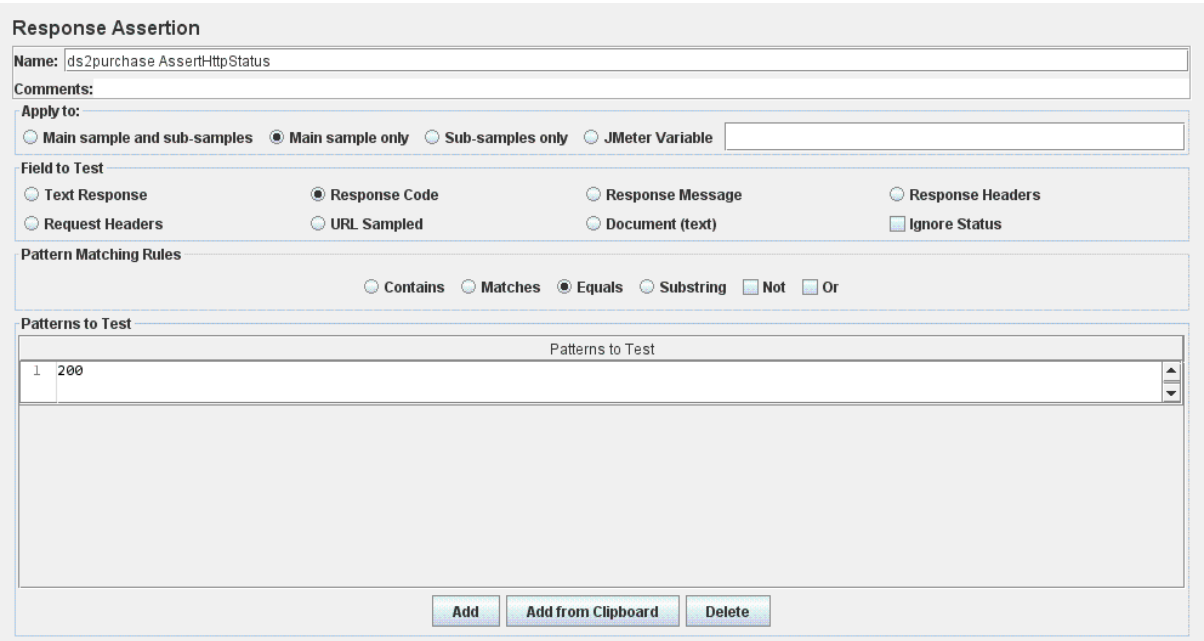
The seventh step is an out of box JMeter HTTP request sampler named *ds2purchase*. The sampler executes /dspurchase.jsp.

The query parameters are sent the JMeter out of box option (Send Parameters With the Request). The default request headers sent are Accept-Language, Accept-Encoding, Accept and User-Agent. The query parameters are as follows.

