Test Report on the Module com_lib.serialization

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

Tests preparations

Implement the test cases and the specific sub-classes on which the tests will be performed (see ut003_serialization.py).

Declare the following derived classes:

- **BaseStruct**: SerStruct(a: c_short, b: c_float) - 2 + 4 = 6 bytes

- **BaseArray**: SerArray(c_short[2]) - 2 x 2 = 4 bytes
- **BaseDynamicArray**: SerDynamicArray(c_short[]) - ? x 2 bytes
- **NestedStruct**: SerStruct(a: c_short, b: c_float, c: BaseArray) - 2 + 4 + 2 x 2 = 10 bytes
- **NestedDynamicStruct**: SerStruct(a: c_short, b: c_float, c: BaseDynamicArray) - 2 + 4 + ? x 2 = (6 + ? x 2) bytes
- **NestedArray**: SerArray(BaseStruct[2]) - (2 + 4) x 2 = 12 bytes - base test class for the fixed length array implementation
- **NestedDynamicArray**: SerDynamicArray(BaseStruct[]) - (2 + 4) x ? = 6 x ? bytes - base test class for the dynamic length array implementation
- **ComplexStruct**: SerStruct(a: c_short, b: c_float, c: NestedDynamicStruct) - 2 + 4 + (2 + 4 + ? x 2) = (12 + ? x 2) bytes - base test class for the structure implementation
- **ArrayArray**: SerArray(BaseArray[3]) - (2 x 2) x 3 = 12 bytes
- **DynamicArrayArray**: SetDynamicArray(BaseArray[]) - (2 x 2) x ? = (4 x ?) bytes

Define the unit test cases as methods of the unit test suits (respective test classes).

Additionally, define a number of badly declared classes:

- Struct
  - **BadStruct1** - not string name of a field
  - **BadStruct2** - scalar value (instance) instead of type (class) as a field type
  - **BadStruct3** - **float** as a field type
  - **BadStruct4** - **BaseDynamicArray** (not fixed length) type of a field not in the final position
  - **BadStruct5** - **ComplexStruct** (not fixed length) type of a field not in the final position
  - **BadStruct6** - **BadStruct1** as the field type (nested bad declaration)
  - **BadStruct7** - **BadArray10** as the field type (nested bad declaration)
- Fixed length array
  - **BadArray1** - negative length of the array
  - **BadArray2** - zero length of the array
  - **BadArray3** - instance of **ctypes.c_int** instead of just **int** for the array length
  - **BadArray4** - floating point value instead of integer for the array length
  - **BadArray5** - data type (class) instead of value (instance) for the array length
  - **BadArray6** - scalar value (instance) instead of data type (class) as the elements type
  - **BadArray7** - **int** as the elements type
  - **BadArray8** - **BaseDynamicArray** as the element type (not fixed length)
  - **BadArray9** - **ComplexStruct** as the element type (not fixed length)
  - **BadArray10** - **BadStruct6** as the element type (nested bad declaration)
- Dynamic length array
  - **BadDynamicArray1** - scalar value (instance) instead of data type (class) as the elements type
  - **BadDynamicArray2** - **int** as the elements type
  - **BadDynamicArray3** - **BaseDynamicArray** as the element type (not fixed length)
  - **BadDynamicArray4** - **ComplexStruct** as the element type (not fixed length)
  - **BadDynamicArray15** - **BadStruct6** as the element type (nested bad declaration)

Test definitions (Analysis)

Test Identifier: TEST-A-300

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

Verification method: A

Test goal: Check that the recruired classes are implemented, and they deliver the serializaton and de-serialization functionality.

Expected result: The module defines the classes for: NULL object, C-like structure, fixed and dynamic length arrays. All these classes provide methods to serialized their stored content into a packed bytestring and JSON format string, as well as methods to create new instances of these classes from the serialized data exactly matching the content of the original serialized objects.

Test steps: Review the source code. Execute all test cases defined in the test module ut003_serialization.py.

Test result: PASS

Test definitions (Test)

Test Identifier: TEST-T-300

Requirement ID(s): REQ-FUN-302

Verification method: T

Test goal: Check the presence of the required methods and their type: class or instance method.

Expected result: All concerned classes have class methods *getSize*, *unpackBytes*, *unpackJSON* and the instance methods *packBytes*, *packJSON*, *getNative*

Test steps: Perform the test on the class, not on an instance (do not instantiate). Check that the class has the required attributes using the built-in Python function *hasattr*() and the unittest method *assertTrue*(), than access the attribute using the Python built-in function *getattr*() and check its type using the unittestmethod *assertIsInstance*(): the class methods should be **types.MethodType** and the instance - **types.FunctionType**.

Implemented as method *Test_Basis.test_API* inherited by the actual test suits.

Test result: PASS

Test Identifier: TEST-T-301

Requirement ID(s): REQ-AWM-305

Verification method: T

Test goal: Check the limitation on the attribute access.

Expected result: The following limitations on the attribute access are implemented on top of the standard Python attribute resolution scheme:

- New instance attributes can not be added in the run-time

- The values of the attributes cannot be changed, except for the declared fields of a structure
- The values of the *magic* and *private* (names starting with, at least, one underscore) attrubutes cannot be accessed; except for the attributes *__name__* and *__cls__*, which access may be allowed.

A sub-class of **AttributeError** is raised upon violation of the attribute resolution limitations.

Test steps: This test must be performed on an instance, not on a class.

- Try to assign any value to a non-existing attribute. Check that the expected exception is raised.
- Try to assign any value to the following attributes: *__dict__* (must be present on all instances of the classes being tested), *_Fields* (must be present in structures), *_ElementType* and *_Length* (must be present in arrays). Check that the expected exception is raised in all cases.
- Try to assign any value to a method attribute (*getSize*, *unpackBytes*, *unpackJSON*, *packBytes*, *packJSON*, *getNative*), which must be present in all classes. Check that the expected exception is raised in all cases.
- Try to access the value of the following attributes: *__dict__* (must be present on all instances of the classes being tested), *_Fields* (must be present in structures), *_ElementType* and *_Length* (must be present in arrays). Check that the expected exception is raised in all cases.
- Try to access the value of a non-exitsting attribute. Check that the expected exception is raised.

Implemented as methods *Test_Basis.test_Read_AttributeError* and *Test_Basis.test_Write_AttributeError* inherited by the actual test suits.

Test result: PASS

Test Identifier: TEST-T-302

Requirement ID(s): REQ-AWM-303

Verification method: T

Test goal: Check that **TypeError** or its sub-class is raised with an improper type argument is passed into the unpacking methods.

