TE003 Test Report on the Module codecs_lib.wichmann_hill

Conventions

Each test is defined following the same format. Each test receives a unique test identifier and a reference to the ID(s) of the requirements it covers (if applicable). The goal of the test is described to clarify what is to be tested. The test steps are described in brief but clear instructions. For each test it is defined what the expected results are for the test to pass. Finally, the test result is given, this can be only pass or fail.

The test format is as follows:

Test Identifier: TEST-[I/A/D/T]-XYZ

Requirement ID(s): REQ-uvw-xyz

Verification method: I/A/D/T

Test goal: Description of what is to be tested

Expected result: What test result is expected for the test to pass

Test steps: Step by step instructions on how to perform the test

Test result: PASS/FAIL

The test ID starts with the fixed prefix 'TEST'. The prefix is followed by a single letter, which defines the test type / verification method. The last part of the ID is a 3-digits *hexadecimal* number (0..9|A..F), with the first digit identifing the module, the second digit identifing a class / function, and the last digit - the test ordering number for this object. E.g. 'TEST-T-112'. Each test type has its own counter, thus 'TEST-T-112' and 'TEST-A-112' tests are different entities, but they refer to the same object (class or function) within the same module.

The verification method for a requirement is given by a single letter according to the table below:

Term	Definition
Inspection (I)	Control or visual verification
Analysis (A)	Verification based upon analytical evidences
Test (T)	Verification of quantitative characteristics with quantitative measurement
Demonstration (D)	Verification of operational characteristics without quantitative measurement

Test preparations

Implement the Wichmann-Hill alogorithm in a spreadsheet and generate the expected / benchmark pseudo-random sequences using different combination of the seed values. The first test case set should be the same as the default seed values.

Test definitions (Test)

Test Identifier: TEST-T-300

Requirement ID(s): REQ-FUN-302

Verification method: T

Test goal: Test the continous encoding and decoding mode.

Expected result: Encoding an decoding results are as expected, no exception is raised.

Test steps:

- Instantiate the coder class without arguments and the WH generator - also without arguments
- Seed the coder with the 3 random values, and the generator with the same values
- Create empty lists to store the original random numbers *Initial* and their processed counterparts - *Encoded*
- Generate a random integer number and add it into *Initial* list
- Encode that number and store the result in the *Encoded* list
- Get a random number from the WH generator and calculate the expected encoded value
- Compare the obtained and calculated encoded values using *almostEqual* test
- Generate a random integer number and add it into *Initial* list
- Decode that number and store the result in the *Encoded* list
- Get a random number from the WH generator and calculate the expected decoded value
- Compare the obtained and calculated decoded values using *almostEqual* test
- Generate a short list of mixed integer and random floating point numbers (*N* about 20 elements)
- Extend the *Initial* list with the just generated list twice
- Encode the entire just generated list (passed as the argument) and extend the *Encoded* list with the result
- Decode the entire generated list (passed as the argument) and extend the *Encoded* list with the result
- Generate a random number using the generator class 2 * *N* times
- Generate a random floating point number and add it into *Initial* list
- Encode that number and store the result in the *Encoded* list
- Get a random number from the WH generator and calculate the expected encoded value
- Compare the obtained and calculated encoded values using *almostEqual* test
- Generate another random floating point number and add it into *Initial* list
- Decode that number and store the result in the *Encoded* list
- Get a random number from the WH generator and calculate the expected decoded value
- Compare the obtained and calculated decoded values using *almostEqual* test
- Decode the first element from the *Encoded* list and compare it with the first element of *Initial* list using *almostEqual* test
- Encode the second element from the *Encoded* list and compare it with the second element of *Initial* list using *almostEqual* test
- Decode as a single sequence the slice *Encoded*[2 : 2 + *N*] and compare per-element (*almostEqual* test) to the corresponding slice of *Initial*[2 : 2 + *N*]
- Encode as a single sequence the slice *Encoded*[2 + *N* : 2 + 2 * *N*] and compare per-element (*almostEqual* test) to the corresponding slice of *Initial*[2 + *N* : 2 + 2 * *N*]

- Decode the next to the last element from the *Encoded* list and compare it with the next to the last element of *Initial* list using *almostEqual* test
- Encode the last element from the *Encoded* list and compare it with the last element of *Initial* list using *almostEqual* test
- Repeat the entire procedure multiple times starting from the seeding with the random values
- Delete the created instances of the generator and coder classes

Test result: PASS

Test Identifier: TEST-T-310

Requirement ID(s): REQ-FUN-310

Verification method: T

Test goal: Check that the pseudo-random generated is implemented properly according to the modified Wichmann-Hill algorithm.

Expected result: Being seeded with the known numbers the generator yields a the expected sequence of the pseudo-random numbers. Re-seeding the generator with the same numbers results in the same sequence of the pseudo-random numbers.

