**File Change Log**

TechDetails - SimpleRandomSample\_00js.docx corrects bullet 12 in the “Detailed Specification: Seed Generation” section of TechDetails - SimpleRandomSample.docx. In particular, the present document states “greater than equal to 30269” in bullet 12 rather than “greater than equal to 30307” which incorrectly appeared in the previous version of this document.

**File Descriptions**

randomReplicate\_03js.r**:** Example code to replicate RAT-STATS random number generator

SpecSheet - SimpleRandomSample.xlsx: A description of the required inputs and outputs for the RAT-STATS random number generator.

Examples - SimpleRandomSample.xlsx: Two example RAT-STAT inputs.

Also refer to pages 24 through 33 (sections 2-4 through 2-13) in the RAT-STATS 2010 User Guide which can be found on the following page <https://oig.hhs.gov/compliance/rat-stats/>. The data limitations associated with RAT-STATS can be found on pages 386 through 393 (A-1 through B-5) of the RAT-STATS 2010 User Guide.

**Exception Checking**

Your program does not need to provide the exact same error messages as RAT-STATS, but exceptions handled by RAT-STATS must be handled by your submissions as well. Exceptions will be checked through the use of test datasets.

**General Overview**

The RAT-STATS random number generator is based on the Wichmann-Hill algorithm, which is relatively straightforward. Replicating the algorithm; however, is complicated by the steps that RAT-STATS uses to generate the three seed numbers required to initialize the Wichmann-Hill algorithm. In particular, the seeds are generated through repeated calls the Visual Basic pseudo-random number generator (functions Rnd and Randomize). These latter functions work primarily by bit-wise operations and by casting bit strings from one number type to another. The next section outlines the generation of the Wichmann-Hill seeds in detail. The last section goes over the Wichmann-Hill algorithm itself. Finally, the file “randomReplicate\_03js.r” contains example code for replicating the generator.

**Detailed Specification: Seed Generation**

1. Store the value 3758214 as a signed 32 bit integer.
2. Store the seed provided by the user as a double (64-bit double-precision floating-point).
3. The bit string from step 2 is truncated to include only the first 32 bits.
4. The first 16 bits and the last 16 bits of the bit string in Step 3 are then XOR’d together. As an example suppose we have the bit string 00000000111111110101010101010101. We would XOR 0101010101010101 with 0000000011111111 and get 0101010110101010.
5. The last 8 bits of the bit string from step 1 are appended to the end of the 16 bit string created in step 4.  Eight 0 value bits are then appended to the front of the bit string to result in the full 32 bit string.
6. The bit string from step 5 is cast as a 64 bit integer.
7. The integer from step 6 multiplied by 1140671485 and summed with 12820163.
8. The integer from step 7 is then divided by 2^24 and the remainder is stored (Tracking Seed A).
9. Tracking Seed A is then divided by 2^24, multiplied by 30269, and summed with 1.  If the result is not greater than equal to 30269, then it is defined as the first RAT-STATS seed (Result A). Otherwise, return to step 7 replacing the integer from step 6 with the current Tracking Seed A.
10. Tracking Seed A is cast as a 64 bit integer (if it isn’t one already), multiplied by 1140671485, and summed with 12820163.
11. The integer from step 10 is then divided by 2^24 and the remainder is stored (Tracking Seed B).
12. Tracking Seed B is then divided by 2^24, multiplied by 30307, and summed with 1.  If the result is not greater than equal to 30307, then it is defined as the first RAT-STATS seed (Result B). Otherwise, return to step 10 replacing Tracking Seed A with the current Tracking Seed B.
13. Tracking Seed B is cast as a 64 bit integer (if it isn’t one already), multiplied by 1140671485, and summed with 12820163.
14. The integer from step 13 is then divided by 2^24 and the remainder is stored (Tracking Seed C).
15. The remainder from step 14 is then divided by 2^24, multiplied by 30323, and summed with 1.  If the result is not greater than equal to 30323, then it is defined as the first RAT-STATS seed (Result C). Otherwise return to step 13 replacing Tracking Seed B with the current Tracking Seed C.
16. Result A, Result B, and Result C are the initialization values for the Wichmann-Hill algorithm which is used by RAT-STATS to generate its pseudorandom numbers.

Note: If a language allows for 64 bit integers then steps 7, 8, 10, 11, 13, and 14 can be calculated directly. If not, then an alternate method must be used to perform this calculation. Otherwise using a 32 bit integer can lead to an overflow (i.e., a number bigger than the largest number allowed by a 32 bit integer). Using a 64 bit double leads to a loss of the precision necessary to complete the exact calculation.

**Detailed Specification: Wichmann Hill Algorithm**

The Wichmann Hill Algorithm is more straightforward than the randomization in Visual Basic. Below is the pseudo code for the Wichmann Hill Algorithm as implemented in RAT-STATS. RAT-STATS generates random numbers without replacement. This means that each random number can be generated at most once within a given sample. The Wichmann Hill Algorithm as applied can produce the same random number multiple times. Consequently, RAT-STATS checks each generated number against the numbers already selected.

Definitions:

samsiz: The sum of the “Sequential Order” and “Spares in Random Order” fields

Result A: The first of the three seeds generated using Visual Basic pseudo-random number generating functions (see Step 9 from the previous section).

Result B: The first of the three seeds generated using Visual Basic pseudo-random number generating functions (see Step 12 from the previous section).

Result C: The first of the three seeds generated using Visual Basic pseudo-random number generating functions (see Step 15 from the previous section).

Universe: The difference between the “Low Number” and “High Number” fields plus one.

LowNumber: The value in the “Low Number” field.

RandomNumber: An array that contains the random numbers generated by the loop.

CheckDuplicate: Function to check if the random number has been pulled yet before.

DuplicateFlag: Used to store whether the random number has been pulled before. If it has been pulled before then the next random number is generated.

Pseudo Code

For Next iterating j from 1 to samsiz

DuplicateFlag = FALSE

Do

Term\_1=floor(Result\_A/177)

Term\_2=Result\_A - (177\*Term\_1)

Result\_A = 171\*Term\_2 - 2\*Term\_1

if (Result\_A <= 0) then {Result\_A = Result\_A +30269}

Term\_1=floor(Result\_B/176)

Term\_2=Result\_B - (176\*Term\_1)

Result\_B = 172\*Term\_2 - 35\*Term\_1

if (Result\_B <= 0) then {Result\_B = Result\_B +30307}

Term\_1=floor(Result\_C/178)

Term\_2=Result\_C - (178\*Term\_1)

Result\_C = 170\*Term\_2 - 63\*Term\_1

if (Result\_C <= 0) then {Result\_C = Result\_C +30323}

Term\_4 = Result\_A/30269 + Result\_B/30307 + Result\_C/30323

TempRandom = floor((Term\_4 - floor(Term\_4))\*Universe)+LowNumber

if (CheckDuplicate(TempRandom)) then

{DuplicateFlag = TRUE

}else{RandomNumber[j] = TempRandom}

While DuplicateFlag

End Loop