Expected result: TypeError or its sub-class is raised if:

- *unpackJSON* method receives any type but string argument
- The native Python data type not compatible with the declared class data structure is encoded in the JSON string passed into the class method *unpackJSON*()
- *unpackBytes* method receives any type but bytestring argument

Test steps:

- Try to call *unpackJSON* method on the class with any type of the argument except for the string. Check that **TypeError** or its sub-class exception is raised.
- Repeat with the different incompatible types of the argument.
- Try to unpack a proper JSON object using method *unpackJSON*, which doen't match the declared data structure of the class:
    - Not a dictionary for structure
    - Not a list for arrays

- Check that **TypeError** or its sub-class exception is raised.
- Repeat with the different incompatible types of the argument.
- Try to call *unpackBytes* method on the class with any type of the argument except for the bytestring. Check that **TypeError** or its sub-class exception is raised.
- Repeat with the different incompatible types of the argument.

Implemented as the methods *Test_Basis.test_unpackJSON_TypeError* and *Test_Basis.test_unpackBytes_TypeError* as well as *test_unpackJSON_TypeError* methods of the derived test suit classes **Test_SerNULL**, **Test_SerStruct**, **Test_SerArray** and **Test_SerDynamicArray**.

Test result: PASS

Test Identifier: TEST-T-303

Requirement ID(s): REQ-AWM-304

Verification method: T

Test goal: Check that **ValueError** or its sub-class is raised with an improper type argument is passed into the unpacking methods.

Expected result: The **ValueError** or its sub-class exception is raised if:

- A string, but not a proper JSON object is passed into the method *unpackJSON*
- A proper JSON object is passed into the method *unpackJSON*, which, however, does not match the internal declared data structure of the class
- Too short or too long bytestring is passed into the method *unpackBytes*, i.e. its length doesn't match the declared data size of the class

Test steps:

- Try to unpack an arbitrary string using method *unpackJSON*, which is not a proper JSON object representation.
- Try to unpack a proper JSON object using method *unpackJSON*, which doen't match the declared data structure of the class:
    - Dictionary with the missing keys (declared fields) for structure
    - Dictionary with the keys not matching the declared fields for structure
    - Dictionary with, at least, ony key holding the wrong data type (as declared for the respective field) for structure
    - List containing too few elements for the fixed length array
    - List containing too many elements for the fixed length array
    - List with, at least, one element being of the type incompatible with the declared data type of the array elements - both fixed and dynamic
- Try to unpack a bytestring, which is too short or too long compared to the size of the declared data structure; for the dynamic length arrays - the length of the bytestring is not a multiple of the size of an element

Implemented as *Test_Basistest_Improper_JSON*() method; *test_unpackJSON_ValueError* and *test_unpackBytes_ValueError* methods of the test suit classes **Test_SerStruct**, **Test_SerArray** and

Test_SerDynamicArray as well as *Test_SerNULL.test_unpackBytes_ValueError*.

Test result: PASS

Test Identifier: TEST-T-304

Requirement ID(s): REQ-AWM-301

Verification method: T

Test goal: Check that the improper type of the argument of the initialization method is treated as an error.

Expected result: The **TypeError** or its sub-class exception is raised if the type of the passed into initializer is not compatible with the class declared data structure:

- Not a mapping type or an instance of structure class - for a structure
- Not a sequence type or an instance of dynamic or fixed length structure class - for a dynamic or fixed length array

Test steps:

- Try to instantiate a class being tested with an argument of a wrong type.
- Check that the **TypeError** or its sub-class exception is raised
- Repeat the process with several different types of the argument.

Implemented as *test_init_TypeError* methods of the test suit classes **Test_SerStruct**, **Test_SerArray** and **Test_SerDynamicArray**.

Test result: PASS

Test Identifier: TEST-T-305

Requirement ID(s): REQ-AWM-300, REQ-FUN-320, REQ-FUN-330, REQ-FUN-340

Verification method: T

Test goal: Check the implementation of the data structure declaration check.

Expected result: The classes (struct and arrays) with the data structure declaration confirming the rules described in the requirements can be instantiated and their class and instance methods can be called. If the data structure declaration is incorrected, those classes cannot be instantiated - an exception (sub-class of **TypeError**) is raised. The same exception is raised upon calling the class methods on such classes without instantiation. The improper definition examples are:

- Arrays:
    - Any data type of the elements except C primitives (scalars), fixed length arrays or fixed length structs
    - Anything but positive integer as the declared length (only fixed arrays)
- Structs:
    - Fields defiition is not a tuple of 2 element tuples
    - Any field name is not a string (first element of the nested tuples)

    - Any field declared type (second element of the nested tuples) is anything but C primitive (scalar), array or structs
    - A field declared as a dynamic array or nested structure containg a dynamic array is not the last field declared

These limitations are applicable recursively to the nested elements.

Test steps:

- Try to instantiate several properly defined clases (part of TEST-T-320, TEST-T-330 and TEST-T-340). No exception is raised. Try to call some of their class methods. No exception should be raised.
- Try to instantiate several wrongly defined classes (see Tests Preparation section). Sub-class of **TypeError** must be raised.
- On the same classes (without instantiation) try to call the following class methods - and check that **TypeError** is raised:
    - *getSize*() - no arguments, all classes
    - *unpackJSON*() - arbitrary JSON dictionary string argument for struct, arbitrary JSON array string argument - arrays
    - *unpackBytes*() - arbitrary bytestring argument, all classes
    - *getMinSize*() - no argument, only struct
    - *getElementSize*() - no argument, only dynamic length arrays

The tests raising **TypeError** are implemented as a separate test suite **Test_BadDeclaration**, specificially the methods *test_BadStructures*(), *test_BadArrays*() and *test_BadDynamicArrays*().

Test result: PASS

Test Identifier: TEST-T-306

Requirement ID(s): REQ-AWM-306

Verification method: T

Test goal: Check the implementation of the index access limitations.

Expected result: Only the integer numbers can be used as the index and only if its value is greater then or equal to the minus length of the array but less than the length of the array (i.e. -len(Array) <= Index <= (len(Array) -1)). Any other type or value used for the index access must result in **IndexError** exception or its sub-class

Test steps: Perform the following tests:

- Instantiate **BaseArray** (length is 2, elements type is **ctypes.c_short**) without arguments.
- Try to read-access elements using the following index values: -3, 2, 1.0, **int**, **float**, '1', c_short(1), and (slices) [0:1], [:-1], [0:]. Check that an instance of sub-class of **IndexError**.
- Try to assign a proper integer value of 1 to an element using the same index values as in the previous test. Check that an instance of sub-class of **IndexError**.
- Instantiate **BaseDynamicArray** with a list [1, 2].
- Try to read-access elements using the following index values: -3, 2, 1.0, **int**, **float**, '1', c_short(1), and (slices) [0:1], [:-1], [0:]. Check that an instance of sub-class of **IndexError**.

- Try to assign a proper integer value of 1 to an element using the same index values as in the previous test. Check that an instance of sub-class of **IndexError**.

Implemented as method *test_IndexError* of the test suit classes **Test_SerArray** and **Test_SerDynamicArray**.

