Getting Started with JMeter

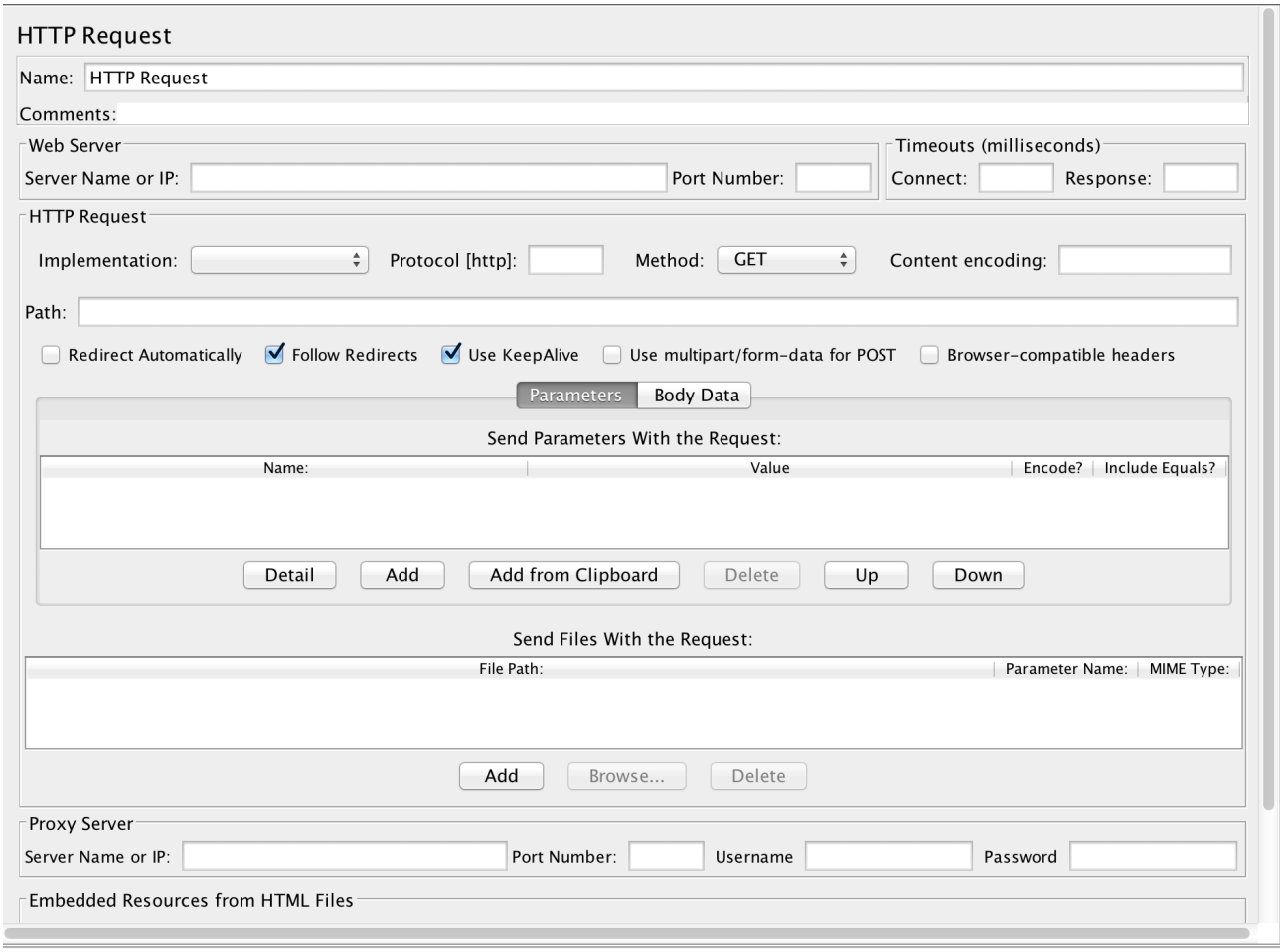
I think the best way to start learning JMeter is to script a user journey through a web application; logging in, searching for and adding items to a basket, checking out, the usual stuff.

Some of the steps to achieve this are actually quite complicated once you start digging into the detail, so I'm going to split them up over a series of sections. This one covers navigating to a web page and asserting that it's the correct one once you get there.

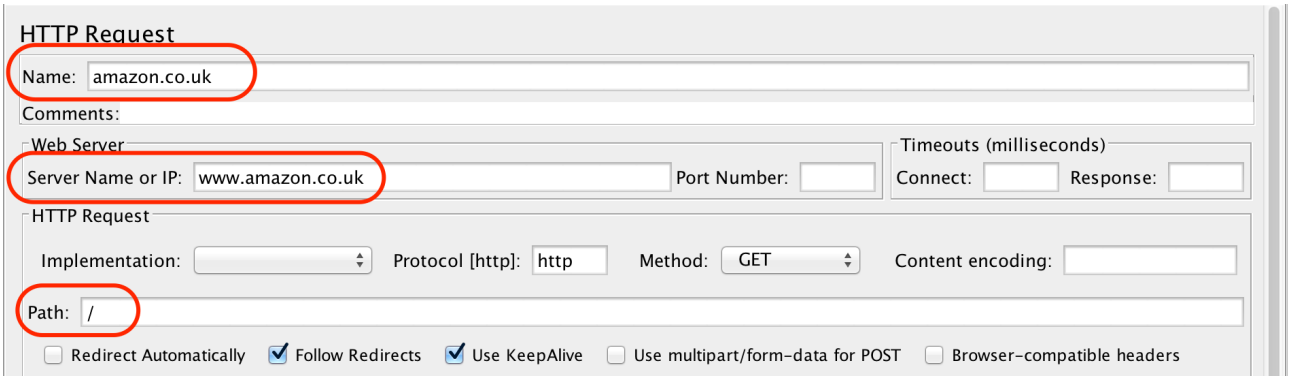
If you're following along, the first things you'll need to do before we actually get into the details are:

1. Open up JMeter. I'm assuming that you've downloaded it and installed it on your machine. If you haven't, you can download it from here. I use the word installed loosely since, as long as you have a Java Runtime Environment [JRE] on your machine, JMeter should work fine. If you have some problems with this step then they're probably machine/environment specific; I recommend a quick hunt for the solution via your favourite search engine.
2. Once JMeter is open - right click on the Test Plan icon and add a Thread Group via Add > Threads (Users) > Thread Group. Don't worry too much about what a Thread Group actually is at this point; I'll cover that some other time.
3. Right click on the Thread Group and add a Simple Controller via Add > Logic Controller > Simple Controller
4. Right click on the Simple Controller and add a HTTP Request via Add > Sampler > HTTP Request

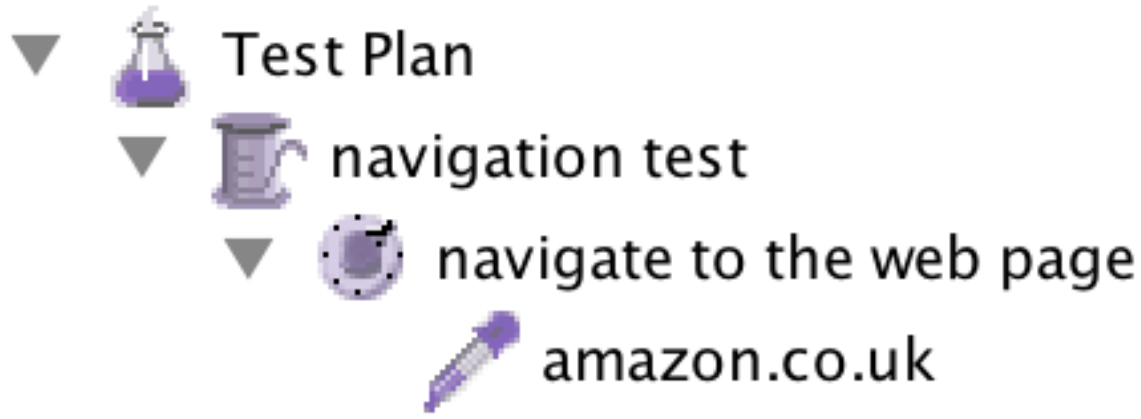
A blank HTTP Request looks like this:



There's a lot of stuff there that you don't need right now. You only need to fill out the Name, Server Name or IP and Path fields so they look like the below:

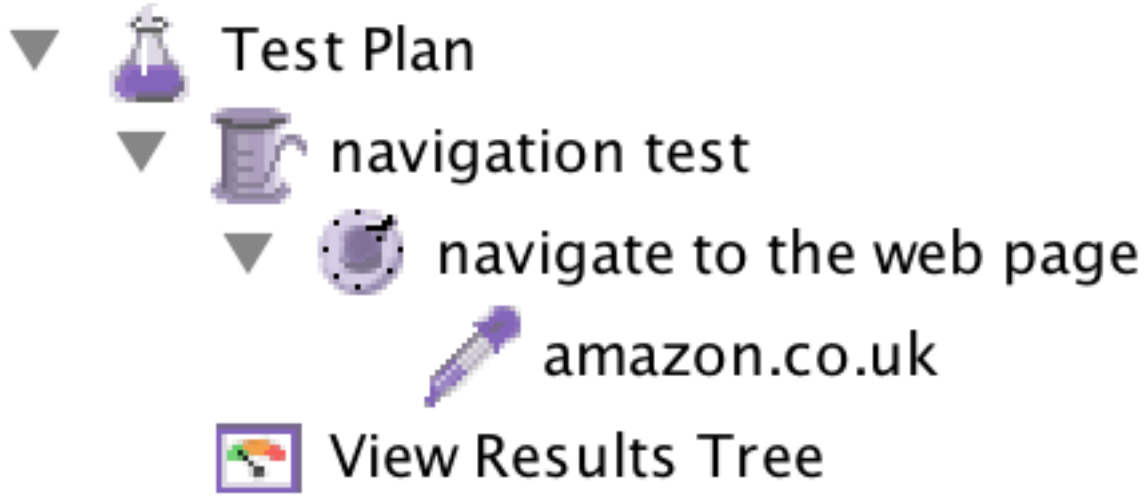


Having done so - your Test Plan should look more or less like the below:

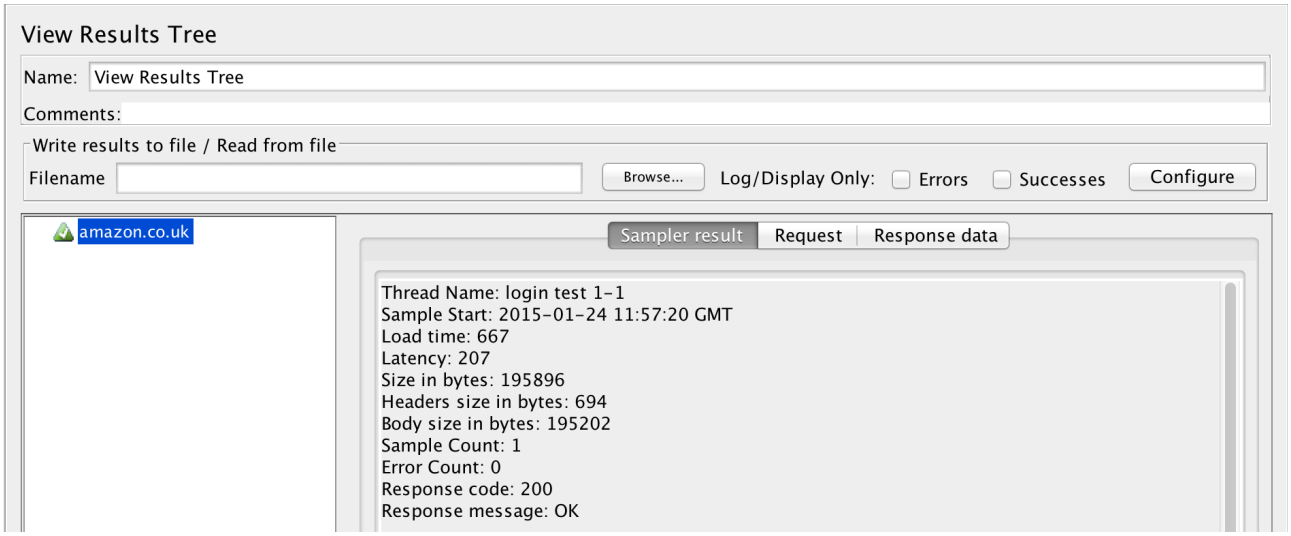


Notice that I've given my Thread Group and Simple Controllers meaningful names. It's not crucial that you do that right now, but once your tests become more complex, it will be.

If we go ahead and run our test right now, we'll see something has happened, but we've no way of checking exactly what at the moment, You need to add a Listener to show you what's going on with your test. Do that now by right clicking on the Test Plan icon, then Add > Listener > View Results Tree.

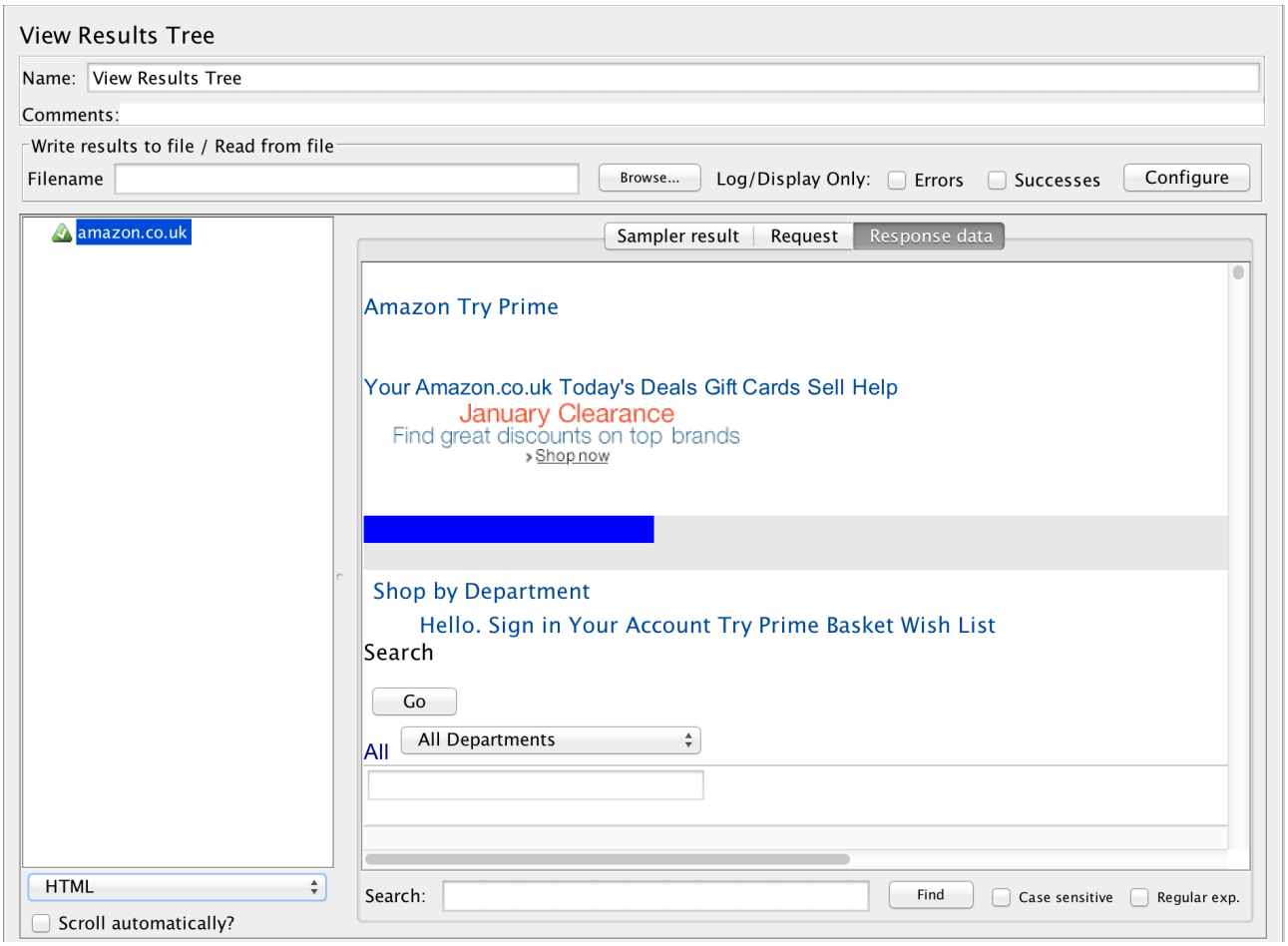


Now click on the View Results Tree icon and run the test again. You should see a result like this:



A Response Code of 200 means our HTTP Request was fulfilled ok. Some of the other results may be of interest too, but I'll cover those another time.