Test steps:

- Instantiate the generator class without arguments, i.e. with the default seed values
- Take the first prepared set of the seed values and the expected generated sequence
- For each number in the expected sequence:
    - Generate a random number using the generator being tested
    - Compare that the generated number is *almost equal* (using default precision of 7 digits) to the expected one
- For each of the prepared sets:
    - Re-seed the generator with the selected seed values
    - For each number in the expected sequence:
        - Generate a random number using the generator being tested
        - Compare that the generated number is *almost equal* (using default precision of 7 digits) to the expected one
    - Re-seed the generator again with the same values and repeat the per-element comparison

Test result: PASS

Test Identifier: TEST-T-311

Requirement ID(s): REQ-AWM-300

Verification method: T

Test goal: Check that an improper type seed passed into the pseudo-random numbers generator class raises **TypeError** or its sub-class exception.

Expected result: A sub-class of **TypeError** is raised with each improper call.

Test steps:

- For each of the defined 'bad seed type' values try to instantiate the random number generator class as:
    - Using this value as the single argument
    - Using 1 value as the first argument and this value as the second
    - Using 1 value as the first and the second arguments and this value as the third
    - Check that a sub-class of **TypeError** exception is raised in each case
- Instantiate the class without arguments, i.e. with the default seed values
- For each of the defined 'bad seed type' values try to re-seed as:
    - Using this value as the first argument, and 1 values for the second and third arguments
    - Using 1 value as the first and the third arguments, and this value as the second
    - Using 1 value as the first and the second arguments and this value as the third
    - Check that a sub-class of **TypeError** exception is raised in each case
- Delete the created instance

Test result: PASS

Test Identifier: TEST-T-312

Requirement ID(s): REQ-AWM-301

Verification method: T

Test goal: Check that an improper value range integer seed passed into the pseudo-random numbers generator class raises **ValueError** or its sub-class exception.

Expected result: A sub-class of **ValueError** is raised with each improper call.

Test steps:

- Try to instantiate the random number generator class as:
    - Using zero as the single argument
    - Using 1 value as the first argument and zero as the second
    - Using 1 value as the first and the second arguments and zero as the third
    - Check that a sub-class of **ValueError** exception is raised in each case
- Repeat the previous steps multiple times (e.g., 100) using a randomly generated negative integer ([-1, -1000000000]) instead of zero
- Instantiate the class without arguments, i.e. with the default seed values
- Try to re-seed as:
    - Using zero as the first argument, and 1 values for the second and third arguments
    - Using 1 value as the first and the third arguments, and zero as the second
    - Using 1 value as the first and the second arguments and zero as the third
    - Check that a sub-class of **TypeError** exception is raised in each case
- Repeat the re-seeding part of test multiple times (e.g., 100) using a randomly generated negative integer ([-1, -1000000000]) instead of zero
- Delete the created instance

Test result: PASS

Test Identifier: TEST-T-320

Requirement ID(s): REQ-FUN-320

Verification method: T

Test goal: Check that the encoding is performed according the algorithm description

Expected results: The returned encoded values *almost equal* to the manually calculated, expected values using the already tested WH generator seeded with the same initial values. No exceptions are raised in the process.

Test steps:

- Instantiate the coder class without arguments and the WH generator - also without arguments
- Seed the coder with the 3 random values, and the generator with the same values
- Generate a positive random integer number
- Encode the random number
- Get a random number from the WH generator and calculate the expected encoded value
- Compare the obtained and calculated encoded values using *almostEqual* test
- Repeat the check using a random negative integer, random positive and random negative floating point numbers
- Repeat the same checks multiple times with random +/- int and float values
- Generate a long list of random floating point numbers and append a random integer to the end
- Encode the entire list (passed as the argument) - the result should be a list of the same length
- For each element in the encoded values list:
    - Get a random number from the WH generator and calculate the expected encoded value for the same positional index element in the intial, input data list
    - Compare the obtained and calculated encoded values using *almostEqual* test
- Convert the generated input data list into a tuple, encode the tuple and repeat the per-element checks
- Repeat the entire procedure multiple times starting from the seeding with the random values
- Delete the created instances of the generator and coder classes

Test result: PASS

Test Identifier: TEST-T-321

Requirement ID(s): REQ-FUN-321

Verification method: T

Test goal: Check that the decoding is performed according the algorithm description

Expected results: The returned decoded values *almost equal* to the manually calculated, expected values using the already tested WH generator seeded with the same initial values. No exceptions are raised in the process.