Test result: PASS

Test Identifier: TEST-T-307

Requirement ID(s): REQ-AWM-307

Verification method: T

Test goal: Check that only compatible native Python values can be assigned to the struct's fields or array elements, and only if the field's / element type is declared as the C privimitive.

Expected result: An exception of the sub-class of **TypeError** is raised upon assignment to a structure field or array element if: a) the respective field / element is not a C primitive declared type (i.e. a nested struct or array), OR b) the respective field / element is declared as C primitive, but the value to be assigned in not a native Python type compatible with that C primitive.

Test steps: Perform the following tests:

- For struct
  - Instantiate **ComplexStruct** with the argument {'c' : {'c' : [1, 2]}} as *objTest*
  - Try to assign different not integer values to *objTest.a*, *objTest.c.a* and *objTest.c.c[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
  - Try to assign different not numeric values to *objTest.b* and *objTest.c.b*. Check that a sub-class of **TypeError** exception is raised in all cases.
  - Try to assign different values, including native Python scalars, dict and list to *objTest.c* and *objTest.c.c*. Check that a sub-class of **TypeError** exception is raised in all cases.
- For fixed length array
  - Instantiate **BaseArray** without argument as *objTest*
  - Try to assign different not integer values to *objTest[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
  - Instantiate **NestedArray** without argument as *objTest*
  - Try to assign different not integer values to *objTest[0].a*. Check that a sub-class of **TypeError** exception is raised in all cases.
  - Try to assign different values, including native Python scalars and dict to *objTest.[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
  - Instantiate **ArrayArray** without argument as *objTest*
  - Try to assign different not integer values to *objTest[0][0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
  - Try to assign different values, including native Python scalars and list to *objTest.[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
- For dynamic length array
  - Instantiate **BaseDynamicArray** with the [1, 1] argument as *objTest*

    - Try to assign different not integer values to *objTest[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
    - Instantiate **NestedArray** with the [{'a': 1}, {'a' : 1}] argument as *objTest*
    - Try to assign different not integer values to *objTest[0].a*. Check that a sub-class of **TypeError** exception is raised in all cases.
    - Try to assign different values, including native Python scalars and dict to *objTest.[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
    - Instantiate **ArrayArray** with the [[1, 1], [1, 1]] argument as *objTest*
    - Try to assign different not integer values to *objTest[0][0]*. Check that a sub-class of **TypeError** exception is raised in all cases.
    - Try to assign different values, including native Python scalars and list to *objTest.[0]*. Check that a sub-class of **TypeError** exception is raised in all cases.

Implemented as the method *test_assignment_TypeError*() of the test suit classes **Test_SerStruct**, **Test_SerArray** and **Test_SerDynamicArray**

Test result: PASS

Test Identifier: TEST-T-308

Requirement ID(s): REQ-AWN-302

Verification method: T

Test goal: Check the implementation of the data sanity checks on the keys / elements being copied from the argument passed into initialization method of the implemented classes.

Expected result: A sub-class of **ValueError** exception is raised if:

- A dictionary is passed into the initializer of struct class with, at least, one key with the same name as any of the declared fields holds a value not compatible with the declared data type of that field
- A sequence is passed into the initializer of a fixed length array with, at least, one element being incompatible with the declared type of the arrray elements and its index being smaller than the declared length of the array
- A sequence is passed into the initializer of a dynamic length array with, at least, one element being incompatible with the declared type of the arrray elements

Test steps: Perform the following process:

- Try to instantiate **ComplexStruct** with each of the values {'a' : 1.0}, {'c' : [1, 2]}, {'c' : {'a' : 1.0}} and {'c' : {'c' : [2, 1.0]}}
- Try to instantiate **BaseArray** with each of the values [1.0], [1, 1.0], [1, [1, 2]]
- Try to instantiate **BaseDynamicArray** with the same values
- Try to instantiate **NestedArray** with each of the values [{'a' : 1.0}], [{}, {'a' : 1.0}], [[1, 2]]
- Try to instantiate **NestedDynamicArray** with the same values
- Try to instantiate **ArrayArray** with each of the values [[1.0, 1], [1, 1]], [{}, [1, 2]], [[1, 2], [1.0]]
- Try to instantiate **DynamicArrayArray** with the same values
- Check that the expected exception is raised in each case

Implemented as the method *test_init_ValueError*() of the test suit classes **Test_SerStruct**, **Test_SerArray** and **Test_SerDynamicArray**

Test result: PASS

Test Identifier: TEST-T-309

Requirement ID(s): REQ-FUN-303, REQ-FUN-323, REQ-FUN-324, REQ-FUN-333, REQ-FUN-334, REQ-FUN-343, REQ-FUN-344

Verification method: T

Test goal: Check the implementation of the serialization into a bytestring (bytes packing) and de-serialization from a bytestring (bytes unpacking) as well as support for the little- and big- endian byte order

Expected result: The following functionality is implemented

- All defined classes provide an instance method to create a bytestring representing the entire stored data, with each separate value (field or element) being represented by a number of hole bytes (no support for bit-fields)
- All defined classes provide a class method to create a new instance from a bytestring representing the entire data from another instance of the same class
- The packing and unpacking methods are mutually inverse, i.e. 1) packing -> unpacking sequence creates the exact copy of the initial object (class instance), and 2) unpacking -> packing sequence creates the exact copy of the bytestring
- The both methods support big- endian and little- endian byte order, which affects only the order of bytes in the representation of a single value (field or element), but not the order of fields / elements
    - E.g., in the big-endian order **c_short** (**c_int16**) value of 1 is represented as b'\x00\x01', whereas *single precision* floating point (**c_float**) value of 1.0 as b'\x3F\x80\x00\x00'
    - In the little-endian order **c_short** (**c_int16**) value of 1 is represented as b'\x01\x00', whereas *single precision* floating point (**c_float**) value of 1.0 as b'\x00\x00\x80\x3F'
    - The specific byte order must be explicitely indicated (requested by an optional argument value), otherwise the native endianness for the host platform is used

Test steps: Perform the following checks:

- Check that an instance of **SerNULL* class always packs into an empty bytestring (b'') regardless of the requested endianness
- Check that a new instance of **SerNULL** class is created from an empty bytestring regardless of the requested endianness, and it evaluates to **None** (using *getNative*() instance method)
- Checks for **SerArray** class:
    - Instantiate **ComplexStruct** with the dictionary {'a' : 1, 'b' : 1.0, 'c' : {'a' : 2, 'b' : 1.0, 'c' : []}}
    - With explicit request for little endian it should be byte-packed into b'\x01\x00\x00\x00\x80\x3F\x02\x00\x00\x00\x80\x3F'
    - With explicit request for big endian it should be byte-packed into b'\x00\x01\x3F\x80\x00\x00\x00\x02\x3F\x80\x00\x00'
    - Without indication (native order) it should be byte-packed into either the first or the second variant - check the endianness of the platform