- confirmpurchase
- customerid
- purchaseitemsqty

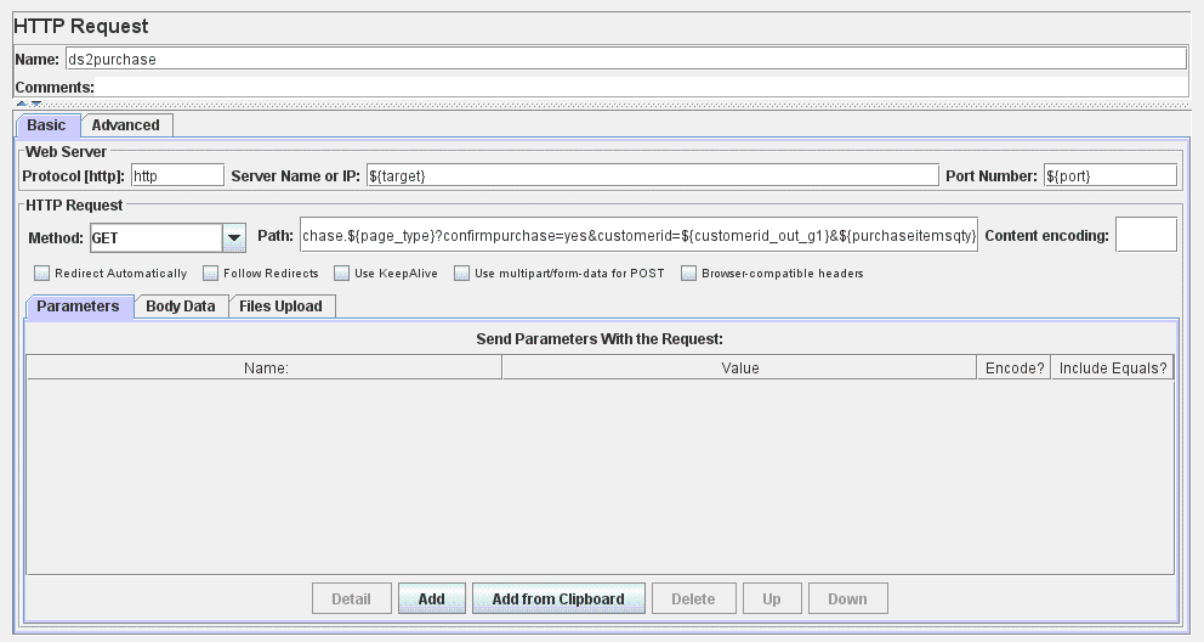



First the http response code is checked to see if it is 200.



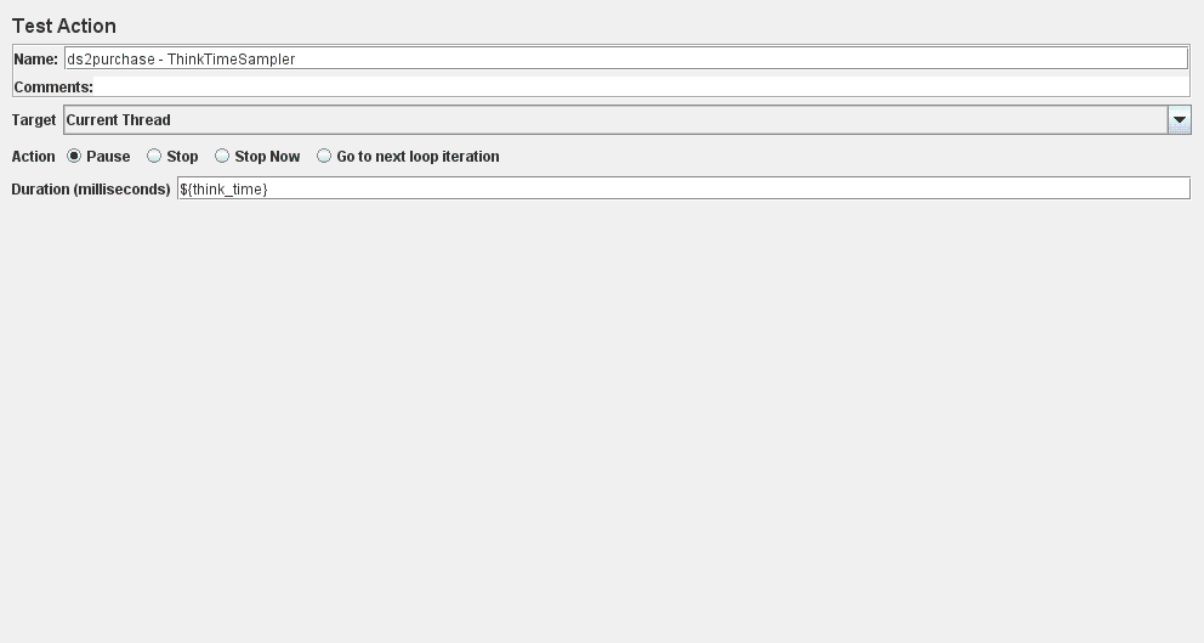

Next response text is checked to see if the following regular expression matches.