Clicking on the Response Data tab and then selecting the HTML view (in the dropdown just above the Scroll automatically checkbox) will render some of the HTML for us.

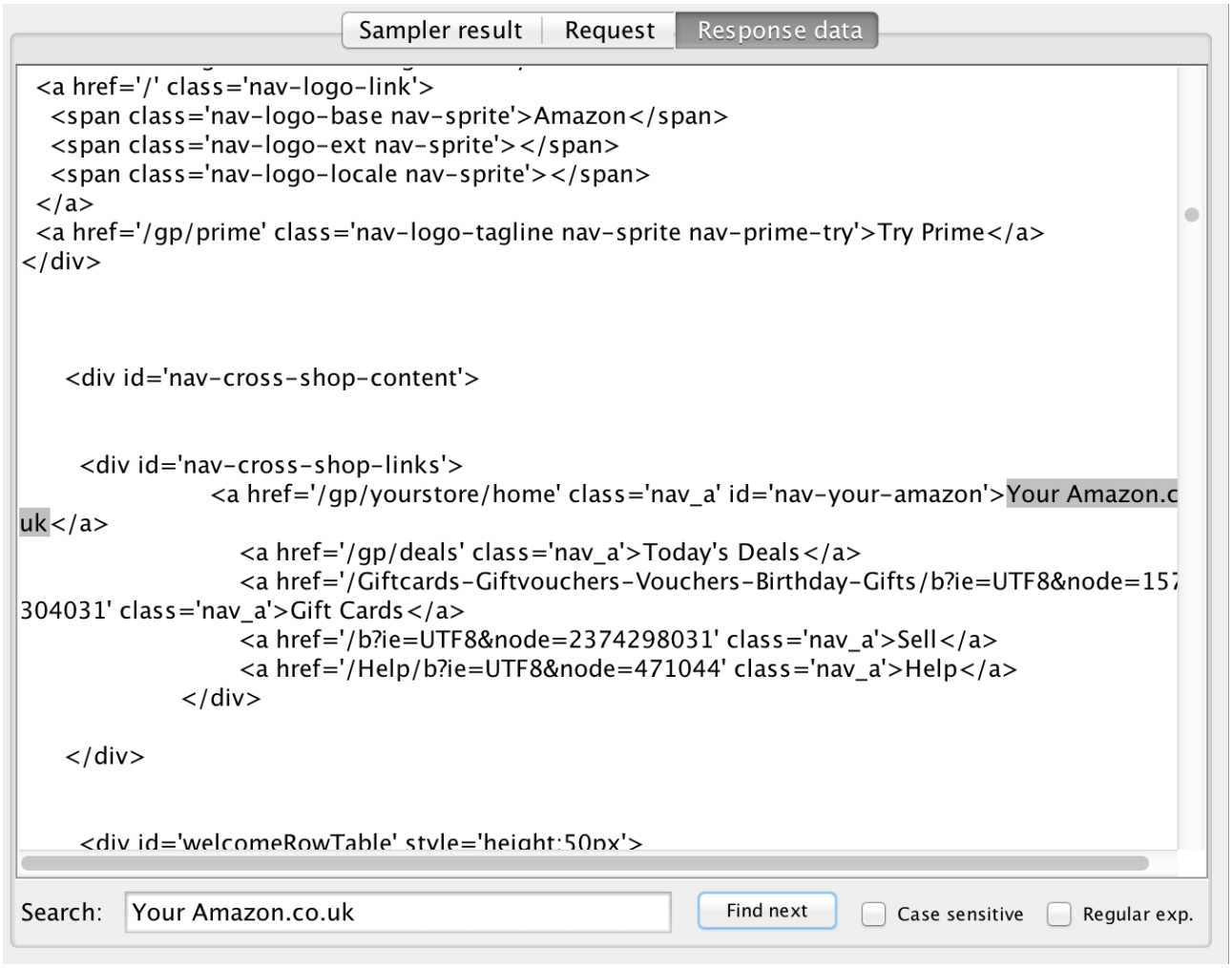


It looks kind of basic because JMeter doesn't load the other page resources (images etc) by default. We can ask it to, but there's no need right now.

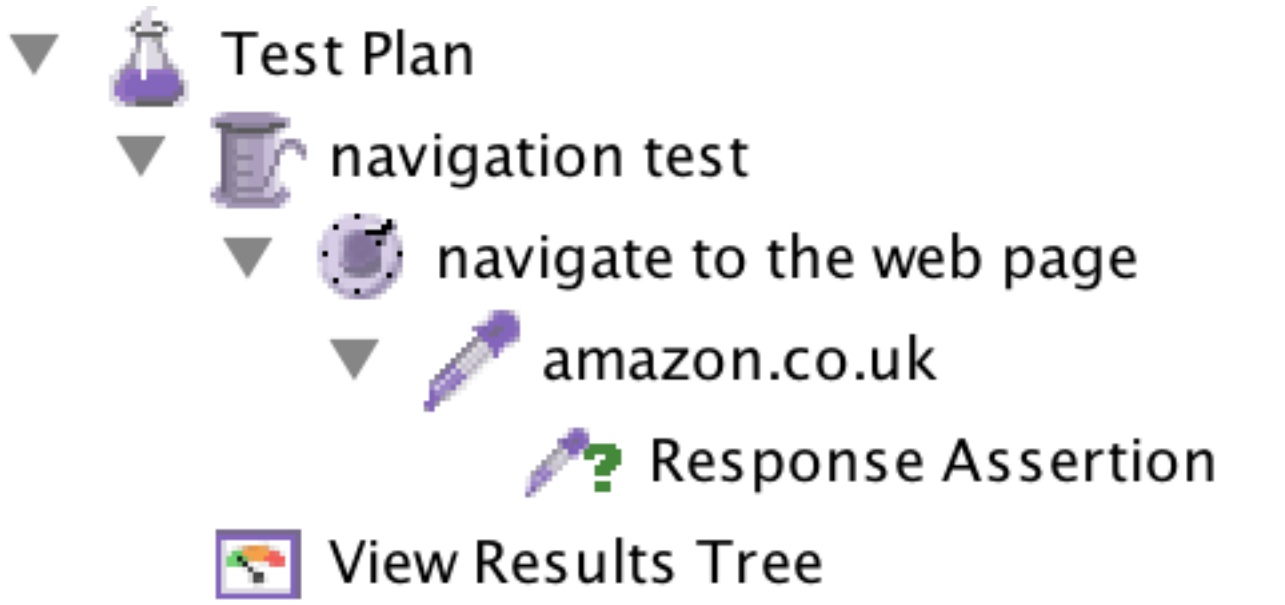
By looking at the response we can see that the page we requested has been returned, but we need to add some kind of check to do that for us in future. We want JMeter to carry out an assertion on the page and verify that it can see what we think it should see. Probably the simplest thing to do at this stage is to check for something specific to prove we're on the correct page.

In the response we can see the text "Your Amazon.co.uk".

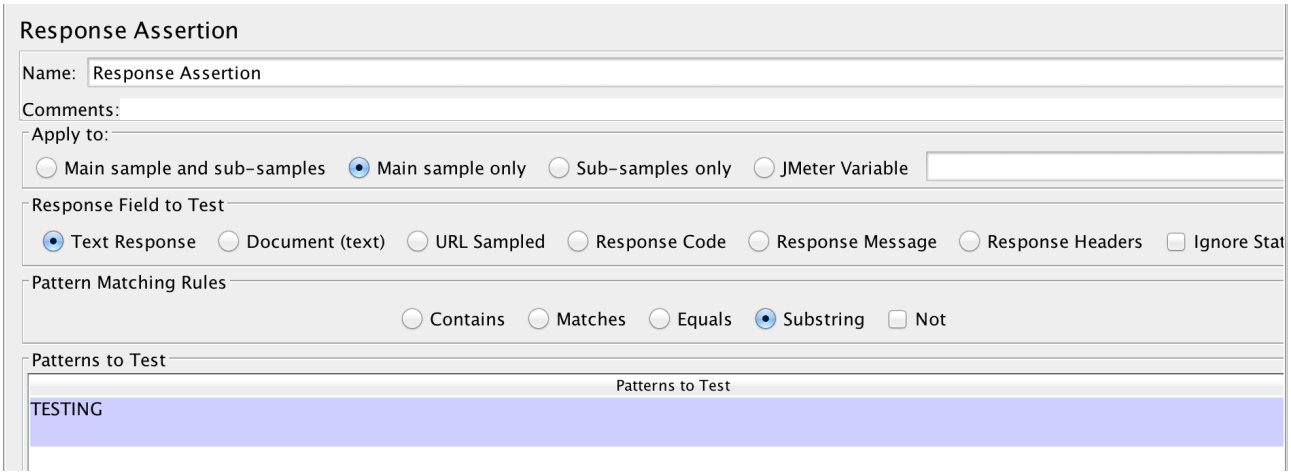
If I switch to text view in the response data and do a Search for the same text, JMeter finds it ok - which suggests to me it's a good candidate for an assertion.



So let's add one by right clicking on the HTTP Request and adding an assertion via Add > Assertions > Response Assertion.



Open up the Response Assertion and click on the Add button at the bottom of the page. This adds a new assertion pattern. To verify that our assertion is going to do the right thing, let's start with a failing test. Add the word "TESTING" into the assertion pattern so it looks like the below:

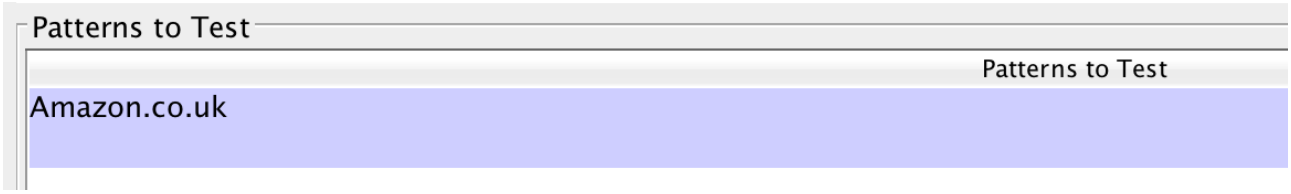


Now run your test. It should fail.

Expanding on the failed response will open up the Assertion result, and you should see something similar to the below:

```
Assertion error: false
Assertion failure: true
Assertion failure message: Test failed: text expected to contain /TESTING/
```

Now add the text “Your Amazon.co.uk” to the assertion pattern instead of “TESTING” and run your test again.



You should see a green result again in the Results Tree, indicating that the assertion was successful.

Congratulations. You have your first passing JMeter test! :-)

Using the JMeter HTTP(S) Test Script Recorder

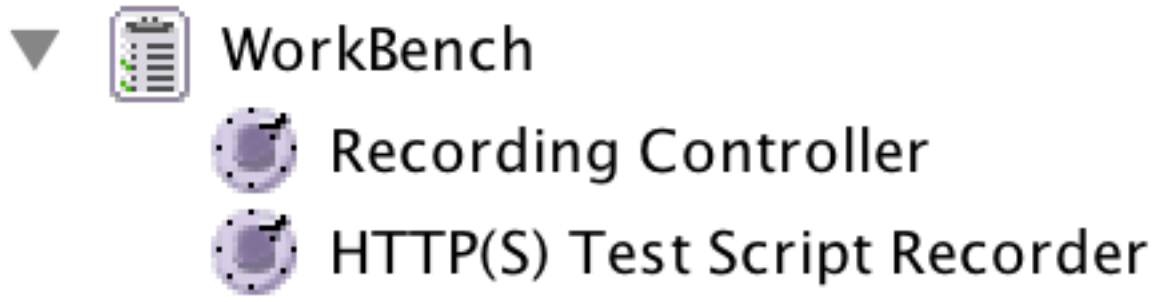
In my last JMeter post I showed you how to navigate to a web page and assert it’s the right one once you get there. Soon, I want to show you how to login to a web app, but before that it’s time to take a look at the JMeter HTTP(S) Test Script Recorder.

The JMeter HTTP(S) Test Script Recorder is the new(-ish) name for what used to known as the JMeter Proxy Recorder. It’s JMeter’s utility for capturing requests that are initiated from the browser for subsequent analysis in JMeter. In order to successfully craft a login script, we need to know how to build our requests so that the web app’s servers will accept them as being valid requests. The easiest way to get this information into JMeter is to record it via the proxy (as it will subsequently be referred to, because the alternative is too much of a mouthful!).

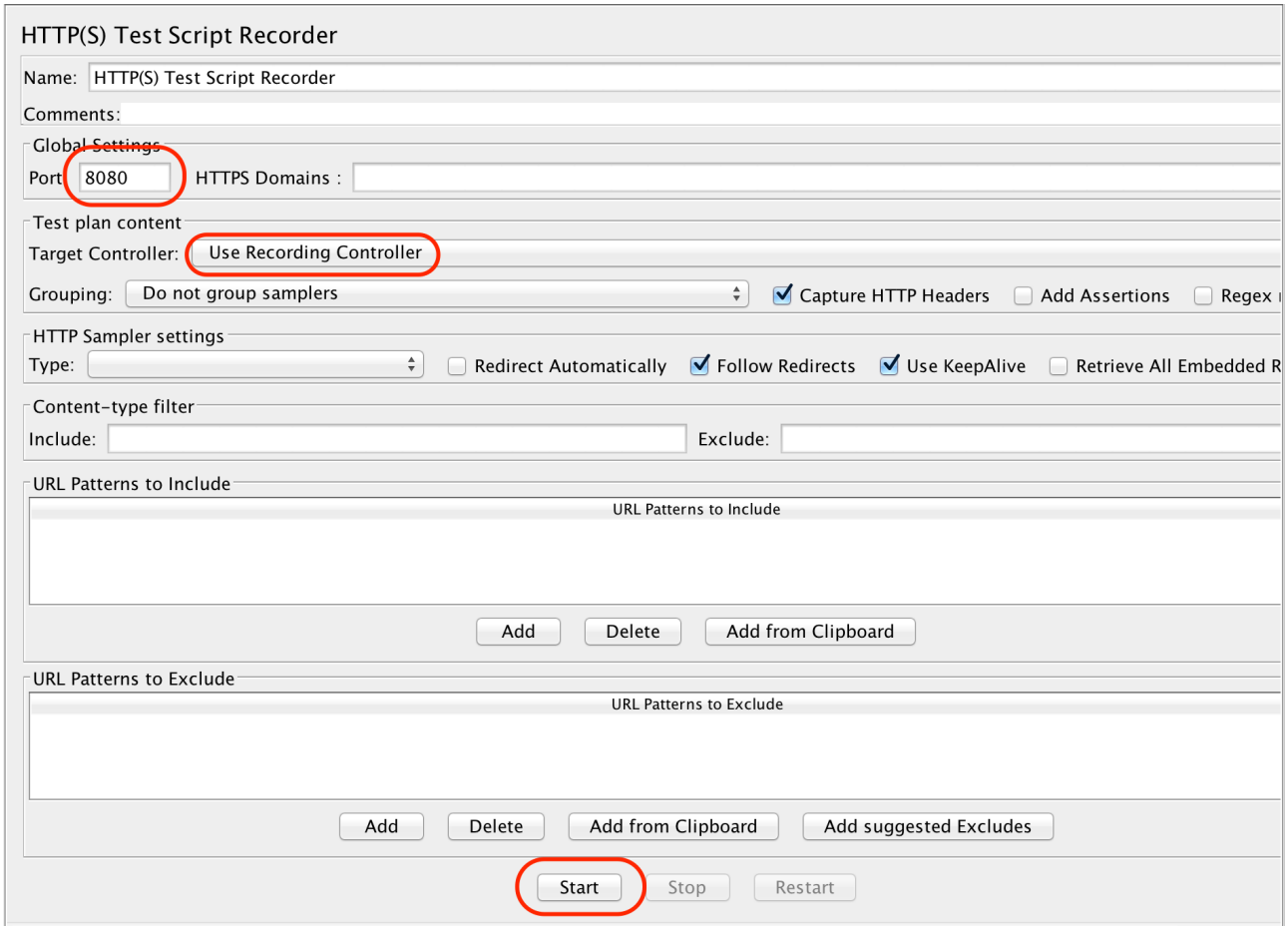
Other proxy tools are available; for the time being though, we’ll stick with the JMeter Proxy. Oh, and the browser plugin FoxyProxy, which will help us send our requests to the right place.