    - Unpack the little-endian encoded bytestring using the corresponding class method with the explicit request for little endian byte order. Check that an instance of the proper class is created, and its method *getNative*() returns the exact copy of the initial dictionary.
    - Unpack the big-endian encoded bytestring using the corresponding class method with the explicit request for big endian byte order. Check that an instance of the proper class is created, and its method *getNative*() returns the exact copy of the initial dictionary.
    - Unpack either the little- or big- endian encoded string (depending on the platform endianness) without indication of the required endiannes. Check that an instance of the proper class is created, and its method *getNative*() returns the exact copy of the initial dictionary.
    - Repeat the steps using a different dictionary 'a' : 1, 'b' : 1.0, 'c' : {'a' : 2, 'b' : 1.0, 'c' : [3, 4]}} - elements added to the nested dynamic array, where the expected bytestring encoded data is:
        - b'\x01\x00\x00\x00\x80\x3F\x02\x00\x00\x00\x80\x3F\x03\x00\x04\x00' - little endian
        - b'\x00\x01\x3F\x80\x00\x00\x00\x02\x3F\x80\x00\x00\x00\x03\x00\x04' - big endian
- Checks for **SerArray** class - same process flow logic
    - Class **BaseArray**
        - Instantiation argument - list [1, 2]
        - Little endian bytestring - b'\x01\x00\x02\x00'
        - Big endian bytestring - b'\x00\x01\x00\x02'
    - Class **NestedArray**
        - Instantuation argument - list [{'a' : 1, 'b' : 1.0}, {'a' : 2, 'b' : 1.0}]
        - Little endian bytestring - b'\x01\x00\x00\x00\x80\x3F\x02\x00\x00\x00\x80\x3F'
        - Big endian bytestring - b'\x00\x01\x3F\x80\x00\x00\x00\x02\x3F\x80\x00\x00'
    - Class **ArrayArray**
        - Instantuation argument - list [[1, 2], [3, 4], [5, 6]]
        - Little endian bytestring - b'\x01\x00\x02\x00\x03\x00\x04\x00\x05\x00\x06\x00'
        - Big endian bytestring - b'\x00\x01\x00\x02\x00\x03\x00\x04\x00\x05\x00\x06'
- Checks for **SerDynamicArray** - same process flow logic, use sub-classes **BaseDynamicArray**, **NestedDynamicArray** and **DynamicArrayArray** instead

Implemented as a separate test suite **Test_BytesSerialization**

Test result: PASS

Test Identifier: TEST-T-310

Requirement ID(s): REQ-FUN-310

Verification method: T

Test goal: Check the basic functionality of the 'NULL' object.

Expected result: A new instance can be created using initialization method without argument, de-serialization (class methods) from "null" JSON string or an empty bytestring. In all cases the instance returns None as the native Python representation and is serialized into null" JSON string or an empty bytestring.

Test steps: Execute unit-test method *Test_SerNULL.test_main_functionality*, which performs the described steps.

Test result: PASS

Test Identifier: TEST-T-311

Requirement ID(s): REQ-FUN-311

Verification method: T

Test goal: Check the instantiation options for the 'NULL' object.

Expected result: Any type argument can be passed into the instatiation method. No exception is raised. The created instance represents NULL / None regardless of the passed value.

Test steps:

- Instantiate the class **SerNULL** with an arbitrary value
- Check that *getNative* method returns None
- Check that *packJSON* method returns the string "null"
- Check that *packBytes* method returns an empty bytestring
- Repeat the checks with a different type of the passed value

Implemented as method *Test_SerNULL.test_init*.

Test result: PASS

Test Identifier: TEST-T-320

Requirement ID(s): REQ-FUN-320, REQ-FUN-328

Verification method: T

Test goal: Check the implementation of the additional API expected to be provided by a serializable fixed length array object.

Expected result: An instance of a serializable fixed length array can be passed as the argument of the Python built-in function *len*(), which will return the declared length of the array, regardless of the declared array element type and the length of the sequence argument of its instantiation.

Test steps: Perform the following tests

- Instantiate **BaseArray** class without an argument. Pass the instance into *len*() function. It should return value 2 (declared length).
- Instantiate **BaseArray** class with 2-elements, 1-element and 3-elements int lists. Check its length - must be 2 in all 3 cases.
- Instantiate **NestedArray** class without an argument. Pass the instance into *len*() function. It should return value 2 (declared length).
- Instantiate **NestedArray** class with 2-elements, 1-element and 3-elements lists, with the elements being {'a' : 1, 'b' : 1.0} dictionaries. Check its length - must be 2 in all 3 cases.
- Instantiate **ArrayArray** class without an argument. Pass the instance into *len*() function. It should return value 3 (declared length).

- Instantiate **ArrayArray** class with 3-elements, 2-element and 4-elements lists, with each element being a list [1, 1]. Check its length - must be 3 in all 3 cases.

Implemented as the method *Test_SerArray.test_Additional_API*().

Test result: PASS

Test Identifier: TEST-T-321

Requirement ID(s): REQ-FUN-321, REQ-FUN-327

Verification method: T

Test goal: Check that the elements are read accessible regardless of the declared type, whereas the compatible native Python type value can be assigned to the elements provided that the declared type is C primitive. Also check the performance of the instance method *getNative*().

Expected result:

- Native Python scalar type values are returned upon read access to the elements with the C primitive declared type
- An instance of the respective class (reference) is returned upon read access to the elements with the (nested) container declared type
- Compatible native Python scalar values can be assigned to the elements provided that the declared elements type is C primitive, and values are properly stored and can be read back
- The method *getNative*() returns a native Python list containing the entire stored data, with each element being native Python data type

Test steps: Perform the following tests

- Instantiate **BaseArray** without an argument as *objTest*
- Check that:
  - Length is 2
  - Both *objTest*[0] and *objTest*[1] are **int** and equal to 0
  - *getNative*() method returns [0, 0] list
- Make the following assignments
  - *objTest[-1]* = 3
  - *objTest[0]* = 2
- Check that:
  - Both *objTest*[0] and *objTest*[1] are **int** and equal to 2 and 3 respectively
  - *getNative*() method returns [2, 3] list
- Instantiate **NestedArray** with the [{'a' : 1}, {'b' : 1.0}] argument as *objTest*
- Check that:
  - Length is 2
  - Both *objTest*[0] and *objTest*[1] are **BaseStruct** instances
  - *getNative*() method returns [{'a' : 1, 'b' : 0.0}, {'a' : 0, 'b' : 1.0}] list
- Make assignment *objTest[0].b* = 2.0. Check that *objTest[0].b* is **float** and equal to 2.0
- Instantiate **ArrayArray** with the [[1, 1], [2, 2]] argument as *objTest*
- Check that:

    - Length is 3
    - Both *objTest*[0] and *objTest*[1] are **BaseArray** instances
    - *getNative*() method returns [[1, 1], [2, 2], [0, 0]] list
- Make assignment *objTest[0][0]* = 2. Check that *objTest[0][0]* is **int** and equal to 2

Implemented as the method *Test_SerDynamicArray.test_elements_access*()

Test result: PASS

Test Identifier: TEST-T-322

Requirement ID(s): REQ-FUN-322

Verification method: T

Test goal: Check the implementation of the initializer method - with and without passed argument.

