```
class _Base1DTaurusExecutor(object):

    class IterativeEnum(Enum):..

    PRINT_STEP_RESULT = True
    ITERATIVE_METHOD  = None
    SAVE_DAT_FILES = []  # list for saving the auxillary files from taurus
    EXPORT_LIST_RESULTS = 'export_resultTaurus.txt'
    HEADER_SEPARATOR = OUTPUT_HEADER_SEPARATOR

    TRACK_TIME_AND_RAM = False
    TRACK_TIME_FILE = '_time_program.log'

    CONSTRAINT    : str = None # InputTaurus Variable to compute
    CONSTRAINT_DT : str = None # DataTaurus (key) Variable to compute

    DTYPE = DataTaurus  # DataType for the outputs to manage
    ITYPE = InputTaurus # Input type for the input management

    SEEDS_RANDOMIZATION = 5 # Number of random seeds for even-even calculation /
                            # ALSO: Number of blocking sp state for odd calculation
    GENERATE_RANDOM_SEEDS = False
    # @classmethod..

    def setUp(self, *args, **kwargs):..

    def __init__(self, z, n, interaction, *args, **kwargs):..

    def resetExecutorObject(self, keep_1stMinimum=False):..

    @property
    def numberParityOfIsotope(self):..

    def _checkExecutorSettings(self):..

    def setInputCalculationArguments(self, core_calc=False, axial_calc=False,..

    def setUpExecution(self, *args, **kwargs):..

    def defineDeformationRange(self, min_, max_, N_steps):..

    def _setDeformationFromMinimum(self, p_min, p_max, N_max):..


    def _runUntilConvergence(self, MAX_STEPS=3):..

    def run(self):..

    def _run_backwardsSweeping(self, oblate_part=None):..

    def _backPropagationAcceptanceCriteria(self, result, prev_result):..

    def _energyDiffRejectionCriteria(self, curr_energ,  old_energ, old_e_diff, ..

    def _runVariableStep(self):..

    def _auxWindows_executeProgram(self, output_fn):..

    def _namingFilesToSaveInTheBUfolder(self):..
```

Line: 546

```
    def saveFinalWFprocedure(self, base_execution=False):..

    def _getMinimumIterationTimeTaurus(self):..

    def _executeProgram(self, base_execution=False):..

    def _addExecutionPerformanceData(self, result: DataTaurus):..

    def printTaurusResult(self, result : DataTaurus, print_head=False, ..


    @property
    def calculationParameters(self):..

    def executionTearDown(self, result : DataTaurus, base_execution, *args, **kwargs):..


    def exportResults(self, output_filename=None):..


    def gobalTearDown(self, *args, **kwargs):..


class _Base1DAxialExecutor(_Base1DTaurusExecutor):

    EXPORT_LIST_RESULTS = 'export_resultAxial.txt'

    DTYPE = DataAxial  # DataType for the outputs to manage
    ITYPE = InputAxial # Input type for the input management
```

```
class ExeTaurus1D_DeformQ20(_Base1DTaurusExecutor):

    ITERATIVE_METHOD = _Base1DTaurusExecutor.IterativeEnum.EVEN_STEP_SWEEPING

    CONSTRAINT    = InputTaurus.ConstrEnum.b20
    CONSTRAINT_DT = DataTaurus.getDataVariable(InputTaurus.ConstrEnum.b20,
                                               beta_schm = 0)

    EXPORT_LIST_RESULTS = 'export_TESq20'

    def setUp(self):...


    def setUpExecution(self, reset_seed=False,  *args, **kwargs):...

    def _getStatesAndDimensionsOfHamiltonian(self):...

    def _oddNumberParitySeedConvergence(self):...

    def _evenNumberParitySeedConvergence(self):...

    def _exportBaseResultFile(self, bu_results):...

    def _preconvergenceAccepted(self, result: DataTaurus):...

    def saveFinalWFprocedure(self, base_execution=False):...

    def run(self):...

    def gobalTearDown(self, zip_bufolder=True, *args, **kwargs):...


class ExeAxial1D_DeformQ20(ExeTaurus1D_DeformQ20):

    CONSTRAINT    = InputAxial.ConstrEnum.b20
    CONSTRAINT_DT = DataAxial .getDataVariable(InputAxial.ConstrEnum.b20,
                                               beta_schm = 0)
    EXPORT_LIST_RESULTS = 'exportAx_TESq20'

    EXPORT_LIST_RESULTS = 'export_resultAxial.txt'

    DTYPE = DataAxial  # DataType for the outputs to manage
    ITYPE = InputAxial # Input type for the input management



class ExeTaurus1D_DeformB20(ExeTaurus1D_DeformQ20):...

class ExeAxial1D_DeformB20(ExeTaurus1D_DeformB20):

    CONSTRAINT    = InputAxial.ConstrEnum.b20
    CONSTRAINT_DT = DataAxial .getDataVariable(InputAxial.ConstrEnum.b20,
                                               beta_schm = 1)

    EXPORT_LIST_RESULTS = 'export_resultAxial.txt'

    DTYPE = DataAxial  # DataType for the outputs to manage
    ITYPE = InputAxial # Input type for the input management
```