First let’s go into JMeter and setup the proxy by right-clicking on the Workbench icon and selecting Add > Non-Test Elements > HTTP(S) Test Script Recorder.

While you’re there, right-click on the Workbench icon again and add a Recording Controller via Add > Logic Controller > Recording Controller.



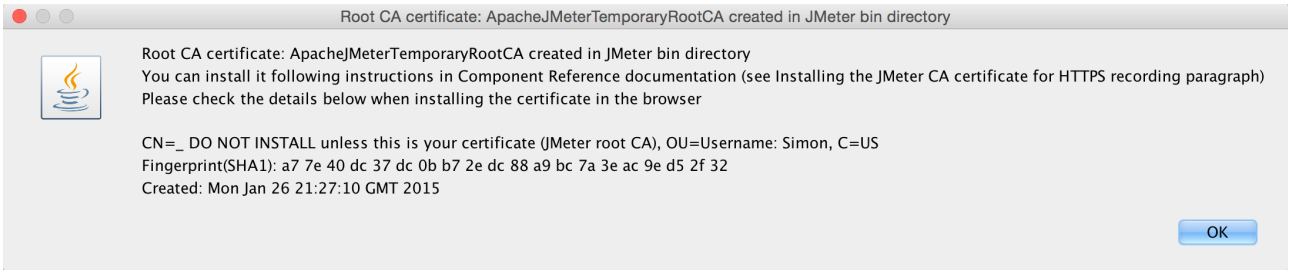
Now go into the proxy (HTTP(S) Test Script Recorder), which will look like the below. I’ve highlighted the areas that are important to us right now:



Port is the port number we should direct the browser to if we want the JMeter proxy to capture the requests.

The Target Controller is where we want JMeter to send the requests to, once they've been captured. It's set to Use Recording Controller by default, and we've added a Recording Controller, so I recommend leaving this alone.

Everything else can be ignored for the time being. Just hit the Start button. When you do, you'll probably see the message below (unless you're using JMeter 2.9 or earlier):

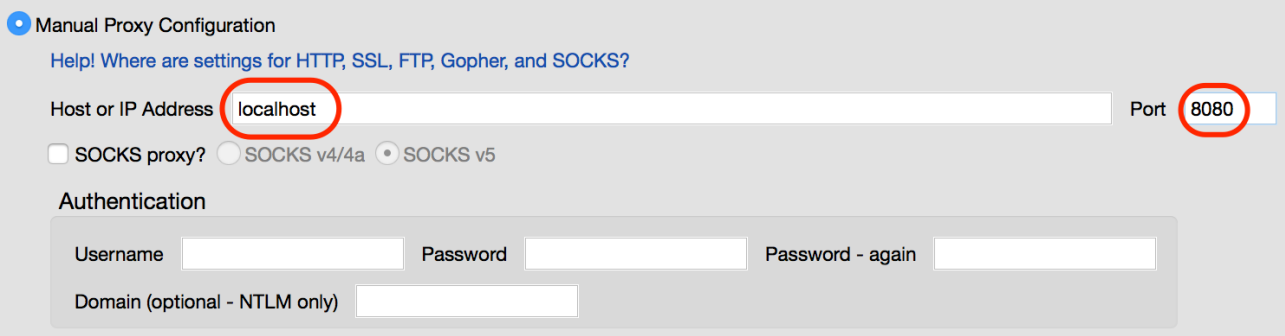


JMeter is telling us that it has generated a certificate to enable interception of HTTPS traffic from the web server. When we get around to trying to simulate a login carried out over HTTPS, you'll need to accept the JMeter generated certificate in your browser in order to proceed. But we're not quite there yet.

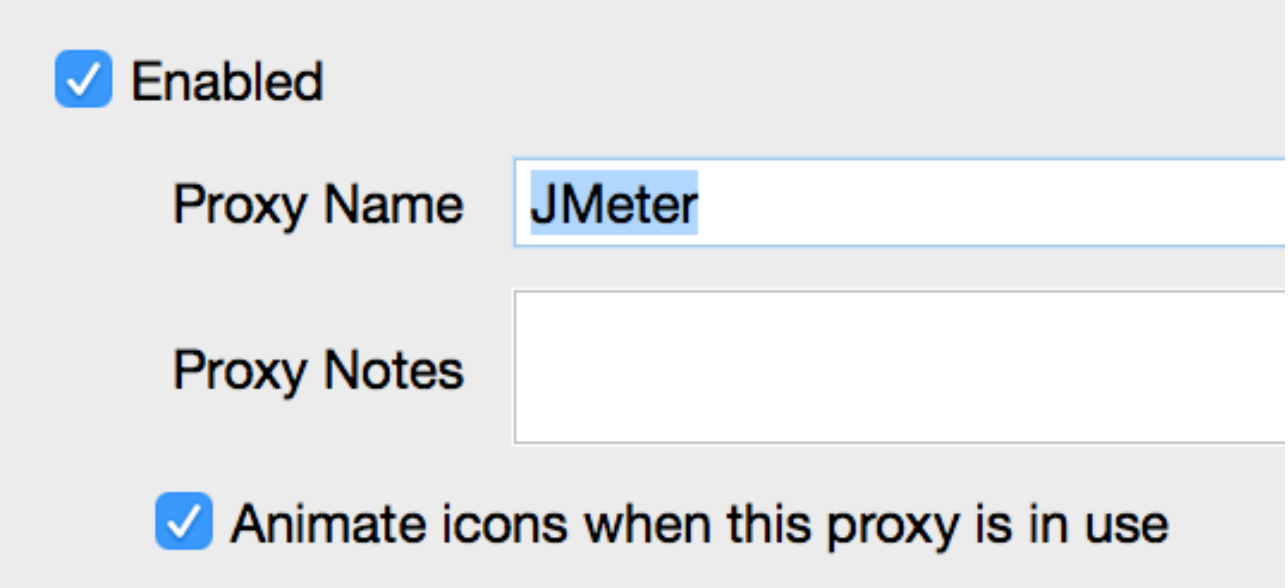
Before we can start recording you need to install Fox Proxy. Actually, you don't strictly need to do this - you could change the proxy manually and then change it back again once you're done; but using FoxyProxy makes life slightly easier.

Open up your preferred browser (I'm assuming that's either Chrome or Firefox) and download FoxyProxy by navigating here and following the instructions for the browser you want to use it with.

Once you've done that (and re-started your browser to activate the plugin), create some new proxy settings in FoxyProxy by opening up the main FoxyProxy window and clicking Add New Proxy. Navigate to the Proxy Details tab, and you should see a form like the below. Enter "localhost" and the port number 8080 in the marked areas.

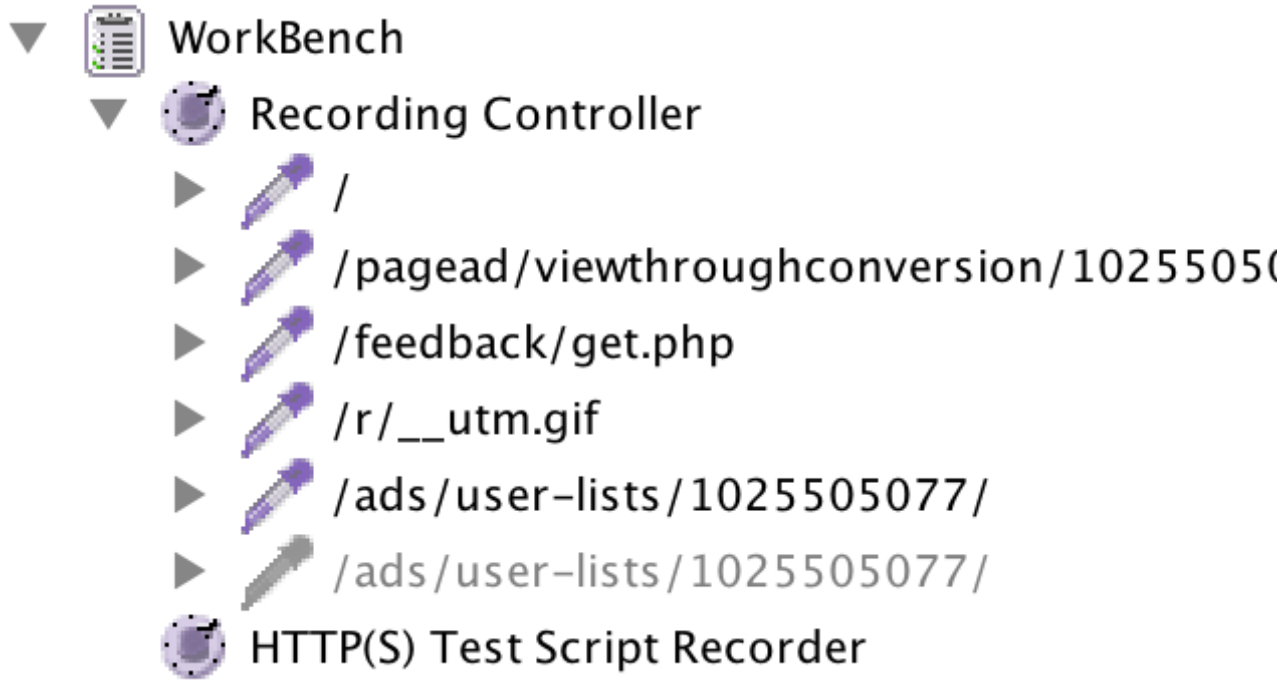


Click on the General tab and give your proxy a name. I imaginatively called mine JMeter, below.

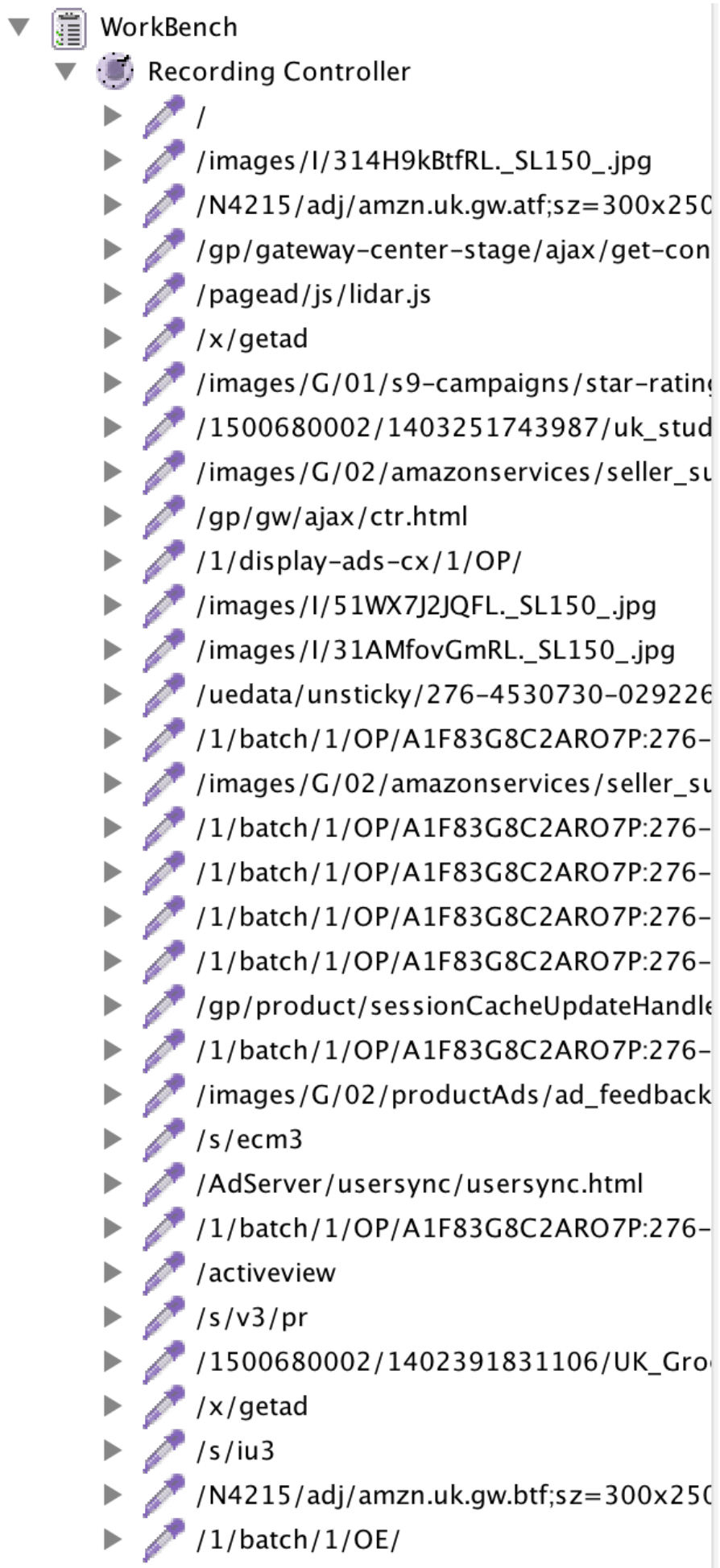


When you're done, click on OK, close the main window and then activate your proxy settings by right-clicking on the FoxyProxy button in your browser tool/address bar and selecting Use proxy "JMeter" for all URL's, where "JMeter" is whatever you decided to call your proxy settings.

Navigate to a web page in your browser then go back to the JMeter window. Under your Recording Controller you should see that JMeter has captured the requests your browser made when loading whatever web page it was you opened up. I used http://getfoxyproxy.org and saw the results below:



Your results may vary. Navigating to the amazon.co.uk home page produces a significantly different result:

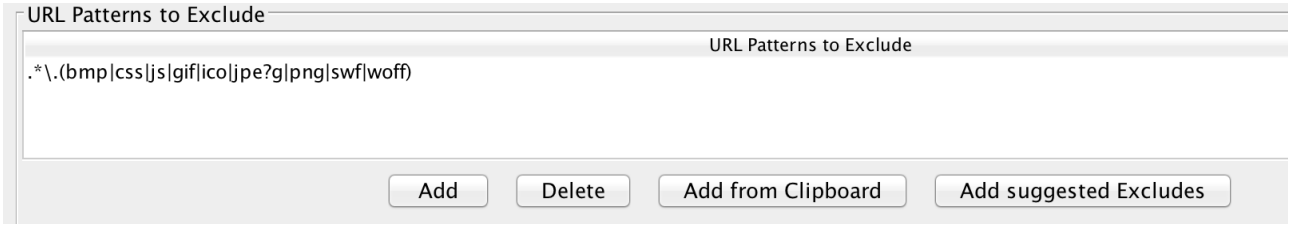

WorkBench
Recording Controller
/
/images/I/314H9kBtfRL._SL150_.jpg
/N4215/adj/amzn.uk.gw.atf;sz=300x250
/gp/gateway-center-stage/ajax/get-con
/pagead/js/lidar.js
/x/getad
/images/G/01/s9-campaigns/star-ratin
/1500680002/1403251743987/uk_stud
/images/G/02/amazonservices/seller_su
/gp/gw/ajax/ctr.html
/1/display-ads-cx/1/OP/
/images/I/51WX7J2JQFL._SL150_.jpg
/images/I/31AMfovGmRL._SL150_.jpg
/uedata/unsticky/276-4530730-029226
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/images/G/02/amazonservices/seller_su
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/gp/product/sessionCacheUpdateHandle
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/images/G/02/productAds/ad_feedback
/s/ecm3
/AdServer/usersync/usersync.html
/1/batch/1/OP/A1F83G8C2ARO7P:276-
/activeview
/s/v3/pr
/1500680002/1402391831106/UK_Gro
/x/getad
/s/iu3
/N4215/adj/amzn.uk.gw.btf;sz=300x250
/1/batch/1/OE/

There's a few different things going on here. I'm not going to go into detail, but a quick scan shows:

- / = the original page request
- /images/I/314H9kBtfRL._SL150_.jpg = an image from the amazon site
- /N4215/adj/amzn.uk.gw.atf;sz=300x250;oe=ISO-8859... = a call to a third party web application - Google Double-click in this instance
- /gp/gateway-center-stage/ajax/get-content.html = an ajax call for further content from the Amazon server

And so on and so forth.