Expected result: Without an argument the declared amount of the elements is created, and all these elements are of the declared type, but initiated with their default values. With an argument of the compatible data type passed, still the declared number of elements is created, and their are of the declared type, whereas the data from the passed argument is copied on per element basis when possible.

Test steps: Perform the following operations:

- Instantiate **BaseArray** without an argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [0, 0] list
- Instantiate **BaseArray** with the tuple (1, 2) argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2] list
- Instantiate **BaseArray** with the list [1] argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 0] list
- Instantiate **BaseArray** with the list [1, 2, 3] argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2] list
- Instantiate **BaseArray** with another instance **BaseArray**([1, 2]) argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2] list
- Instantiate **BaseArray** with an instance **BaseDynamicArray**([1, 2, 3]) argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2] list
- Instantiate **ArrayArray** without an argument
- Check that its length is 3, and index access returns **BaseArray** type value for all elements
- Check that the instance method *getNative*() returns [[0, 0], [0, 0], [0, 0]] list
- Instantiate **ArrayArray** with the tuple ([1, 2], [3]) argument
- Check that its length is 3, and index access returns **BaseArray** type value for all elements
- Check that the instance method *getNative*() returns [[1, 2], [3, 0], [0, 0]] list
- Instantiate **NestedArray** without an argument

- Check that its length is 2, and index access returns **BaseStruct** type value for all elements
- Check that the instance method *getNative*() returns [{'a' : 0, 'b' : 0.0}, {'a' : 0, 'b' : 0.0}] list
- Instantiate **NestedArray** with the tuple ({'a' : 1, 'b' : 2.0}, {'d' : 0, 'b' : 3.0}) argument
- Check that its length is 2, and index access returns **BaseStruct** type value for all elements
- Check that the instance method *getNative*() returns [{'a' : 1, 'b' : 2.0}, {'a' : 0, 'b' : 3.0}] list

Implemented as the method *Test_SerArray.test_instantiation*()

Test result: PASS

Test Identifier: TEST-T-323

Requirement ID(s): REQ-FUN-325

Verification method: T

Test goal: Check that the packing into JSON method returns a proper format string.

Expected result: The instance method *packJSON*() returns a string, which is JSON format representation of a list, containg as may elements as the array, and they are the JSON representation of the respective array elements converted into their native Python form.

Test steps: Perform the following operations:

- Instantiate **BaseArray** class without an argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [0, 0]
- Instantiate **BaseArray** class with the [1, 2, 3] argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [1, 2]
- Instantiate **NestedArray** class without an argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [{'a' : 0, 'b' : 0.0}, {'a' : 0, 'b' : 0.0}]
- Instantiate **NestedArray** class with the [{'a' : 1, 'b' : 2.0}, {'a' : 3, 'b' : 4.0}, {'a' : 5, 'b' : 6.0}] argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [{'a' : 1, 'b' : 2.0}, {'a' : 3, 'b' : 4.0}]
- Instantiate **ArrayArray** class without an argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [[0, 0], [0, 0], [0, 0]]
- Instantiate **ArrayArray** class with the [[1], [2, 3]] argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [[1, 0], [2, 3], [0, 0]]

Implemented as the method *Test_SerArray.test_packJSON*()

Test result: PASS

Test Identifier: TEST-T-324

Requirement ID(s): REQ-FUN-326

Verification method: T

Test goal: Check that the implementation of de-serialization from a JSON string.

Expected result: A new instance of the class can be created if the passed JSON string encodes a list with the same number of elements as the declared length of the array, and all values in that list are compatible with the declared type of the array elements.

Test steps: Perform the following operations:

- Pass string '[1, 2]' into the class method *BaseArray.unpackJSON*()
- Check that an instance of **BaseArray** is created
- Check that its instance method *getNative*() returns [1, 2] list
- Pass string '[{"a" : 1, "b" : 2.0}, {"a" : 3, "b" : 4.0}]' into the class method *NestedArray.unpackJSON*()
- Check that an instance of **NestedArray** is created
- Check that its instance method *getNative*() returns [{'a' : 1, 'b' : 2.0}, {'a' : 3, 'b' : 4.0}] list
- Pass string '[[1, 2], [3, 4], [5, 6]]' into the class method *ArrayArray.unpackJSON*()
- Check that an instance of **ArrayArray** is created
- Check that its instance method *getNative*() returns [[1, 2], [3, 4], [5, 6]] list

Implemented as the method *Test_SerArray.test_unpackJSON*()

Test result: PASS

Test Identifier: TEST-T-330

Requirement ID(s): REQ-FUN-330, REQ-FUN-338

Verification method: T

Test goal: Check the implementation of the additional API expected to be provided by a serializable dynamic length array object.

Expected result: The length of an instance of the dynamic length array can be obtained using the standard Python built-in function *len*(), and the returned result is determined by the lengt of the sequence (or array) object passed as the argument during the instantiantion of the dynamic array. The class method *getElementSize*() returns the size in bytes of the declared type of the elements.

Test steps: Perform the following tests

- Check the **BaseDynamicArray** class has the class method *getElementSize*().
- Instantiate **BaseDynamicArray** class without an argument. Pass the instance into *len*() function. It should return value 0.

- Check the element size (method *getElementSize*() of the instance) - must be 2.
- Instantiate **BaseDynamicArray** class with 2-elements, 1-element and 3-elements int lists. Check its length - must be 2, 1 and 3 respectively.
- Check the element size (method *getElementSize*() of the instance) - must be 2 in all 3 cases.
- Check the **NestedDynamicArray** class has the class method *getElementSize*().
- Instantiate **NestedDynamicArray** class without an argument. Pass the instance into *len*() function. It should return value 0.
- Check the element size (method *getElementSize*() of the instance) - must be 6.
- Instantiate **NestedDynamicArray** class with 2-elements, 1-element and 3-elements lists, with the elements being {'a' : 1, 'b' : 1.0} dictionaries. Check its length - must be 2, 1 and 3 respectively.
- Check the element size (method *getElementSize*() of the instance) - must be 6 in all 3 cases.
- Check the **DynamicArrayArray** class has the class method *getElementSize*().
- Instantiate **DynamicArrayArray** class without an argument. Pass the instance into *len*() function. It should return value 0.
- Check the element size (method *getElementSize*() of the instance) - must be 4.
- Instantiate **DynamicArrayArray** class with 3-elements, 2-element and 4-elements lists, with each element being a list [1, 1]. Check its length - must be 3, 2 and 4 respectively.
- Check the element size (method *getElementSize*() of the instance) - must be 4 in all 3 cases.