(<H2>Order Completed Successfully --- ORDER NUMBER: ([0-9]+)<\/H2><BR>|<H3>Insufficient stock - order not processed<\/H3>)



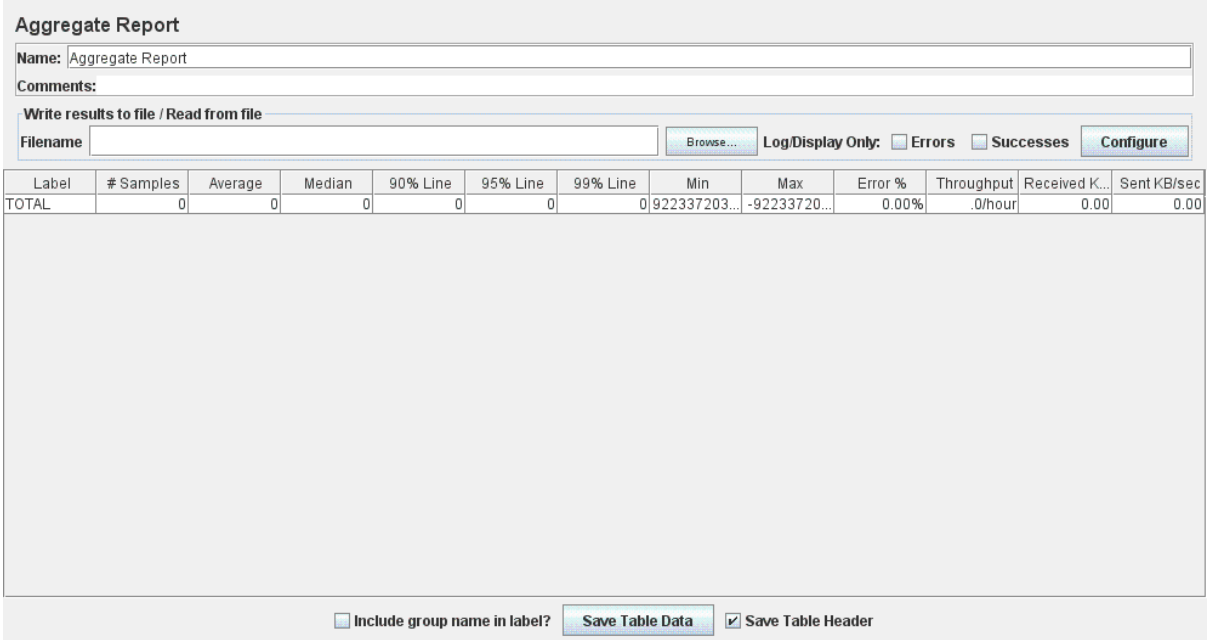

### 1.3.8 Step 8: ds2purchase – ThinkTimeSampler

The eight and last step is a Test Action sampler which pauses the current thread by the specified value.