Most of this stuff isn't relevant for our scripting efforts, though it will be when we come to actually analysing performance results. For the time being we can exclude it from our recordings by going back to the HTTP(S) Test Script Recorder and clicking on the Add Suggested Excludes button, which will result in something similar to the below.



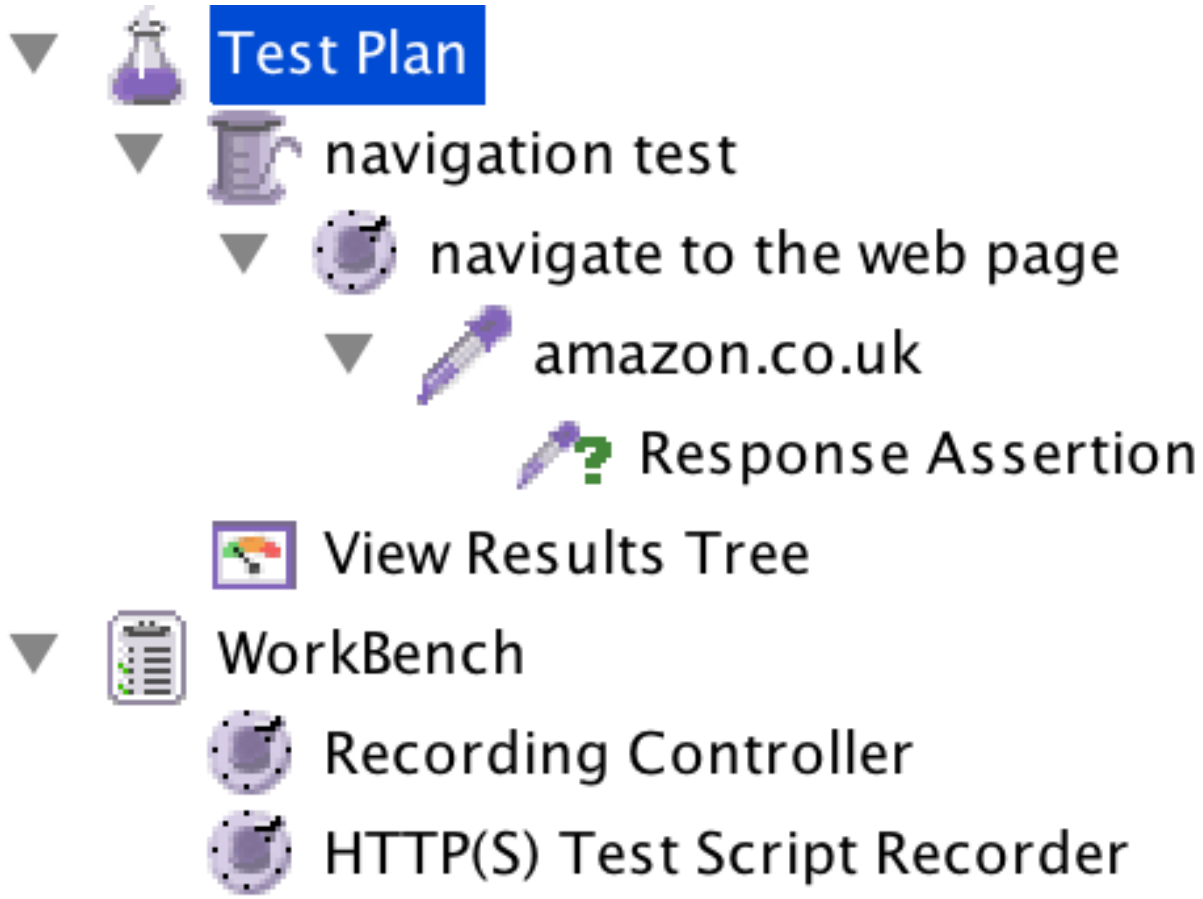
Re-recording our amazon.co.uk request will generate a significantly reduced result set, probably similar to the below:

Now we're ready to start recording our login script.

Recording a Login Script

Following on from my last couple of posts where I covered initial setup and use of the HTTP(S) Test Script Recorder, I'm going to build on what's been done so far in order to develop a login script.

I'm assuming that you have a test plan setup more or less identical to the below, in which case we're good to go. If you don't - then I suggest you read through the preceding posts before continuing further.

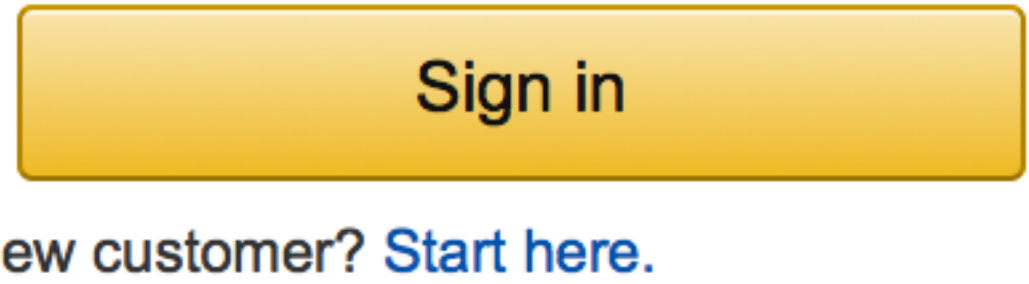


The thing we need to do next is figure out what requests need to be sent in order to simulate a user login. We could try and craft them from scratch, but it's easier to simply record what happens when I carry out a login and then modify the recorded requests to make them resilient and reusable.

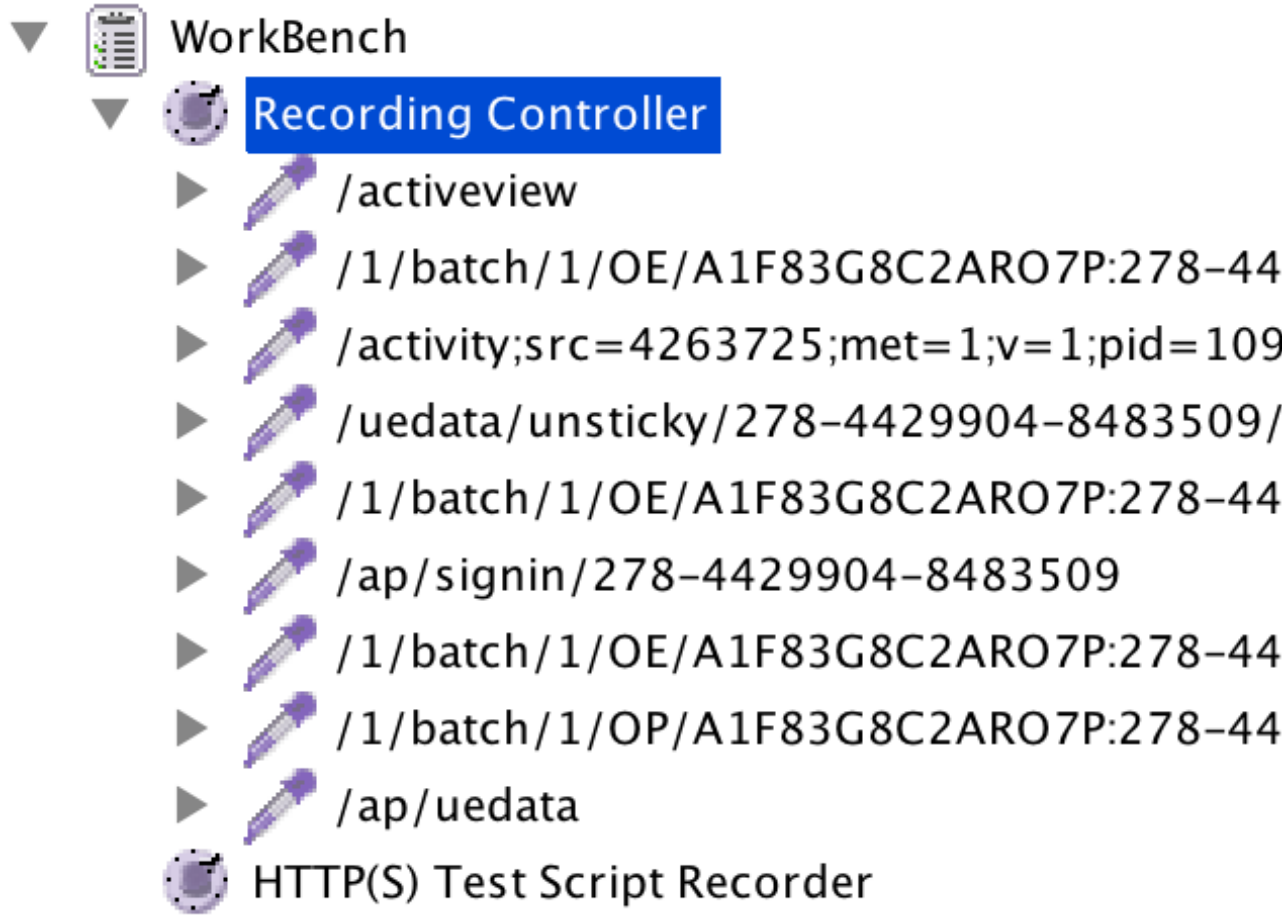
To do that - we first need to make sure that:

1. The JMeter proxy is recording our requests
2. The browser being used is directing traffic to the proxy

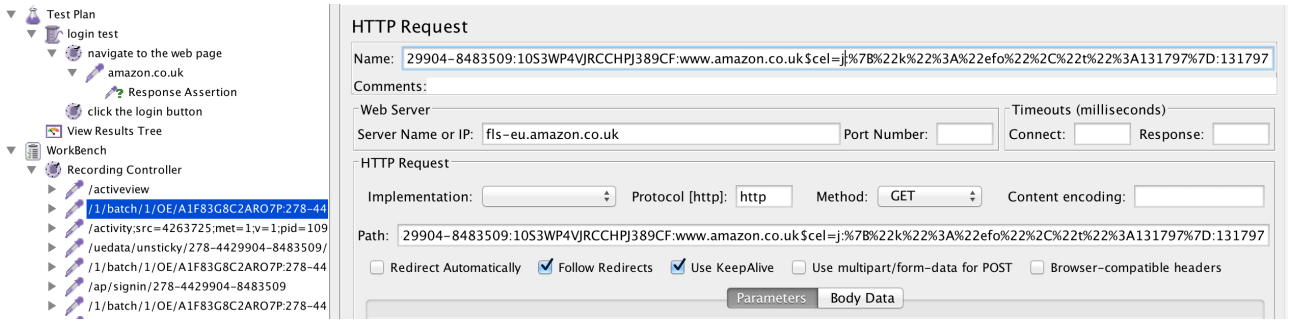
Again if you're not sure how to do steps 1 & 2 above, I refer you to the previous post. Assuming that your proxy is recording properly, then we can go ahead and click on the login button and see what happens:



Once you've done so then, under the Recording Controller you should see some activity. When I wrote this, I saw the responses below:

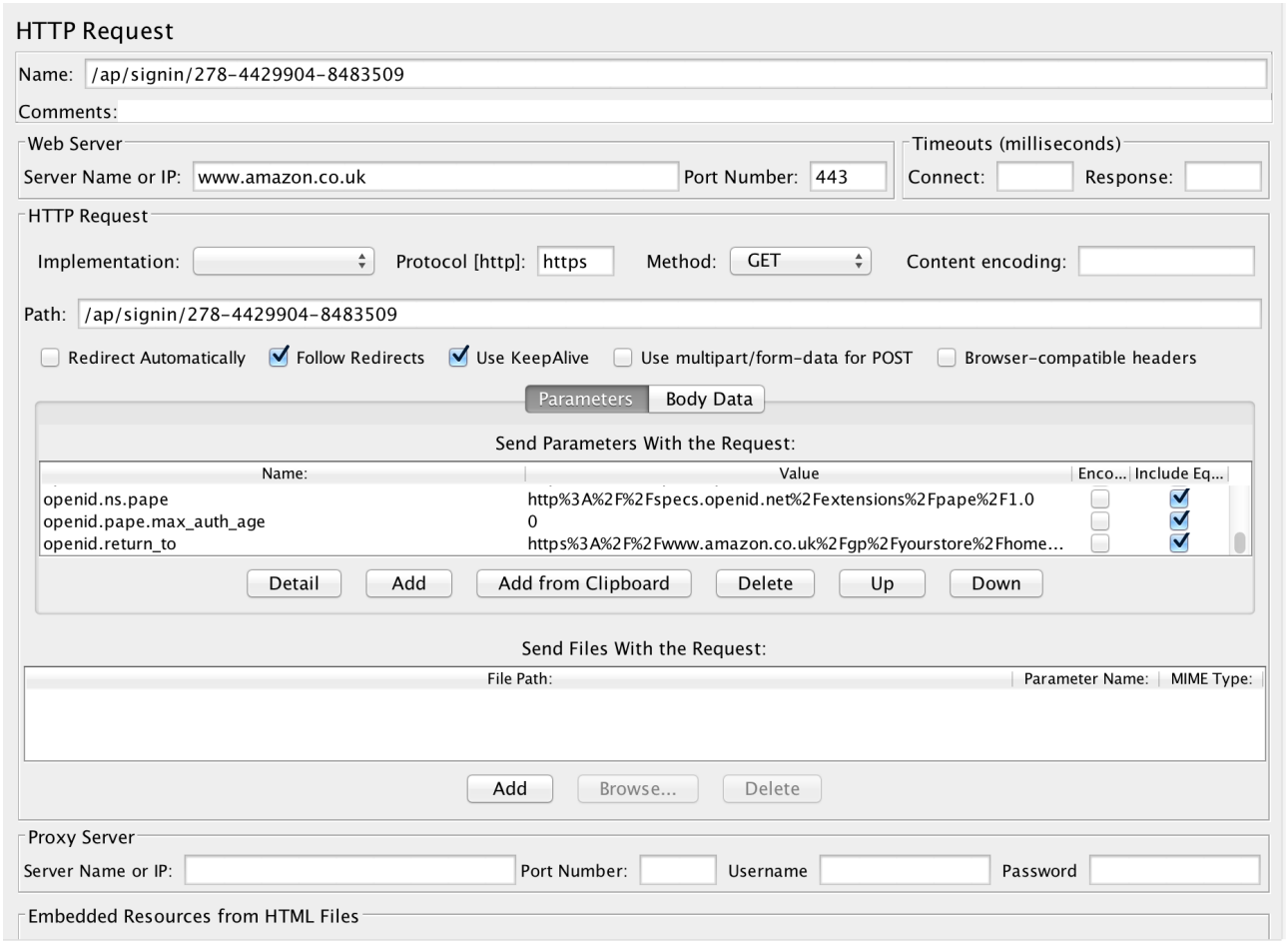


Further examination of some of the recorded requests suggests they're irrelevant to the task at hand. I have no idea what the request below is doing for example:



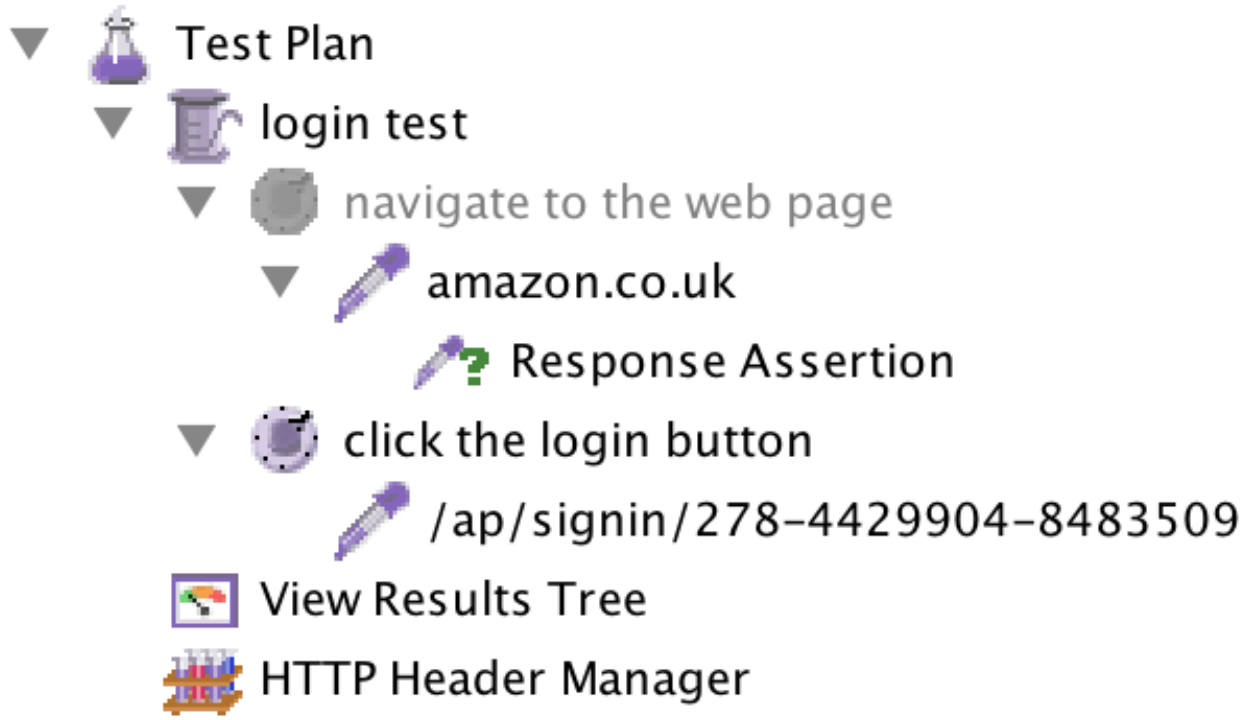
If I actually worked at Amazon I could probably go and ask one of the developers what's happening here, but for the time being I'm going to assume I don't need it.

Of more interest to me is the /ap/signin/… request:



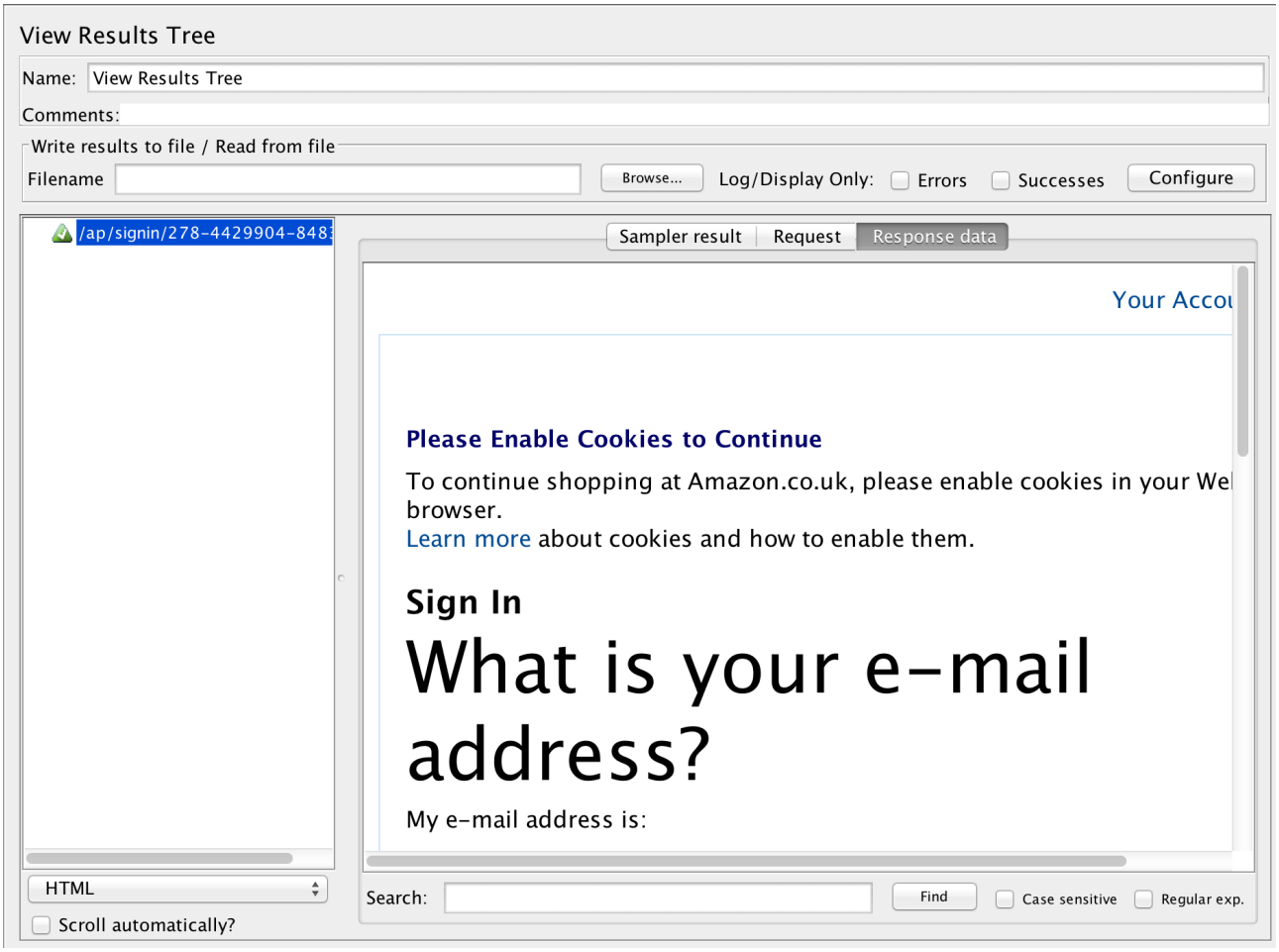
I'm intrigued by the /ap/uedata/ request also, but I'll ignore that one too for the time being. In the meantime, I'm working with the hypothesis that the /ap/signin/… request is the one that actually requests the login page from the server.

I've added another Simple Controller under my Thread Group. In order to test out my hypothesis I can move that request to the Controller and run it to see what happens. I'll disable the homepage navigation (right-click on the Controller > Disable) as well, since we don't need that right now. Disabling it will prevent any child requests from being executed.

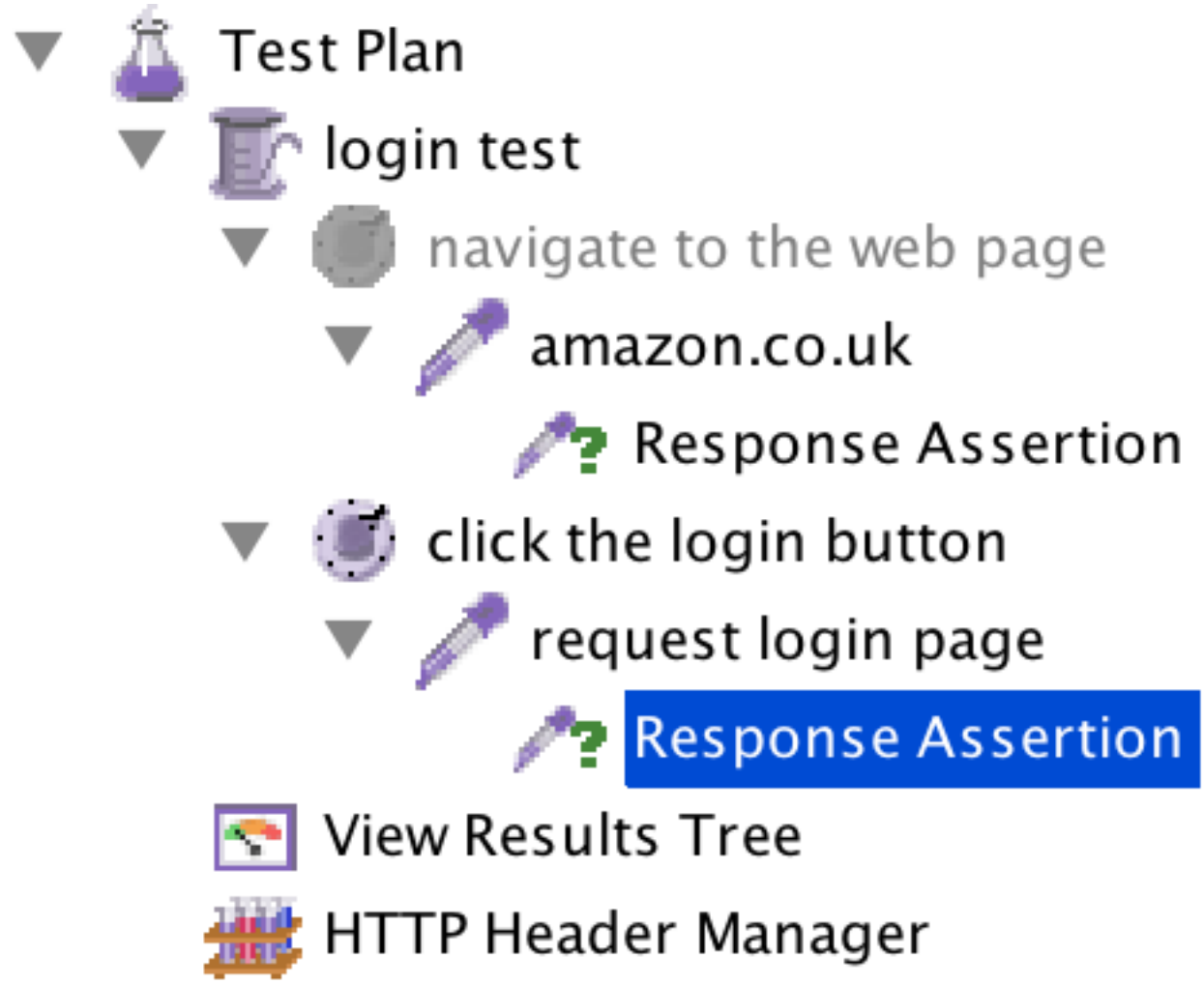


I've also moved the HTTP Header Manager that was originally under the HTTP request, and placed it under the Test Plan instead. Every recorded request will be paired with one of these, but we only need one of them to act as a default for the entire test. All subsequent header managers can be discarded, otherwise they'll just clutter things up.

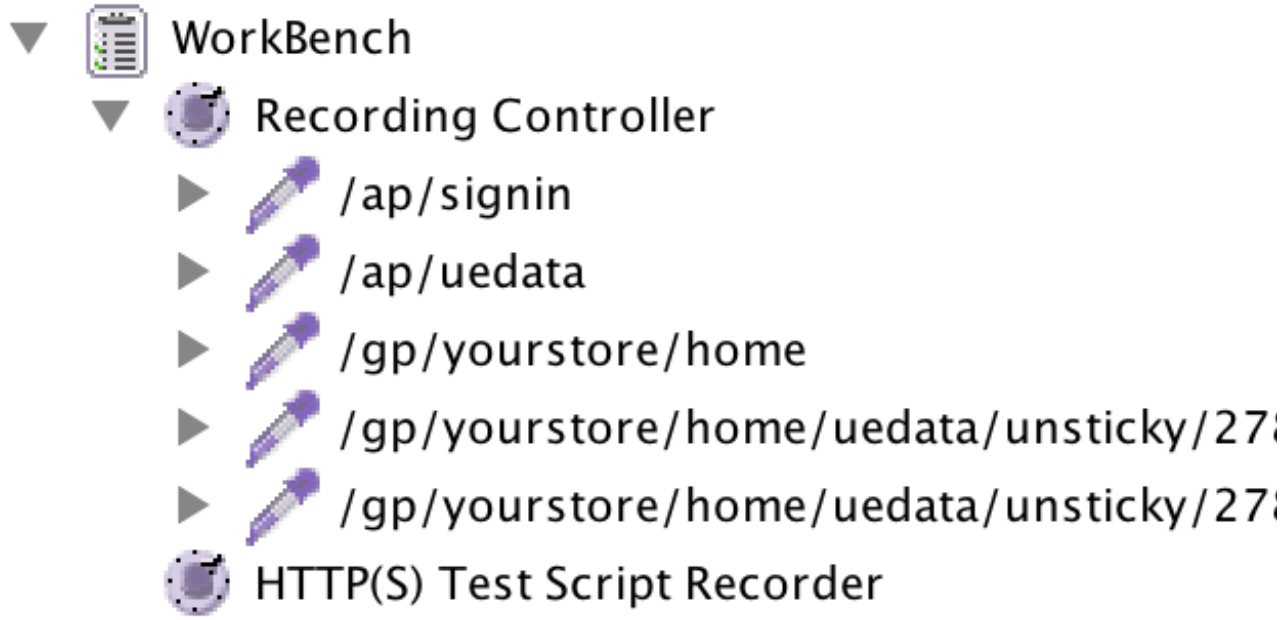
Running the test confirms my hypothesis, since I observe the following result:



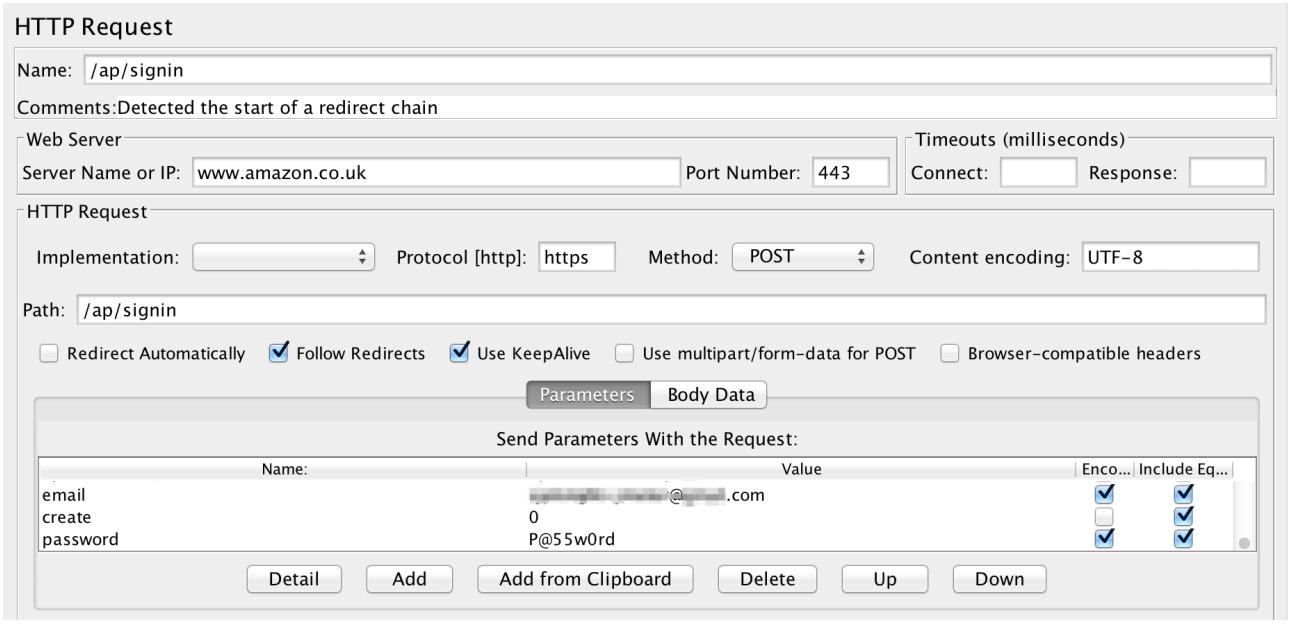
I'll rename the request to something more meaningful and add a Response Assertion to check for some text ("What is your e-mail address?"), so we know the request is doing the right thing going forward.