Implemented as the method *Test_SerDynamicArray.test_Additional_API*().

Test result: PASS

Test Identifier: TEST-T-331

Requirement ID(s): REQ-FUN-331, REQ-FUN-337

Verification method: T

Test goal: Check that the elements are read accessible regardless of the declared type, whereas the compatible native Python type value can be assigned to the elements provided that the declared type is C primitive. Also check the performance of the instance method *getNative*().

Expected result:

- Native Python scalar type values are returned upon read access to the elements with the C primitive declared type
- An instance of the respective class (reference) is returned upon read access to the elements with the (nested) container declared type
- Compatible native Python scalar values can be assigned to the elements provided that the declared elements type is C primitive, and values are properly stored and can be read back
- The method *getNative*() returns a native Python list containing the entire stored data, with each element being native Python data type

Test steps: Perform the following tests

- Instantiate **BaseDynamicArray** without an argument as *objTest*
- Check that *getNative*() method returns an empty list
- Instantiate **BaseDynamicArray** with the [1, 1] argument as *objTest*

- Check that:
  - Length is 2
  - Both *objTest*[0] and *objTest*[1] are **int** and equal to 1
  - *getNative*() method returns [1, 1] list
- Make the following assignments
  - *objTest[-1]* = 3
  - *objTest[0]* = 2
- Check that:
  - Both *objTest*[0] and *objTest*[1] are **int** and equal to 2 and 3 respectively
  - *getNative*() method returns [2, 3] list
- Instantiate **NestedDynamicArray** with the [{'a' : 1}, {'b' : 1.0}] argument as *objTest*
- Check that:
  - Length is 2
  - Both *objTest*[0] and *objTest*[1] are **BaseStruct** instances
  - *getNative*() method returns [{'a' : 1, 'b' : 0.0}, {'a' : 0, 'b' : 1.0}] list
- Make assignment *objTest[0].b* = 2.0. Check that *objTest[0].b* is **float** and equal to 2.0
- Instantiate **DynamicArrayArray** with the [[1, 1], [2, 2]] argument as *objTest*
- Check that:
  - Length is 2
  - Both *objTest*[0] and *objTest*[1] are **BaseArray** instances
  - *getNative*() method returns [[1, 1], [2, 2]] list
- Make assignment *objTest[0][0]* = 2. Check that *objTest[0][0]* is **int** and equal to 2

Implemented as the method *Test_SerDynamicArray.test_elements_access*()

Test result: PASS

Test Identifier: TEST-T-332

Requirement ID(s): REQ-FUN-332

Verification method: T

Test goal: Check the implementation of the initializer method - with and without passed argument.

Expected result: Without an argument an empty array is created. With an argument of the compatible data type passed, a number of elements of the declared type is created, such that the length of the array equals to the length of the passed sequence / array; and all created elements are initiated using the data stored in the respective element of the passed argument.

Test steps: Perform the following operations:

- Instantiate **BaseDynamicArray** without an argument
- Check that its length is 0
- Check that the instance method *getNative*() returns an empty list
- Instantiate **BaseDynamicArray** with the tuple (1, 2) argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2] list
- Instantiate **BaseDynamicArray** with the list [1] argument

- Check that its length is 1, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1] list
- Instantiate **BaseDynamicArray** with the list [1, 2, 3] argument
- Check that its length is 3, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2, 3] list
- Instantiate **BaseDynamicArray** with an instance **BaseArray**([1, 2]) argument
- Check that its length is 2, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2] list
- Instantiate **BaseDynamicArray** with another instance **BaseDynamicArray**([1, 2, 3]) argument
- Check that its length is 3, and index access returns **int** type value for all elements
- Check that the instance method *getNative*() returns [1, 2, 3] list
- Instantiate **DynamicArrayArray** without an argument
- Check that its length is 0
- Check that the instance method *getNative*() returns an empty list
- Instantiate **DynamicArrayArray** with the tuple ([1, 2], [3]) argument
- Check that its length is 2, and index access returns **BaseArray** type value for all elements
- Check that the instance method *getNative*() returns [[1, 2], [3, 0]] list
- Instantiate **NestedDynamicArray** without an argument
- Check that its length is 0
- Check that the instance method *getNative*() returns am empty list
- Instantiate **NestedDynamicArray** with the tuple ({'a' : 1, 'b' : 2.0}, {'d' : 0, 'b' : 3.0}) argument
- Check that its length is 2, and index access returns **BaseStruct** type value for all elements
- Check that the instance method *getNative*() returns [{'a' : 1, 'b' : 2.0}, {'a' : 0, 'b' : 3.0}] list

Implemented as the method *Test_SerDynamicArray.test_instantiation*()

Test result: PASS

Test Identifier: TEST-T-333

Requirement ID(s): REQ-FUN-335

Verification method: T

Test goal: Check that the packing into JSON method returns a proper format string.

Expected result: The instance method *packJSON*() returns a string, which is JSON format representation of a list, containg as may elements as the array, and they are the JSON representation of the respective array elements converted into their native Python form.

Test steps: Perform the following operations:

- Instantiate **BaseDynamicArray** class without an argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be an empty native Python list []
- Instantiate **BaseDynamicArray** class with the [1, 2, 3] argument
- Check that the method *packJSON*() returns a string

- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [1, 2, 3]
- Instantiate **NestedDynamicArray** class without an argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be an empty native Python list []
- Instantiate **NestedDynamicArray** class with the [{'a' : 1, 'b' : 2.0}, {'a' : 3, 'b' : 4.0}] argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [{'a' : 1, 'b' : 2.0}, {'a' : 3, 'b' : 4.0}]
- Instantiate **DynamicArrayArray** class without an argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be an empty native Python list []
- Instantiate **DynamicArrayArray** class with the [[1, 0], [2, 3], [0, 0]] argument
- Check that the method *packJSON*() returns a string
- Pass that string into the Standard Library function *json.loads*() - the returned value must be a native Python list [[1, 0], [2, 3], [0, 0]]

Implemented as the method *Test_SerDynamicArray.test_packJSON*()

Test result: PASS

Test Identifier: TEST-T-334

Requirement ID(s): REQ-FUN-336

Verification method: T

Test goal: Check that the implementation of de-serialization from a JSON string.

Expected result: A new instance of the class can be created if the passed JSON string encodes a list with all values wherein being compatible with the declared type of the array elements, and the created array has as many elements, as the passed JSON list.