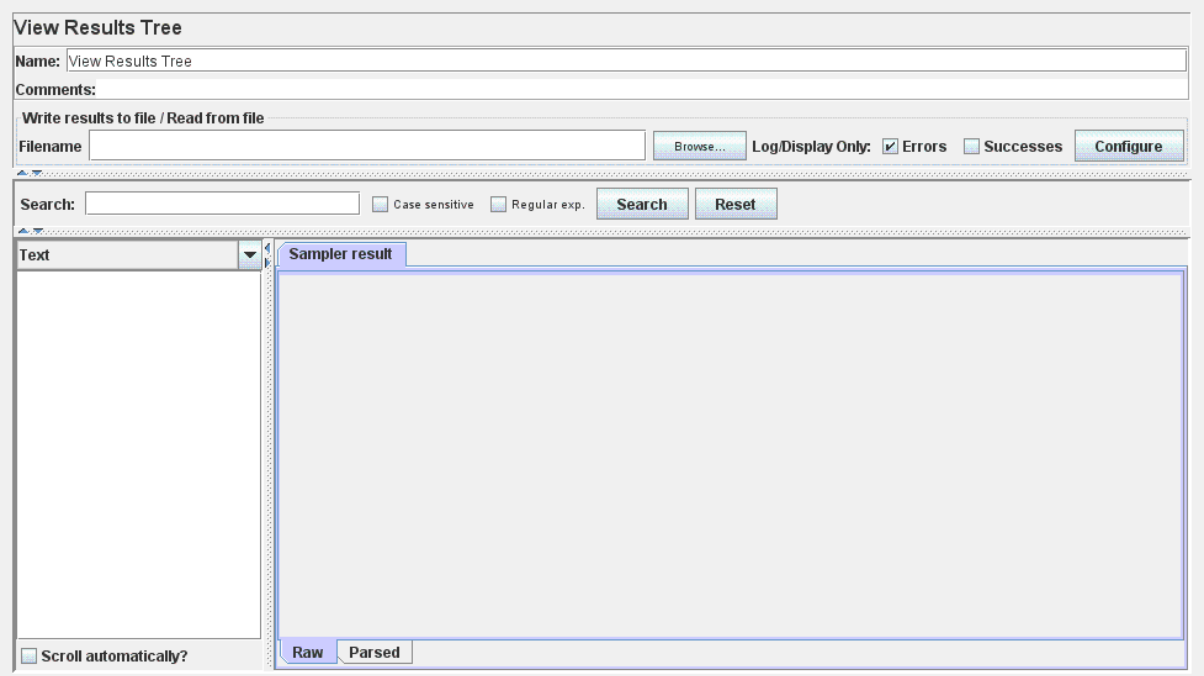

### 1.3.9 Aggregate Report

Displays sampler wise aggregated report containing response time and error statistics.




## 1.4 View Results Tree

Displays granular details of errors. Only the error option has been enabled.




# 2 Java Project

The DS2JMeter project contains the custom Java samplers and beans.

- Beans
  - GlobalConstants
  - GlobalParametersSingleton
  - User

- Samplers
  - BrowseTypeCalculatorSampler
  - ControllerInitializationSampler
  - LoginTypeCalculatorSampler
  - PurchaseDataCreationSampler
  - UserInitializationSampler

Following external jars are required by the project. The jars are present either in the /lib or /lib/ext of the JMeter folder.

- ApacheJMeter_core
- ApacheJMeter_java
- slf4j-api-1.7.25
- slf4j-ext-1.7.25

DS2JMeter is compiled and exported as JMeterClassesForDS2.jar to the /lib/ext folder of JMeter.

# 3 Test Runs

Results of test run with 400 virtual users and 10 iterations are enclosed. Default input parameters were used. The run was executed on the Microsoft Azure cloud with three VMS.

- VM1: Basic A2 (2 vcpus, 3.5 GB memory), Windows OS, Apache JMeter 3.2
- VM2: Basic A1 (1 vcpu, 1.75 GB memory), CentOS 6.8, Tomcat 9.0.0.M22 with DS2 jsp web application, JDK 1.8, threads = 800
- VM3: Basic A1 (1 vcpu, 1.75 GB memory), CentOS 6.8, MySQL with DS2 database, connections = 800

  innodb_buffer_pool_size = 1G
  innodb_log_file_size = 256M
  innodb_log_buffer_size = 8M #default value
  innodb_flush_method = O_DIRECT

# 4 Additional Points

Following features are not available [can be considered in the next release].