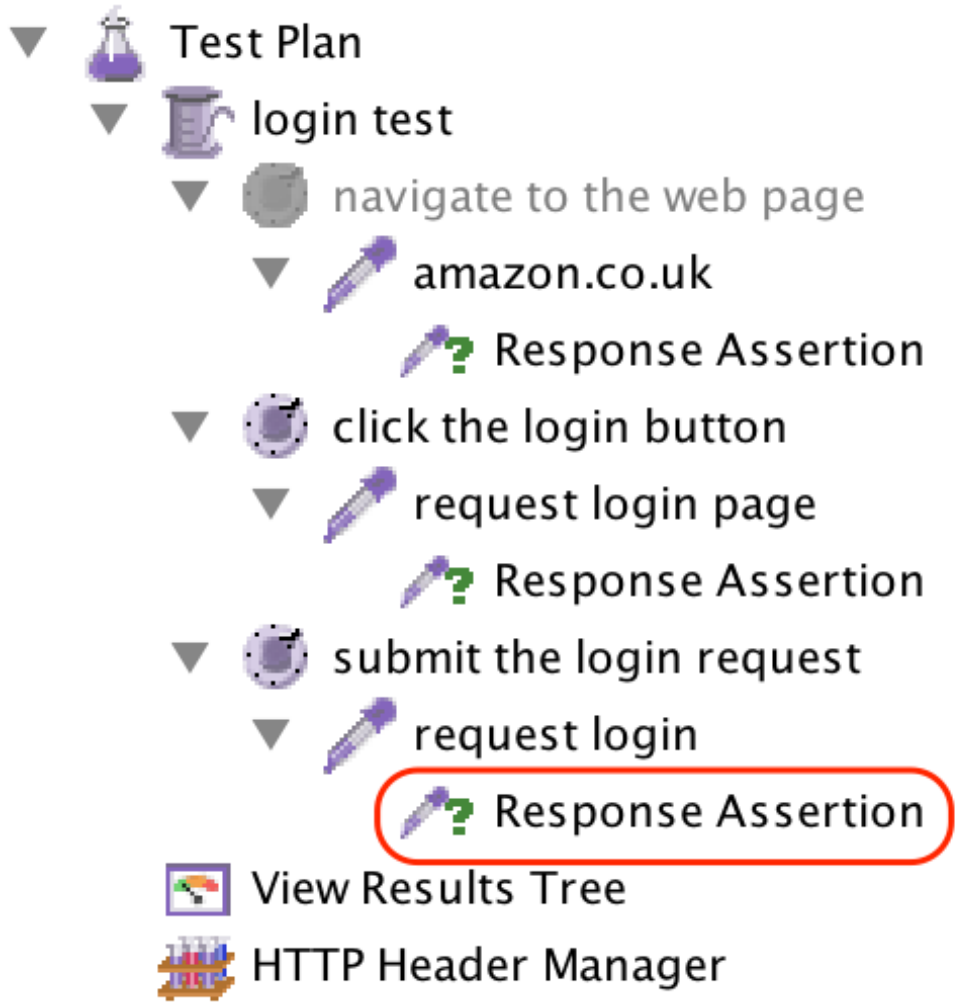
Now we need to submit some actual login details. Again, the best thing will be to submit an actual login and see what the proxy recorder tells us about the process. I did that while writing this paper and saw the responses below:



The /ap/signin request seems like it's probably the one doing the work, and closer examination shows that the request contains the username and password I used to login with.



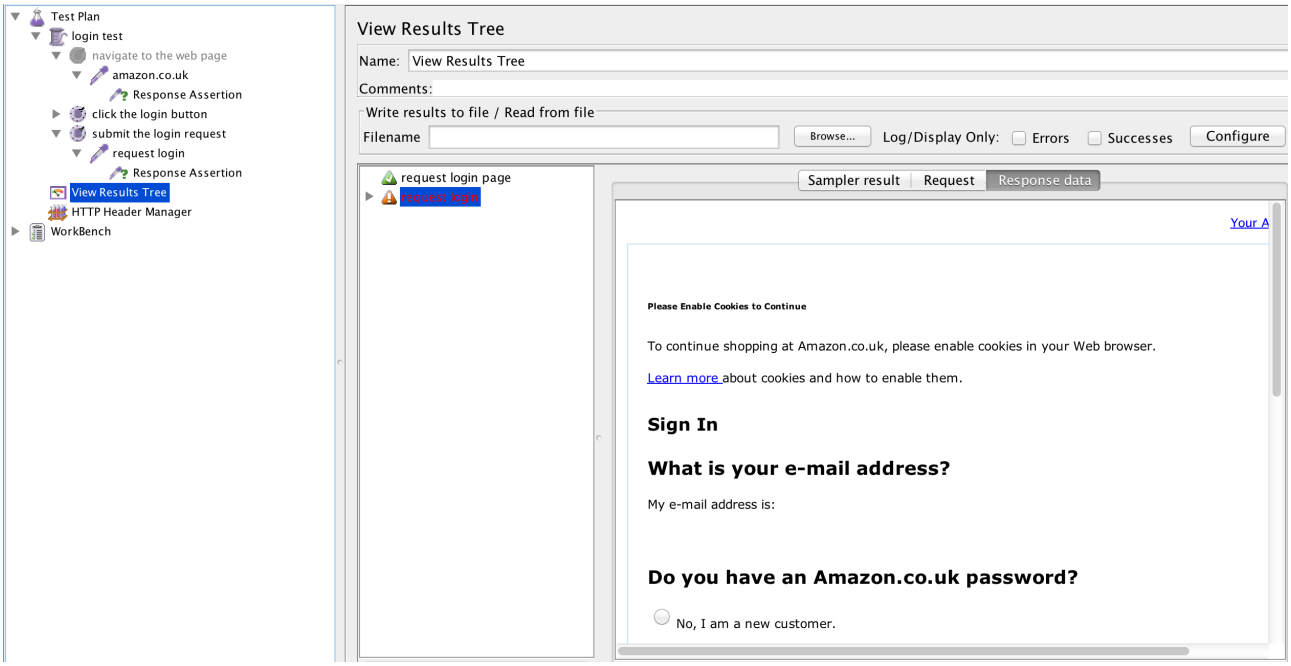
We can move that request up into our Test Plan as well, so it looks like the below, adding an assertion to check we're actually logged in after making the request also. Let's test for some specific HTML that we would only expect to see when we're logged into the site. Something like "Your Browsing History" ought to do it.



All good. We can go ahead and run that now, right?

Wrong.

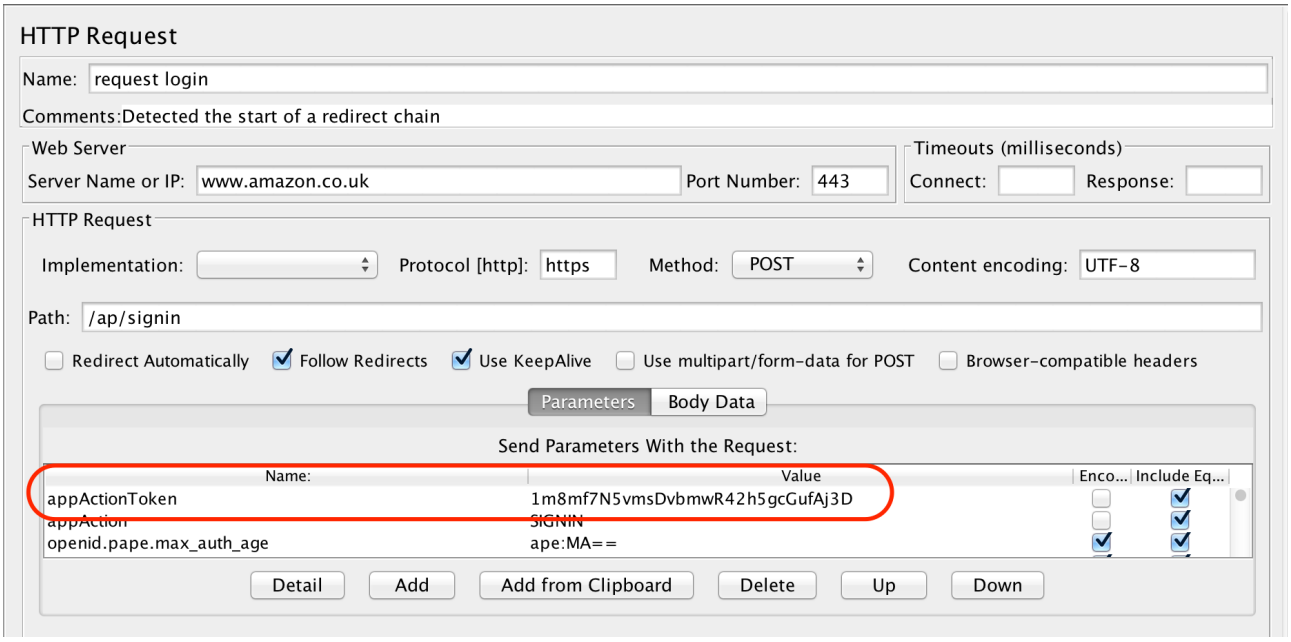
There's a problem. Can you tell what it is yet?



The assertion has failed because JMeter didn't get the page back that it expected. In fact, judging by the page we did get - it doesn't look as though the login has worked at all.

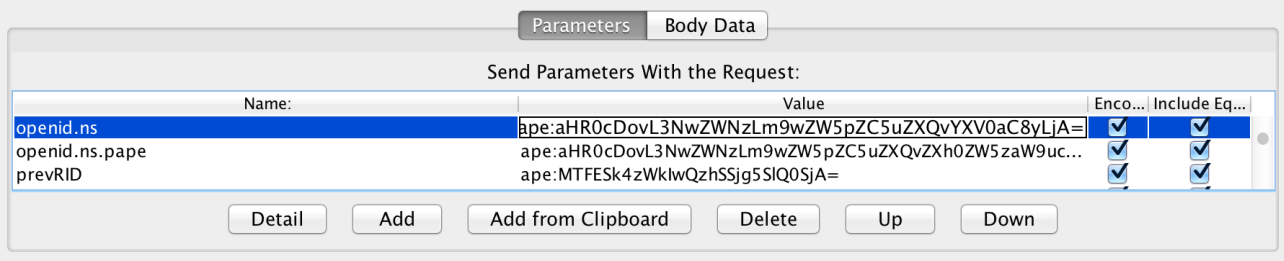
Why not?

If I take a look at the request again, there's a few clues as to why...



The request is sending a token along with the (currently hard-coded) login credentials.

It's sending a bunch of other stuff over as well by the looks of it.

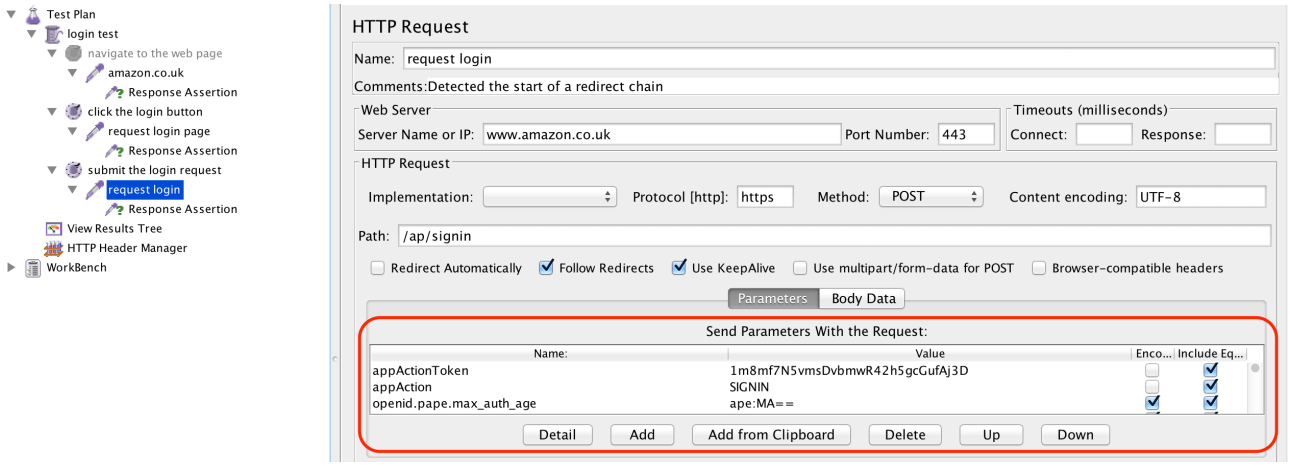


In all, there's actually 11 dynamic variables that need to be correlated across from the previous server response in order for the login to be considered a valid request by the server.

It's going to take a bit of effort to get all that sorted out… I'll show you how it's done next.

Correlating Dynamic Variables in JMeter

In the previous section, the login script wasn't working yet. The actual login request (i.e. the submission of the login credentials as a request to the server to initiate a logged-in session) was failing because we weren't providing it with all of the information it needed.