Test steps: Perform the following operations:

- Pass string '[1, 2]' into the class method *BaseDynamicArray.unpackJSON*()
- Check that an instance of **BaseDynamicArray** is created
- Check that its instance method *getNative*() returns [1, 2] list
- Pass string '[{"a" : 1, "b" : 2.0}, {"a" : 3, "b" : 4.0}]' into the class method *NestedDynamicArray.unpackJSON*()
- Check that an instance of **NestedDynamicArray** is created
- Check that its instance method *getNative*() returns [{'a' : 1, 'b' : 2.0}, {'a' : 3, 'b' : 4.0}] list
- Pass string '[[1, 2], [3, 4], [5, 6]]' into the class method *DynamicArrayArray.unpackJSON*()
- Check that an instance of **DynamicArrayArray** is created
- Check that its instance method *getNative*() returns [[1, 2], [3, 4], [5, 6]] list
- Pass JSON string '[]' into the class method *unpackJSON*() of the same three classes.
- Check that the instances of the classes are returned, but their length is zero.

Implemented as the method *Test_SerDynamicArray.test_unpackJSON*()

Test result: PASS

Test Identifier: TEST-T-340

Requirement ID(s): REQ-FUN-340, REQ-FUN-348

Verification method: T

Test goal: Check the implementation of the additional API expected to be provided by a struct object.

Expected result: The class has a class method *getMinSize*(), which returns a size in bytes of all fixed size fields. It also proviced instance method *getCurrentSize*(), which returns the current size in bytes of all elements, including the last field if it is a dynamic length object.

Test steps: Perform the following tests

- Check that the **BaseStruct** class has the class method *getMinSize*().
- Check that the **BaseStruct** class has the instance method *getCurrentSize*().
- Instantiate **BaseStruct** class without an argument. Both mentioned methods (on instance) must return 6.
- Instantiate the same class with the following arguments {'a' : 1}, {'b': 1.0}, {'a' : 1, 'b' : 1.0, 'c': 1}. Both mentioned methods (on instance) must return 6 in all 3 cases.
- Check that the **NestedStruct** class has the class method *getMinSize*().
- Check that the **NestedStruct** class has the instance method *getCurrentSize*().
- Instantiate **NestedStruct** class without an argument. Both mentioned methods (on instance) must return 10.
- Instantiate the same class with the following arguments {'a' : 1}, {'b': 1.0}, {'a' : 1, 'b' : 1.0, 'c': [1], 'd': 1}. Both mentioned methods (on instance) must return 10 in all 3 cases.
- Check that the **NestedDynamicStruct** class has the class method *getMinSize*().
- Check that the **NestedDynamicStruct** class has the instance method *getCurrentSize*().
- Instantiate **NestedStruct** class without an argument. Both mentioned methods (on instance) must return 6.
- Instantiate the same class with the following arguments {'a' : 1}, {'b': 1.0}. Both mentioned methods (on instance) must return 6 in the both cases.
- Instantiate the same class with the following argument {'a' : 1, 'b': 1.0, 'c' : [1], 'd' : 1.0}. The method *getMinSize*() must return 6, whereas the method *getCurrentSize*() must return 8.
- Check that the **ComplexStruct** class has the class method *getMinSize*().
- Check that the **ComplexStruct** class has the instance method *getCurrentSize*().
- Instantiate **ComplexStruct** class without an argument. Both mentioned methods (on instance) must return 12.
- Instantiate the same class with the following arguments {'a' : 1}, {'b': 1.0}. Both mentioned methods (on instance) must return 12 in the both cases.
- Instantiate the same class with the following argument {'a' : 1, 'b': 1.0, 'c' : {'c' : [1], 'd' : 1.0}, 'd' : 1.0}. The method *getMinSize*() must return 12, whereas the method *getCurrentSize*() must return 14.

Implemented as the method *Test_SerStruct.test_Additional_API*().

Test result: PASS

Test Identifier: TEST-T-341

Requirement ID(s): REQ-FUN-341, REQ-FUN-347

Verification method: T

Test goal: Check that all declared fields are read accessible, whereas the fields declared as C primitives can also be assigned to with compatible native Python type value. Also check the performance of the instance method *getNative*().

Expected result:

- Read access to the C primitive declared fields return a native Python scalar value, which should be of the expected value
- Read access to a nested container field returns an instance of the respective class (reference to), which fields / elements should hold the expected values
- Declared C primitive fields can be assigned to with compatible native Python scalar values, which values are properly stored and can be read back
- The method *getNative*() returns a native Python dictionary with the keys names corresponding to the declared fields, and the bound values being native Python types corresponding to the respective stored data

Test steps: Perform the following tests

- Instantiate **ComplexStruct** with the {'c' : {'c' : [1,1]}} argument as *objTest*
- Check that:
  - *objTest.a* is **int**** equal to 0
  - *objTest.b* is **float** equal to 0.0
  - *objTest.c* is an instance of **NestedDynamicStruct**
  - *objTest.c.a* is **int**** equal to 0
  - *objTest.c.b* is **float** equal to 0.0
  - *objTest.c.c* is an instance of **BaseDynamicArray**
- Check that the method *getNative*() returns a dictionary {'a' : 0, 'b' : 0.0, 'c' : {'a' : 0, 'b' : 0.0, 'c' : [1, 1]}}
- Make the following assignments
  - *objTest.a* = 1
  - *objTest.b* = 2.0
  - *objTest.c.a* = 2
  - *objTest.c.b* = 3.0
  - *objTest.c.c[0]* = 3
- Check that:
  - *objTest.a* is **int**** equal to 1
  - *objTest.b* is **float** equal to 2.0
  - *objTest.c* is an instance of **NestedDynamicStruct**
  - *objTest.c.a* is **int**** equal to 2
  - *objTest.c.b* is **float** equal to 3.0
  - *objTest.c.c* is an instance of **BaseDynamicArray**

- Check that the method *getNative*() returns a dictionary {'a' : 1, 'b' : 2.0, 'c' : {'a' : 2, 'b' : 3.0, 'c' : [3, 1]}}
- Instantiate **ComplexStruct** without an argument, call method *getNative*() on the instance, and check that it returns a dictionary {'a' : 0, 'b' : 0.0, 'c' : {'a' : 0, 'b' : 0.0, 'c' : []}}

Implemented as the method *Test_SerStruct.test_attribute_access*()

Test result: PASS

Test Identifier: TEST-T-342

Requirement ID(s): REQ-FUN-342

Verification method: T

Test goal: Check the implementation of the initializer method - with and without passed argument.

Expected result: Without an argument the default values are assigned to the end nodes (C primitive type fields / elements). With the passed argument of a mapping type or another instance of any struct sub-class, only the values of the matching keys / fields are copied, for other fields the default values are used - this rule is applied recursively to the nested containers.