- ramp_rate
- run_time
- warmup_time

Following logic is not available [can be considered in the next release]

- Initialization of random function with very randomized seed

```
      // Create random stream r with very randomized seed
      Random rtemp = new Random ( Userid * 1000 ); // Temporary
seed
      // For multi-thread runs sleep between 0 - 10 second to
spread out Ticks (100 nsecs)
      if ( Controller.n_threads > 1 ) Thread.Sleep ( rtemp.Next (
10000 ) );
      long DTNT = DateTime.Now.Ticks;
      uint lowDTNT = ( uint ) ( 0x00000000ffffffff & DTNT );
      uint rev_lowDTNT = 0;  // take low 32 bits of Tick counter
and reverse them
      for ( i = 0 ; i < 32 ; i++ ) rev_lowDTNT = rev_lowDTNT | (
( 0x1 & ( lowDTNT >> i ) ) << ( 31 - i ) );
      //Console.Error.WriteLine("DTNT= 0x{0:x16}  lowDTNT= 0x
{1:x8}  rev_lowDTNT= 0x{2:x8}", DTNT, lowDTNT, rev_lowDTNT);
      r = new Random ( ( int ) rev_lowDTNT );
```

- Loop for new user creation if an “already exists” error is returned [can be considered in the next release]

```
            if ( customerid_out == 0 ) Console.Error.WriteLine (
"User name {0} already exists" , username_in );
            } while ( customerid_out == 0 ); // end of do/while
try newcustomer
```

The possibility is reduced by further randomizing the user name. The new user names are not used in the login logic.

```
            //added some more randomization with alphabets to avoid the logic in
            //existing workload generator which keeps generating new customer
            //until find a userid that doesn't exist
            String username_in = "newuser" + ( char ) ( 97 + user.nextInt ( 26 ) ) + ( char )
( 97 + user.nextInt ( 26 ) ) + i_user;
```

- Detailed parsing of responses was not included, as it did not appear to be used in the driver logic. Existing driver logic example is given below. JMeter assertions have been added to act as verification points.

```
        i_row = 0;
        str_acc = str_acc.Substring(str_acc.IndexOf("<TABLE"));
// Snip off everything up to <TABLE> tag
        str_acc = str_acc.Substring(4 + str_acc.IndexOf
("<TR>")); // Skip first <TR> tag
        while (str_acc.IndexOf("<TR>") > 0)
          {
          str_acc = str_acc.Substring(str_acc.IndexOf("<TR>"));
// Find <TR> tag
          str_acc = str_acc.Substring(6 + str_acc.IndexOf
("VALUE"));
          ind_e = str_acc.IndexOf(">");
          prod_id_out[i_row] = Convert.ToInt32(str_acc.Substring
(0, ind_e).Trim('"'));
          str_acc = str_acc.Substring(4 + str_acc.IndexOf
("<TD>")); // Find <TD> tag
          ind_e = str_acc.IndexOf("<");
          title_out[i_row] = str_acc.Substring(0, ind_e);
          str_acc = str_acc.Substring(4 + str_acc.IndexOf
("<TD>")); // Find <TD> tag
          ind_e = str_acc.IndexOf("<");
          actor_out[i_row] = str_acc.Substring(0, ind_e);
          str_acc = str_acc.Substring(4 + str_acc.IndexOf
("<TD>")); // Find <TD> tag
          ind_e = str_acc.IndexOf("<");
          price_out[i_row] = Convert.ToDecimal(str_acc.Substring
(0, ind_e));
          ++i_row;
          ++rows_returned;
          }
```

- Changed the data point O'DONNELL to O DONNEL as it was resulting in a MySQL error because of the single quote.
- The existing logic has a constant think time only at the end of all the steps. However, the practitioner can experiment with the timer – think time after every step, to simulate more realistic patterns.
- Currently the following input values are stored in the *User* bean. These will be moved to the *GlobalParametersSingleton* bean as the values remain same for all the threads [can be considered in the next release].
- Performance metrics calculated by the existing driver is not included [can be considered in the next release]