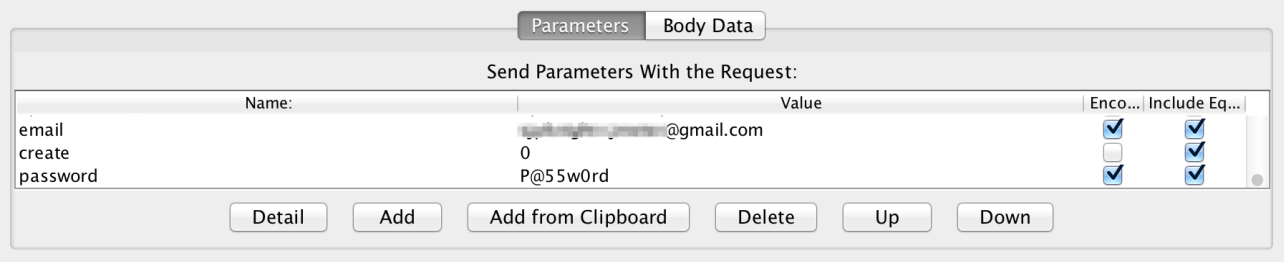


In the Send Parameters With the Request section of the HTTP Request sampler, request login, above, we can see that there's an AppActionToken that looks as though it's been generated by the server, probably to uniquely identify the session. If we continue to scroll down the list of parameters, we'd see that there are a number of other tokens that are required in order to successfully login:

1. appActionToken = 1m8mf7N5vmsDvbmwR42h5gcGufAj3D
2. openid.pape.max_auth_age = ape:MA==
3. openid.ns = ape:aHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvYXV0aC8yLjA=
4. openid.ns.pape
   = ape:aHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvZXh0ZW5zaW9ucy9wYXBlLzEuMA==
5. prevRID = ape:MTFESk4zWkIwQzhSSjg5SlQ0SjA=
6. pageId = ape:Z2JmbGV4

7. openid.identity
   = ape:aHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvYXV0aC8yLjAvaWRlbnRpZmllcl9zZWxlY3Q=
8. openid.claimed_id
   = ape:aHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvYXV0aC8yLjAvaWRlbnRpZmllcl9zZWxlY3Q=
9. openid.mode = ape:Y2hlY2tpZF9zZXR1cA==
10. openid.assoc_handle = ape:Z2JmbGV4
11. openid.return_to
    = ape:aHR0cHM6Ly93d3cuYW1hem9uLmNvLnVrL2dwL3lvdXJzdG9yZS9ob21lP2llPVVURjgmcmVmXz1uYXZfc2lnbmlu

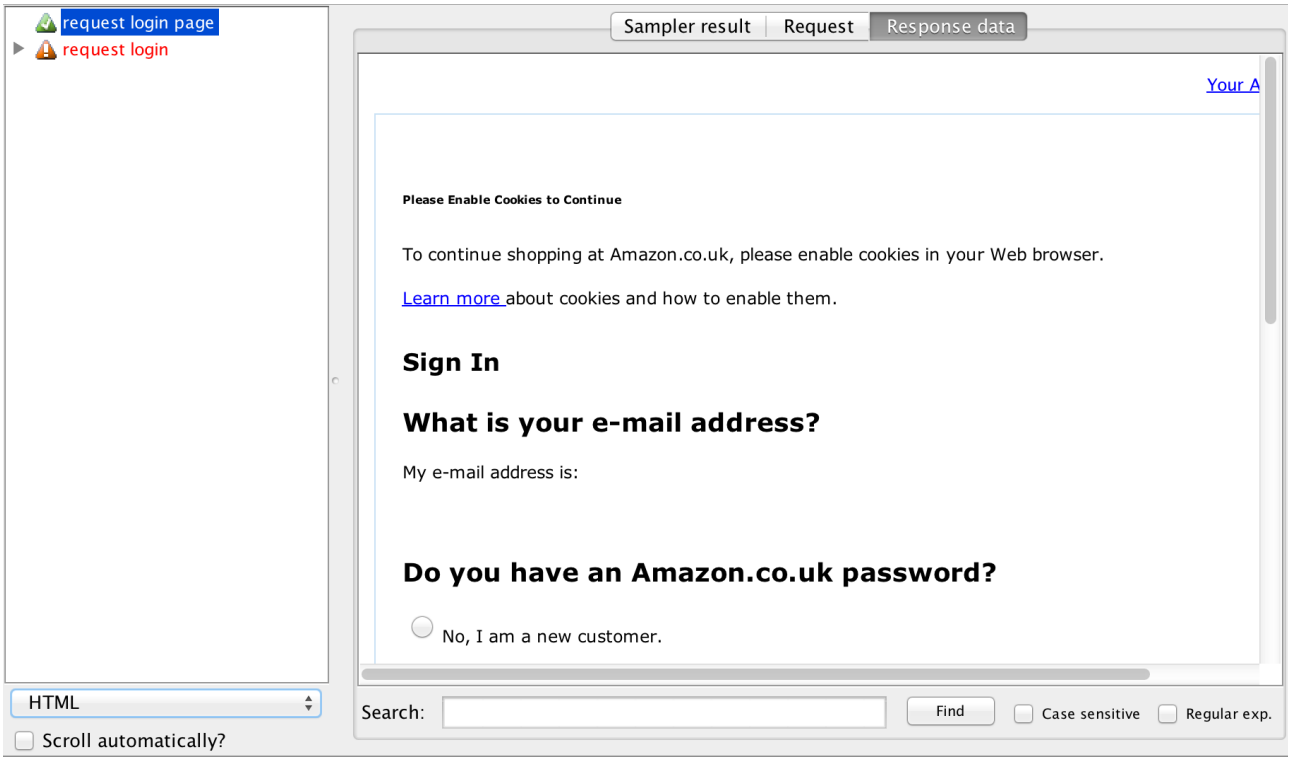
In addition to the dynamic parameters, we also need to submit the username and password, which are currently hardcoded as can be seen below.



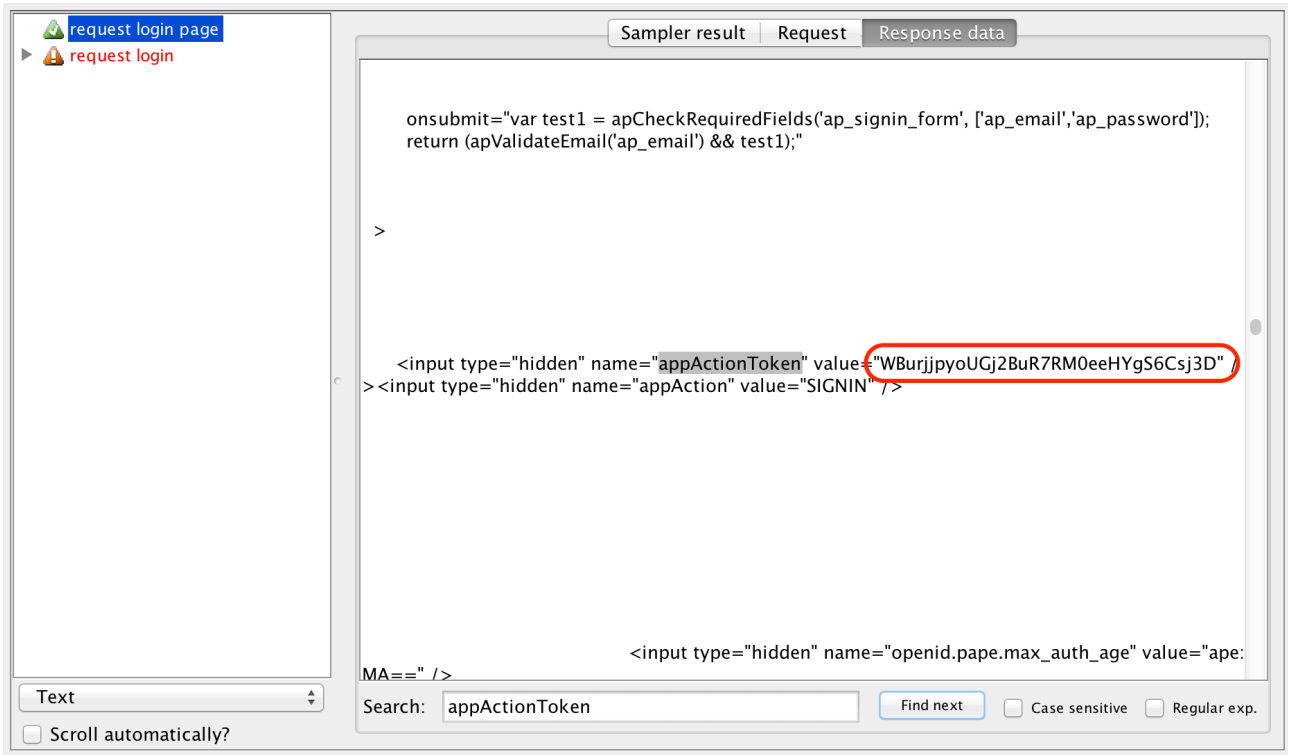
We could parameterise these also, but I'll talk about that another time.

So how do we go about obtaining the correct values for these parameters, such that when we send them all, along with some valid login credentials, we get a logged-in session back from the server?

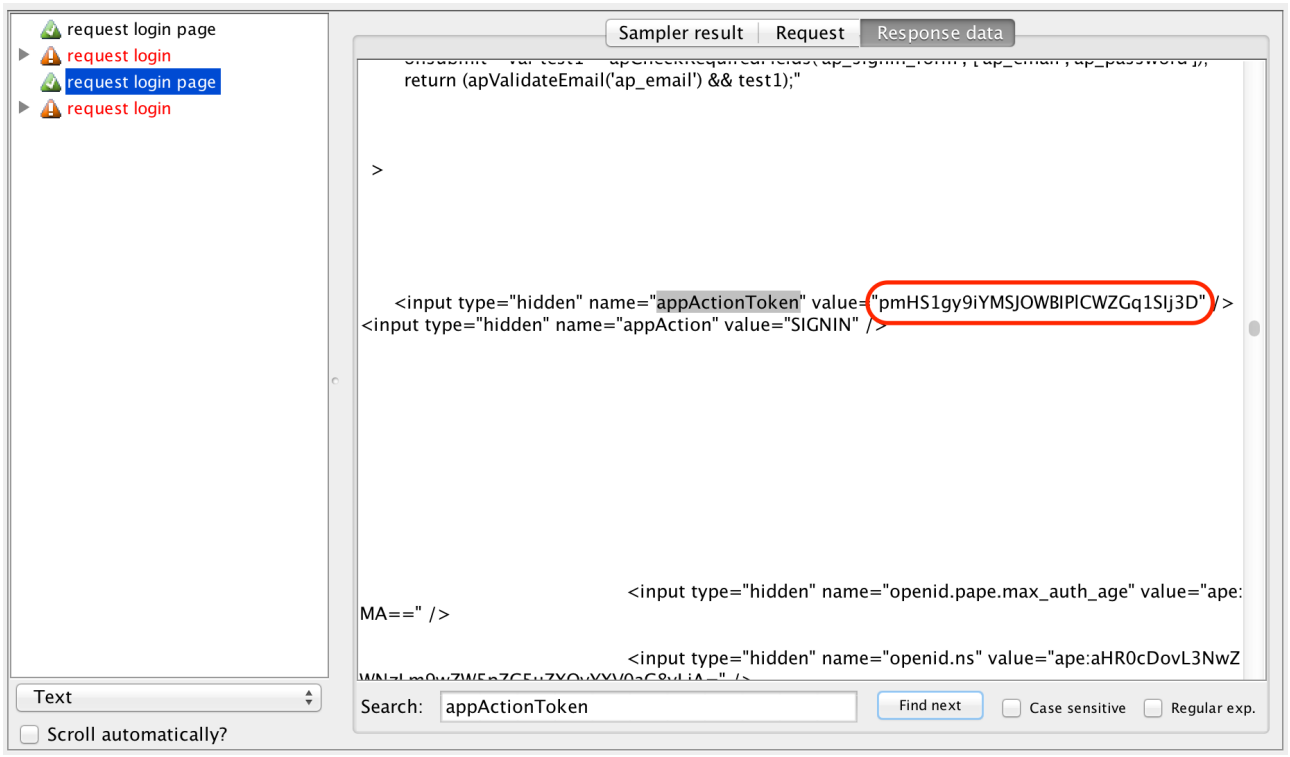
Well, the first thing is to figure out from whence they came. Experience has taught me that it's usually (though by no means always) from the immediately preceding server response. So let's go back to the results of our test and take a look:



The HTML response isn't telling us much… We need to switch to text mode, and then take a look for the parameter name. Let's start with the appActionToken parameter:



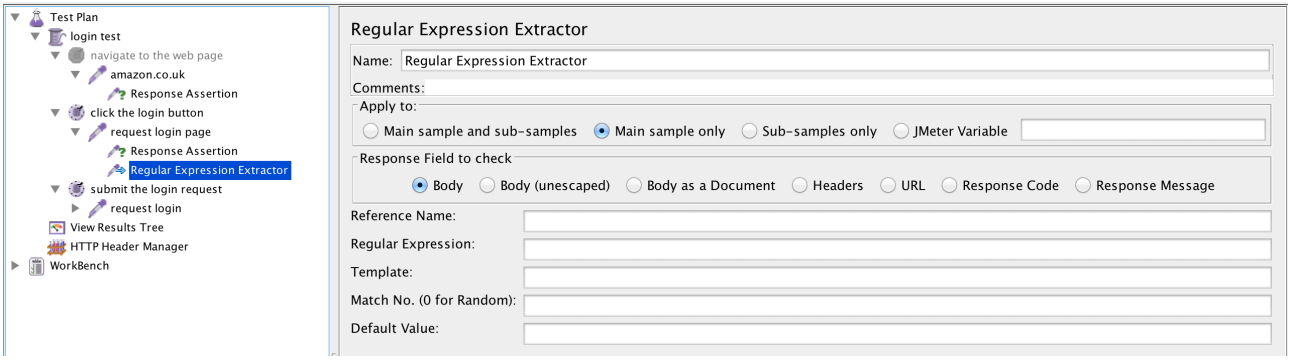
Voila! We've found the token, and the value. But if we run the test again, we'll probably see a different one:



It's a fair (in fact guaranteed) bet that we'll find the rest of our parameters embedded within this response too.

So what we need to try and do is extract the parameters from the server response, each time we get one - and then pass it into the next request. Performance testers call this process correlation. JMeter provides us with the Regular Expression Extractor so that we can go ahead and correlate our parameters from one request/response to another.

The first step is to add a Regular Expression Extractor to the request login page HTTP Sampler, by right-clicking on it and then selecting Add > Post Processors > Regular Expression Extractor, thusly:



Next we need to write some Regex with which to extract the parameter.

If, like me, the idea of writing regex makes your toes curl with horror, don't worry. I'll share a special piece of JMeter goodness with you. It's the only piece of regex I've ever really needed to know. And it goes like this:

(.+?)

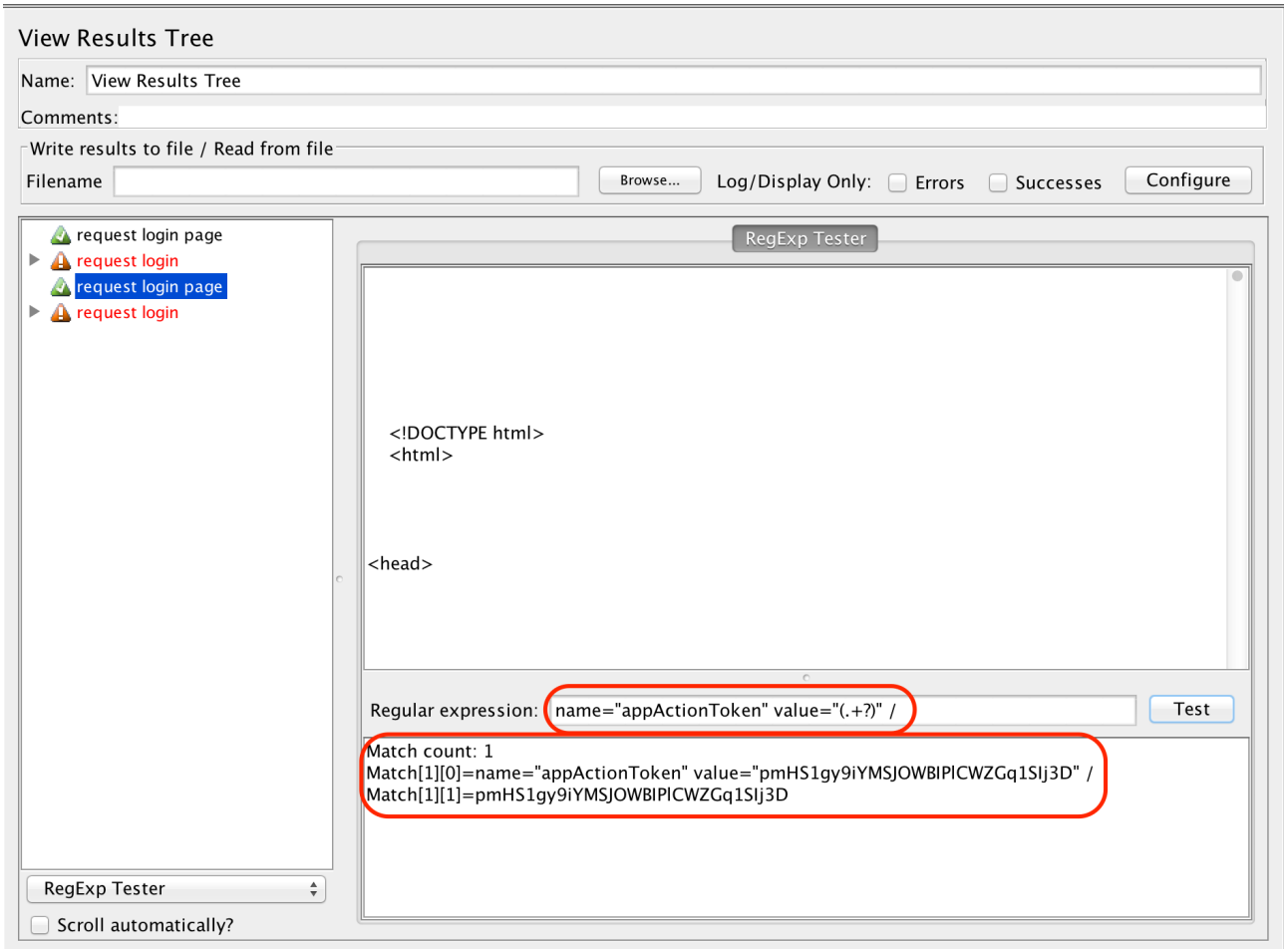
Did you get that? I'll repeat it just in case you missed it...