Test steps:

- Instantiate the coder class without arguments and the WH generator - also without arguments
- Seed the coder with the 3 random values, and the generator with the same values
- Generate a positive random integer number
- Decode the random number
- Get a random number from the WH generator and calculate the expected decoded value
- Compare the obtained and calculated encoded values using *almostEqual* test
- Repeat the check using a random negative integer, random positive and random negative floating point numbers
- Repeat the same checks multiple times with random +/- int and float values
- Generate a long list of random floating point numbers and append a random integer to the end
- Decode the entire list (passed as the argument) - the result should be a list of the same length
- For each element in the decoded values list:
    - Get a random number from the WH generator and calculate the expected decoded value for the same positional index element in the intial, input data list
    - Compare the obtained and calculated decoded values using *almostEqual* test
- Convert the generated input data list into a tuple, decode the tuple and repeat the per-element checks
- Repeat the entire procedure multiple times starting from the seeding with the random values
- Delete the created instances of the generator and coder classes

Test result: PASS

Test Identifier: TEST-T-322

Requirement ID(s): REQ-AWM-300

Verification method: T

Test goal: Check that an improper type seed passed into the coder class raises **TypeError** or its sub-class exception.

Expected result: A sub-class of **TypeError** is raised with each improper call.

Test steps:

- For each of the defined 'bad seed type' values try to instantiate the coder class as:
    - Using this value as the single argument
    - Using 1 value as the first argument and this value as the second
    - Using 1 value as the first and the second arguments and this value as the third
    - Check that a sub-class of **TypeError** exception is raised in each case
- Instantiate the class without arguments, i.e. with the default seed values
- For each of the defined 'bad seed type' values try to re-seed as:
    - Using this value as the first argument, and 1 values for the second and third arguments
    - Using 1 value as the first and the third arguments, and this value as the second
    - Using 1 value as the first and the second arguments and this value as the third
    - Check that a sub-class of **TypeError** exception is raised in each case
- Delete the created instance

Test result: PASS

Test Identifier: TEST-T-323

Requirement ID(s): REQ-AWM-301

Verification method: T

Test goal: Check that an improper value range integer seed passed into the coder class raises **ValueError** or its sub-class exception.

Expected result: A sub-class of **ValueError** is raised with each improper call.

Test steps:

- Try to instantiate the coder class as:
  - Using zero as the single argument
  - Using 1 value as the first argument and zero as the second
  - Using 1 value as the first and the second arguments and zero as the third
  - Check that a sub-class of **ValueError** exception is raised in each case
- Repeat the previous steps multiple times (e.g., 100) using a randomly generated negative integer ([-1, -1000000000]) instead of zero
- Instantiate the class without arguments, i.e. with the default seed values
- Try to re-seed as:
  - Using zero as the first argument, and 1 values for the second and third arguments
  - Using 1 value as the first and the third arguments, and zero as the second
  - Using 1 value as the first and the second arguments and zero as the third
  - Check that a sub-class of **TypeError** exception is raised in each case
- Repeat the re-seeding part of test multiple times (e.g., 100) using a randomly generated negative integer ([-1, -1000000000]) instead of zero
- Delete the created instance

Test result: PASS

Test Identifier: TEST-T-324

Requirement ID(s): REQ-AWM-320

Verification method: T

Test goal: Check that an improper input type for the encoding / decoding raises **TypeError** or its sub-class exception.

Expected result: A sub-class of **TypeError** is raised with each improper call.

Test steps:

- Instantiate the coder class without arguments, i.e. using the default seed values
- For each of the defined 'bad input type' values try to encode such value and check that a sub-class of **TypeError** exception is raised
- Repeat the following multiple times, e.g. 100
  - Re-seed the coder with 3 arbitrary / random integer numbers in the range [1, 1000000000]

    - For each of the defined 'bad input type' values try to encode such value and check that a sub-class of **TypeError** exception is raised
- Delete the created instance
- Repeat the previous steps using decoding method instead

Test result: PASS

Test definitions (Analysis)

Test Identifier: TEST-A-300

Requirement ID(s): REQ-FUN-300, REQ-FUN-301

Verification method: A

Test goal: Test that the module implement all of the required functionality and it performs according to the requirements and specifications.

Expected result: All of the unit tests defined by the test cases TEST-T-300, TEST-T-310, TEST-T-310, TEST-T-311, TEST-T-312, TEST-T-320, TEST-T-321, TEST-T-322, TEST-T-323 and TEST-T-324 must pass.

Test steps: Run the test suit module codecs_lib.tests.ut003_wichmann_hill.py

Test result: PASS / FAIL

Traceability

For traceability the relation between tests and requirements is summarized in the table below:

Requirement ID	**Covered in test(s)**	**Verified [YES/NO])**
REQ-FUN-300	TEST-A-300	YES
REQ-FUN-301	TEST-A-300	YES
REQ-FUN-302	TEST-T-300	YES
REQ-FUN-310	TEST-T-310	YES
REQ-FUN-320	TEST-T-320	YES
REQ-FUN-321	TEST-T-321	YES
REQ-AWM-300	TEST-T-311, TEST-T-322	YES
REQ-AWM-301	TEST-T-312, TEST-T-323	YES
REQ-AWM-320	TEST-T-324	YES

Software ready for production [YES/NO]	**Rationale**
YES	All tests are passed