Test steps: Perform the following tests

- Instantiate **ComplexStruct** without an argument as *objTest*
- Check that:
  - *objTest.a* is **int**** equal to 0
  - *objTest.b* is **float** equal to 0.0
  - *objTest.c* is an instance of **NestedDynamicStruct**
  - *objTest.c.a* is **int**** equal to 0
  - *objTest.c.b* is **float** equal to 0.0
  - *objTest.c.c* is an instance of **BaseDynamicArray** of length 0.
- Delete this instance
- Instantiate **ComplexStruct** with the {'b' : 1.0, 'd' : 1, 'c' : {'a' : 1, 'c' : [1, 1], 'd' : 1}} argument as *objTest*
- Check that:
  - *objTest.a* is **int**** equal to 0
  - *objTest.b* is **float** equal to 1.0
  - *objTest.d* does not exist
  - *objTest.c* is an instance of **NestedDynamicStruct**
  - *objTest.c.a* is **int**** equal to 1
  - *objTest.c.b* is **float** equal to 0.0
  - *objTest.c.c* is an instance of **BaseDynamicArray** of length 2; and its both elements values are 1 (**int**)
  - *objTest.c.d* does not exist
- Instantiate **ComplexStruct** with *objTest* as the argument -> second instance *objNewTest*
- Check that:
  - *objNewTest.a* is **int**** equal to 0
  - *objNewTest.b* is **float** equal to 1.0
  - *objNewTest.c* is an instance of **NestedDynamicStruct**
  - *objNewTest.c.a* is **int**** equal to 1

    - *objNewTest.c.b* is **float** equal to 0.0
    - *objNewTest.c.c* is an instance of **BaseDynamicArray** of length 2; and its both elements values are 1 (**int**)
- Delete both instances
- Instantiate **BaseStruct** with the argument {'a' : 1, 'b': 1.0} as *objTest*
- Instantiate **ComplexStruct** with *objTest* as the argument -> second instance *objNewTest*
- Check that:
    - *objNewTest.a* is **int**** equal to 1
    - *objNewTest.b* is **float** equal to 1.0
    - *objNewTest.c* is an instance of **NestedDynamicStruct**
    - *objNewTest.c.a* is **int**** equal to 0
    - *objNewTest.c.b* is **float** equal to 0.0
    - *objNewTest.c.c* is an instance of **BaseDynamicArray** of length 0
- Delete *objTest* instance
- Instantiate **BaseStruct** with the argument *objNewTest* as *objTest*
- Check that:
    - *objTest.a* is **int**** equal to 1
    - *objTest.b* is **float** equal to 1.0
    - *objTest.c* does not exist
- Delete both instances

Implemented as the method *Test_SerStruct.test_instantiation*()

Test result: PASS

Test Identifier: TEST-T-343

Requirement ID(s): REQ-FUN-345

Verification method: T

Test goal: Check that the packing into JSON method returns a proper format string.

Expected result: The instance method *packJSON*() returns a string, which contains the JSON representation of a dictionary, which reflects the entire stored data.

Test steps: Perform the following operations:

- Instantiate **ComplexStruct** without an argument.
- Call its method *packJSON*()
- Pass the returned value into *json.loads*() function
- Check that the funtion returns a dictionary {'a' : 0, 'b' : 0.0, 'c' : {'a' : 0, 'b' : 0.0, 'c' : []}}
- Instantiate **ComplexStruct** with the {'a' : 1, 'b' : 2.0, 'c' : {'a' : 3, 'b' : 4.0, 'c' : [1, 1]}} argument.
- Call its method *packJSON*()
- Pass the returned value into *json.loads*() function
- Check that the funtion returns a dictionary {'a' : 1, 'b' : 2.0, 'c' : {'a' : 3, 'b' : 4.0, 'c' : [1, 1]}}
- No exceptions should be raised

Implemented as the method *Test_SerStruct.test_packJSON*()

Test result: PASS

Test Identifier: TEST-T-344

Requirement ID(s): REQ-FUN-346

Verification method: T

Test goal: Check that the implementation of de-serialization from a JSON string.

Expected result: A new instance of the class can be created if the passed JSON string encodes a dictionary with all keys being the declared fields names, and for each declared field there is a dictionary key, which holds a value compatible with the declared type of that field.

Test steps: Perform the following operations:

- Pass '{"a" : 1, "b" : 2.0, "c" : {"a" : 3, "b": 4.0, "c" : []}}' into the class method *ComplexStruct.unpackJSON*()
- Check that an instance of **ComplexStruct** is created
- Check that the instance method *getNative* returns {'a' : 1, 'b' : 2.0, 'c' : {'a' : 3, 'b' : 4.0, 'c' : []}} dictionary
- Pass '{"a" : 1, "b" : 2.0, "c" : {"a" : 3, "b": 4.0, "c" : [5, 6]}} into the class method *ComplexStruct.unpackJSON*()
- Check that an instance of **ComplexStruct** is created
- Check that the instance method *getNative* returns {'a' : 1, 'b' : 2.0, 'c' : {'a' : 3, 'b' : 4.0, 'c' : [5, 6]}} dictionary

Implemented as the method *Test_SerStruct.test_unpackJSON*()

Test result: PASS

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-303	TEST-T-309	YES
REQ-FUN-310	TEST-T-310	YES
REQ-FUN-311	TEST-T-311	YES
REQ-FUN-320	TEST-T-305, TEST-T-320	YES
REQ-FUN-321	TEST-T-321	YES
REQ-FUN-322	TEST-T-322	YES

Requirement ID	Covered in test(s)	Verified [YES/NO])
REQ-FUN-323	TEST-T-309	YES
REQ-FUN-324	TEST-T-309	YES
REQ-FUN-325	TEST-T-323	YES
REQ-FUN-326	TEST-T-324	YES
REQ-FUN-327	TEST-T-321	YES
REQ-FUN-328	TEST-T-320	YES
REQ-FUN-330	TEST-T-305, TEST-T-330	YES
REQ-FUN-331	TEST-T-331	YES
REQ-FUN-332	TEST-T-332	YES
REQ-FUN-333	TEST-T-309	YES
REQ-FUN-334	TEST-T-309	YES
REQ-FUN-335	TEST-T-333	YES
REQ-FUN-336	TEST-T-334	YES
REQ-FUN-337	TEST-T-331	YES
REQ-FUN-338	TEST-T-330	YES
REQ-FUN-340	TEST-T-305, TEST-T-340	YES
REQ-FUN-341	TEST-T-341	YES
REQ-FUN-342	TEST-T-342	YES
REQ-FUN-343	TEST-T-309	YES
REQ-FUN-344	TEST-T-309	YES
REQ-FUN-345	TEST-T-343	YES
REQ-FUN-346	TEST-T-344	YES
REQ-FUN-347	TEST-T-341	YES
REQ-FUN-348	TEST-T-340	YES
REQ-AWM-300	TEST-T-305	YES
REQ-AWM-301	TEST-T-304	YES
REQ-AWM-302	TEST-T-308	YES
REQ-AWM-303	TEST-T-302	YES
REQ-AWM-304	TEST-T-303	YES
REQ-AWM-305	TEST-T-301	YES

Requirement ID	Covered in test(s)	Verified [YES/NO])
REQ-AWM-306	TEST-T-306	YES
REQ-AWM-307	TEST-T-307	YES

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