**Scenario 5: How to read data from and write data into a text file**

Sometimes it may be required to read data from an external file or write data to an external file. Let’s see how we do this in a VuGen script.

Let’s first see how to read data from a text file.

**Example:** Say we have to read an XML from an external data file and use it as a request in the script.

To do this, we use **‘fread’** C function.

**This function takes four attributes:**

**buffer** – The buffer in which to store the data stream.  
**size** – The size of the buffer.  
**count** – The number of bytes to read.  
**file\_pointer** – The file pointer

**Example code (with comments) for the same**

**char** buffer[1000]; //The buffer to store the read data stream

**char** \*filename = "C:\\Temp\\mysamplefile.txt"; /\* name and path of the file to be read from \*/

**char** \* accessmode = "r"; /\* access mode r /r+ = open for reading , w /w+ = open for writing ,a /a+ = open for appending \*/

/\* "+" sign indicates that the file must already exist \*/

**long** filepointer; /\* declaring a file pointer \*/

**int** count=500; /\* number of bytes to be read \*/

filepointer = **fopen**(filename, accessmode); /\* open file in read mode \*/

**fread** (buffer,sizeof(char),count,filepointer); /\* read from output file \*/

lr\_save\_string(buffer,"requestbody"); //we can use this as a LR parmeter and can use in the script now

**fclose**(filepointer); //close the file pointer

Note that in the code shown above, we read 500 bytes from the text file. Obviously, we may not know the file size always. So we can use **‘fseek’** and **‘ftell’** C functions (I will leave this for you to explore) to find the size of the file and use the **‘count’**attribute of the **‘fread’** function accordingly.

Now let’s see how to write data to a text file.

**Example:** Say we have a script which creates orders and generates ‘orderid’. If we want to know all the order ids created by our script in a test, we can make our script copy these order ids to an external text file.

**‘fprintf’** – C function writes formatted output to a file.

**This function takes these attributes:**

**file\_pointer** – The file pointer

**format\_string** – The formatted string to write to the file.

**args** – One or more optional print arguments.

The example code is shown below. Let’s assume we have correlated and saved ‘orderid’ into the parameter ‘cOrderId’.

**char** \*filename = "C:\\Temp\\mysamplefile.txt"; /\* name and path of the file to be created or edited \*/

**char** \* accessmode = "a+"; /\* access mode r /r+ = open for reading , w /w+ = open for writing ,a /a+ = open for appending \*/

/\* "+" sign indicates that the file must already exist \*/

**long** filepointer; /\* declaring a file pointer \*/

filepointer = **fopen**(filename, accessmode); /\* open file in append mode \*/

**fprintf** (filepointer, "%s\n", lr\_eval\_string("{cOrderId}")); /\* write orders id to output file \*/

**fclose**(filepointer); /\* close the file pointer \*/

Hope, we are now good with the concepts of reading data from or writing data to an external file (of course there could be the other ways of doing the same thing than what we discussed above). It would be very useful to do a thorough study on file operations in C (you can refer to any good C tutorials or books) as these are very much required in many real-time scenarios.

Recording Option > Network > Mapping & Filtering

* Socket level
* WinINet level
* Socket level and WinINet level

In Socket level recording, the native Socket API developed by Micro Focus (for Loadrunner) is used to interact with (and record the requests/traffic) the various application-level protocols like HTTP, FTP etc. Port mapping is relevant here.

In the WinINet level recording, the WinINet API developed by Microsoft (for Windows) is used to interact with (and record the requests/traffic) the various application-level protocols at a specified port (port mapping is not relevant here).

If we select WinINet level, LoadRunner records traffic that is generated by the application which uses the WinINet API to communicate to the servers. If we select Socket level, Vugen captures the traffic on the socket-level at a specified port.

It is not an easy answer to say which one of these to select when recording the script. It is better to use trial and error method – if the script is not recorded, switch between these options until we succeed.

As explained above, for socket-level capture, Port mapping is relevant and is used to map the traffic from a specified server-port combination to the desired communication protocol (traffic filtering is used to exclude the traffic from a specified server-port).

**Select next row**: Instructs VuGen how to select the data values during the test when multiple Vusers are running (and each Vusers is running for multiple iterations). We mainly have three options here –Sequential, Random, and Unique.

**Update value on:** Instructs VuGen when to update the data values of the parameters. We have three options here –Each iteration, Each occurrence and Once.

The combination of these two options defines what values (from the data file) are to be taken by each Vuser for each iteration and each occurrence (within an iteration).

**Let us understand each of these combinations in detail now:**

**a) Sequential-Each iteration:** Every Vuser starts from the first data value in the text file and goes to the next row value in every new iteration sequentially. If there are not enough values in the file, Vuser returns to the first value in the file (continuing in a loop till the end of the test).

**b) Sequential-Each occurrence:** Every Vuser starts from the first data value in the text file and goes to the next row value in every new occurrence (within the iteration) sequentially. If there are not enough values in the file, Vuser returns to the first value in the file (continuing in a loop till the end of the test).

**c) Sequential-Once:** Every Vuser on every iteration and every occurrence (within an iteration) takes only the first value from the data file.

**Example:** Say we have five urls and for a particular test we want to use only one (of these five), in such a case we can set this option.

**d) Random-Each iteration:** Every Vuser takes a new random data value from the text file on every iteration. Here duplication of values is possible as the selection of values is random.

**Example:** If we have an application which generates sales report for a given month (Jan, Feb …… Dec), we can set this option so that Vusers pick one of these twelve months randomly and generates a report.

**e) Random-Each occurrence:** Every Vuser takes a new random data value from the text file on every occurrence (within an iteration). Here duplication of values is possible as the selection of values is random.

We can set this option for similar scenario (sales report) as explained above; the only difference is here Vusers will pick a random month on every occurrence.

**f) Random-Once:** Every Vuser takes a random data value from the text file on the first iteration and sticks to the same value for all iterations and occurrences.

**Example:** we have an application/scenario where a user has to randomly pick up a product (from a list) and work on the same product for all iterations and occurrences, we can set this option.

**g) Unique-Each iteration:** Every Vuser for every iteration takes the first unused (unique) data value from the text file.