(.+?)

To use this regex and get the parameter we're looking for, I reckon something like the string below should work:

name="appActionToken" value="(.+?)" /

Trying it out in the RegExp Tester view of the response, shows that it will indeed work, since the test came back with only one match:



The RegExp Tester shows me how many matches the regex pattern will create, what the match is, and the value to be extracted - 1, 2 & 3 below respectively:

1. Match count: 1
2. Match[1][0]=name="appActionToken" value="pmHS1gy9iYMSJOWBIPlCWZGq1SIj3D" /
3. Match[1][1]=pmHS1gy9iYMSJOWBIPlCWZGq1SIj3D

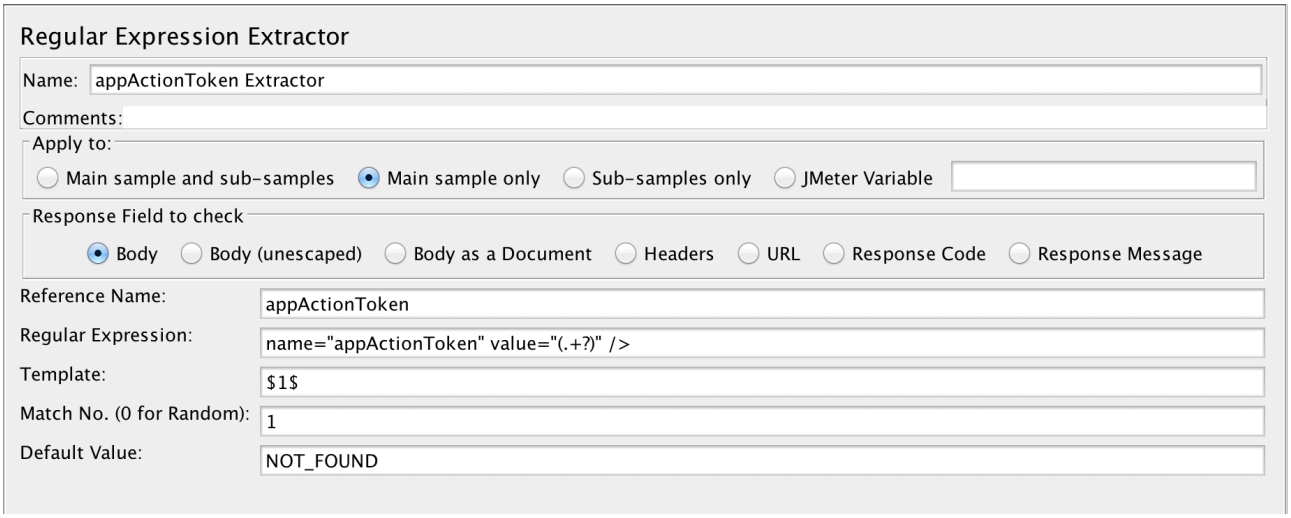
What I've done here is, take the HTML response, and apply the simple bit of regex I described above, in order to capture only the string that we're interested in.

<input type="hidden" name="appActionToken" value="pmHS1gy9iYMSJOWBIPlCWZGq1SIj3D" /><input type="hidden" name="appAction" value="SIGNIN" />

The “pmHS1gy9iYMSJOWBIPlCWZGq1SIj3D” bit, basically.

I’m not going to go into how the regex works, because that’s beyond the scope of this paper. What we do need to do now is plug it into the regex extractor so that we can use it in the request login sampler.

Here’s how the finished extractor looks:



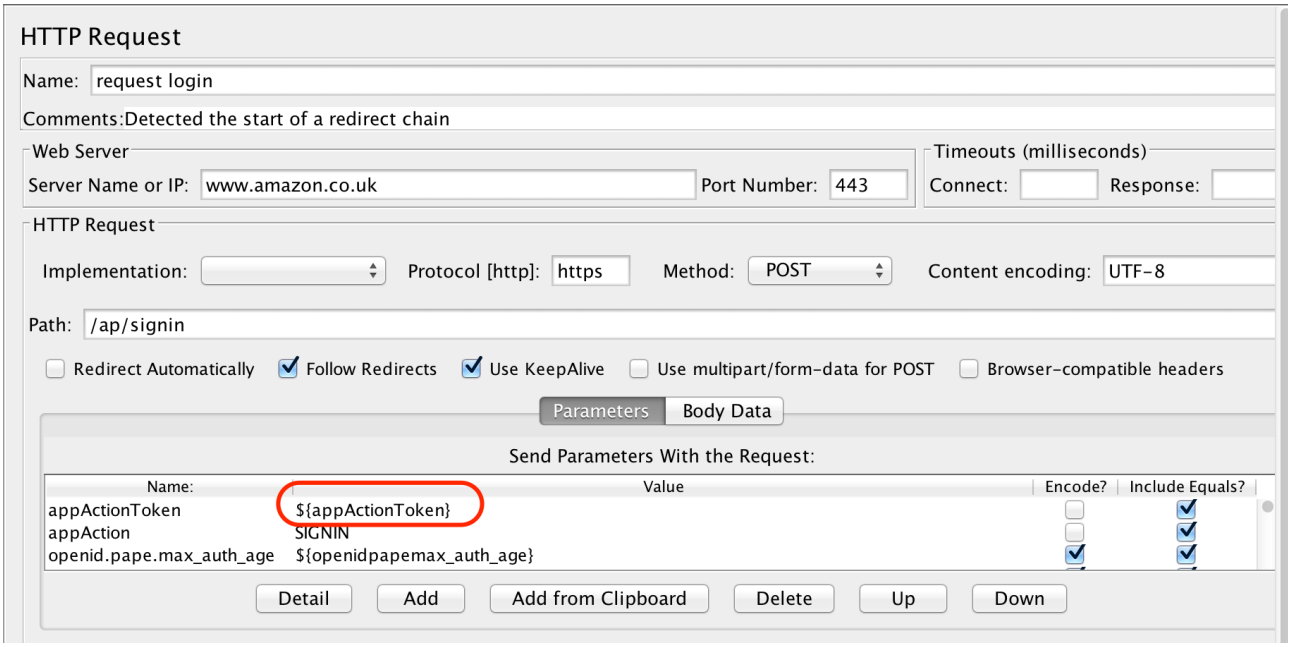
The important things to note are these:

- I’ve given the Regular Expression Extractor a meaningful name - appActionToken Extractor
- I’ve given the variable into which I want to put the extracted token a meaningful name (in the Reference Name field) - appActionToken
- The regular Expression field contains our regex - name="appActionToken" value="(.+?)" /
- The Template, Match No and Default Value fields are filled out more or less as per the Apache JMeter guide.

You can learn more about the Template, Match No and Default Value fields by reading through the online guide from Apache here. I’d recommend sticking with the defaults, but you may gain some mileage in experimenting with them.

Having extracted the value successfully and placed it in a JMeter variable, we now need to use the variable in the submit login sampler. We can do that by referencing the variable where the token was originally.

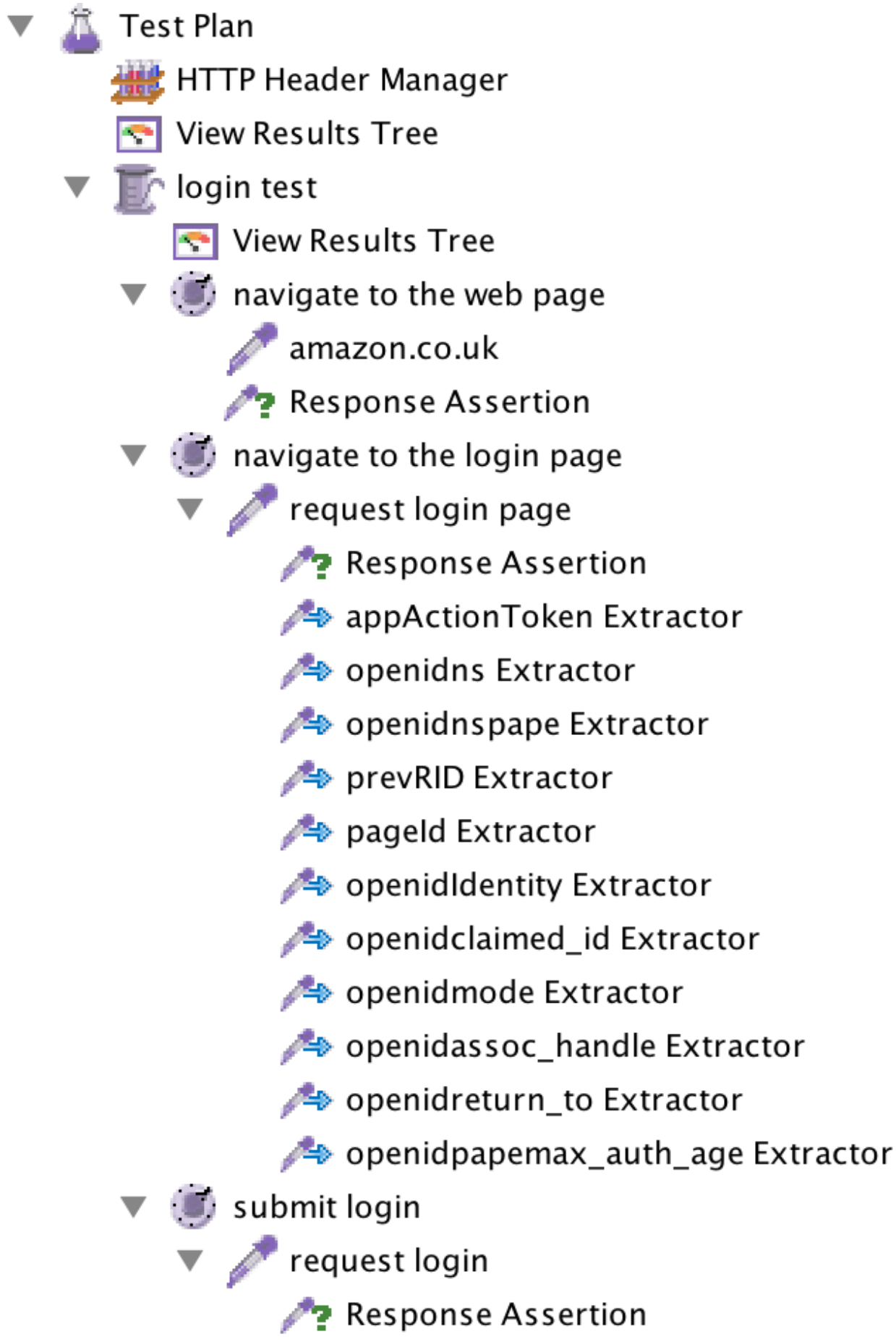
JMeter knows we want to use a variable when we place the name inside curly parenthesis with a leading dollar sign - ${variableName}, like the below:



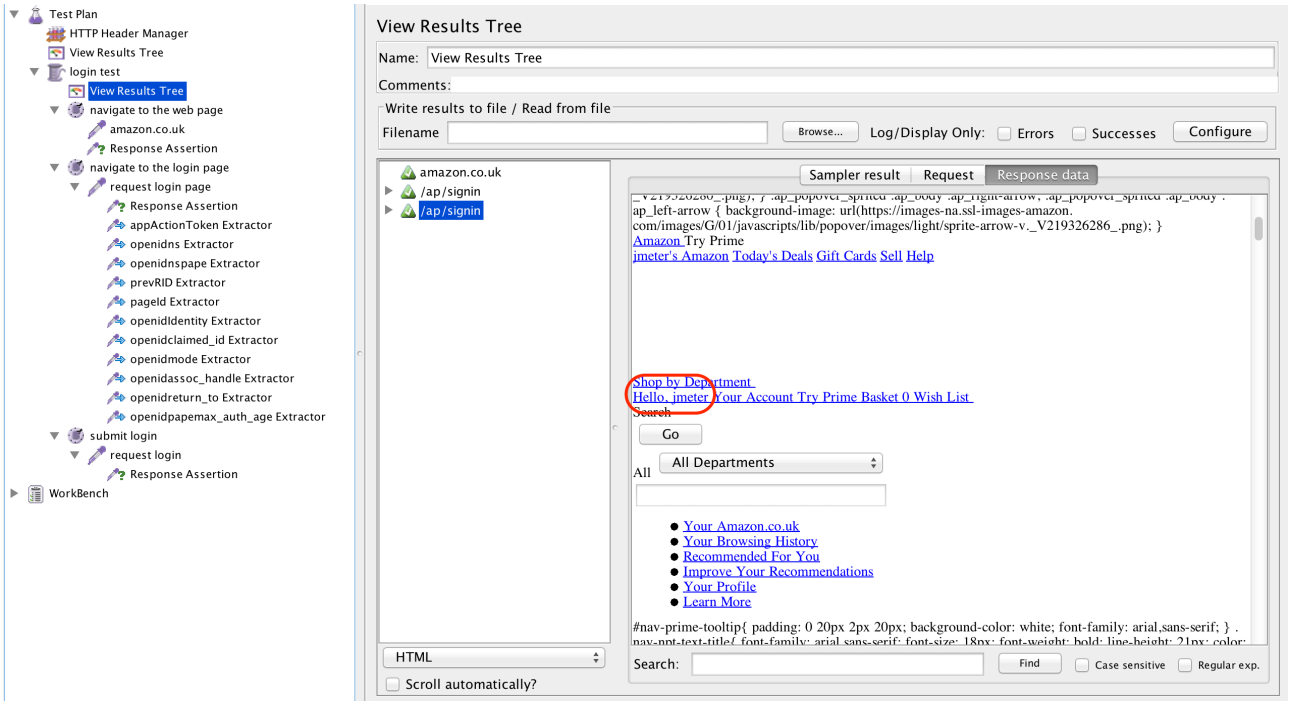
With that done, we're almost ready to go. Except, our script still won't work because there's those other 10 dynamic variables we need to correlate as well. Fortunately, the process is exactly the same for all of them:

1. Locate the variable in the preceding response - the request login page response in the case of our script.
2. Construct the regex to extract the variable - as discussed above.
3. Create a Regular Expression Extractor for each variable to be correlated.
4. Refer to the extracted variable in the subsequent request - the submit login page in our example, as discussed above.

Once you've gone ahead and done all of that, you'll likely end up with something that looks like the below:


Test Plan
HTTP Header Manager
View Results Tree
login test
View Results Tree
navigate to the web page
amazon.co.uk
Response Assertion
navigate to the login page
request login page
Response Assertion
appActionToken Extractor
openidns Extractor
openidnspape Extractor
prevRID Extractor
pageId Extractor
openidIdentity Extractor
openidclaimed_id Extractor
openidmode Extractor
openidassoc_handle Extractor
openidreturn_to Extractor
openidpapemax_auth_age Extractor
submit login
request login
Response Assertion

And, assuming that you've done everything correctly, running the test again will result in the desired logged-in response from the server:



We can see above the JMeter is logged in by the "Hello, jmeter" message in the response.

We're done! Our login script is now ready to rumble. And having covered the basics, we're ready to tackle some more advanced JMeter topics.

Make sure you keep coming back to my website, sjpknight.com for those.