This document presents the different classes and their usage of the Network eXperiment Engine software.

1 NXE

This is the entry point of the NXE software, providing the glue between all the classes. It takes exactly one argument, the name of a XML file corresponding to the description of the experiment that should be run.

1.1 Variables

In this section, we present the variables that are globally defined in the software.

1.2 Functions

- parsing_config(filename)
- handling_reservation(siteList, frontalList, nodeList, date)
- handling_deployment(siteList, frontalList)
- ip_instanciation(nodeList, nodeIPAssociation)
- linking_profile_steps(nodeList, profileStepList)
- pairing_nodes(nodeList)
- handling_BDTS(nodeList, BDTSprogram)
- preparing_schedule(nodeList, threadList, timeout, file, datelabel)
- running(threadList)
- retrieving_logs(nodeList, archivefolder, timelinelog)
- syncing_scripts(nodeList, scriptpath, user)
- generating_stats(siteList, datelabel, applicationtype)
- cleaning(siteList, frontalList, nodeList)
- creating_folders(archivepath)

Name	Description	Type	Default
currentState	Indicate at which stage of the run we currently are	Enum	
pid	Pid of the current instance of the NXE software	Integer	-
spoofed	Indicate that we are spoofing the pid of different NXE instance	Boolean	False
tmpfolder	Indicates where the temp files will be copied	String	
archivefolder	Indicates where the results will be saved	String	
errorStream	Pointer to the error stream used	File	stderr
outputStream	Pointer to the output stream used	File	stdout
config	contains the global configuration of the application	NXEParser.ConfigParser	-
scenario	contains the experiment scenario	NXEParser.ScenarioParser	-
topology	contains the description of the topology	NXEParser.TopologyParser	-
profile	contains the description of the bandwidth profile	NXEParser.BDTSProfileParser	-

Table 1: Global variables used in the NXE software

2 NXEParser

This module contains all the classes responsible for the parsing of the XML input file. It is based on the Python XML Expat library and it is using a SAX model. A good addition to this section is the document that presents the XML format that is used to describe the scenarios.

2.1 TopologyParser

This class is responsible for the topology part of the scenario.

2.2 BDTSProfileParser

2.3 ConfigurationParser

2.4 ScenarioParser

3 NXEUtils

A module that contains the miscellanous stuff that wasn't fit to be in a given particular class. Mainly we have utility functions used for the repetitive tasks of handling dates and time or string manipulation/

4 NXETypes

A module that contains the definition of the business object model that are used to represent the objects needed for the run of an experiment.

4.1 Node

This class contains all the information necessary to contact a given node and the needed information to execute the scenario steps relevant to that node. It is instanced during the parsing of the input file.

We are also storing the pointer to the SSH connexion unto the actual node, so as to avoid the overhead of opening a new one everytime we need to issue a new command to the node. SSH channels are used to execute the command.

A recent addition is the use of two NXERandom objects to represent the distribution law used for the sizes of the transferts and the interwait times between two transferts (but it could be used for almost anything else that require two sets of random numbers)

4.2 execStep

This class represents a step of a scenario to execute. As we are using a command pattern, so we are storing a date at which the execution step should start and the command and its parameters that should be executed

A label is also assigned to the execStep to have a string to describe the effect of the execution step *e.g.* the particular value of the parameter we are evaluating in the scenatio.

4.3 profileStep

This class describes a step of a bandwidth profile. This is no longer used as we are only storing a string to keep the bandwidth profile.

4.4 topologySite

This class is the equivalent of the topology used inside the configuration file. It is used to describe a site (*e.g.* a cluster) containing resources (*e.g.* computation nodes) that can be used during an networking experiment.

4.5 topologyFrontal

This class is associated with the topologySite and contains all the necessary information needed to contact the resource management services such as the kernel image deployment software (*e.g.* kadeploy in Grid'5000) and the node reservation software (*e.g.* OAR in Grid'5000).

4.6 topologyAggregator

This class describes the interaction in terms of network connectivity between the different sites involved in the experiment. It introduces a virtual point (or physical point as an aggregator can be a router or a switch somewhere in the network) where the flows send by the end hosts are gathered.

It is not currently an essential part of the NXE software but it should be used to help selecting groups of sites that need to be taken simultaneously into consideration when generating the statistics and graphs from an experiment.

4.7 topologyLink

This class represents a (virtual or physical) link that exists between a site and an aggregator or between two aggregators.

5 NXERandom

This class is used to store and generate random numbers following a given distribution lw. The distribution laws available depends on the implementation of the random module in Python. The only exception is the Pareto distribution which doesn't use the *paretovariate* function as it doesn't allow to choose the mean value.

A fixed set of numbers can also be provided in the configuration file so as to reproduce a test-case that had been previously executed in another environment.

Every time a random number is generated in an NXERandom Object, the value is stored in a table. It can be accessed at any stage of the execution for generating statistics or saving these values for a later use.

6 NXEStats

This module contains a loose set of functions that are related to the computation of statistical metrics. There are mainly classical functions (*e.g.* mean, standard deviation) that were written to take a set of values as an argument.

7 NXEChart

This module is responsible for the generation of the figures using gnuplot and the Python Gnuplot interface. It takes the output of the statistical part of the NXE software and produces a figure.

Currently, only one type of figure is available: the normalized histogram that provides an easy-to-read figure for users.

This script can also be used as a stand-alone to (re)generate figures from values read from a file.

8 NXEThread

This module is used to handle the different types of threads that are used to execute simultaneous tasks in the NXE software.

8.1 NXEThreadCommand

This class corresponding to a simple thread is used to issue simple commands on distant nodes. It uses the Python paramiko SSH module to run a remote command provided a string. Both stdout and stderr are monitored but currently we are only looking for a "KO!" or "OK!" string on the first line of the stdout output.

Name	Description		
NXE			

Table 2: Description of the classes of the NXE software

It means that the output from the commands (*e.g.* bash scripts) should be relatively quiet. We need that because there is no other simple way to wait for the end of the execution of the remote command. Failure to comply to with this will lead to premature termination of the thread. In this case, it will be impossible to ensure that all the commands performed on the remote hosts will be performed in the right order.

8.2 NXEThreadTimer

This class reproduces a Timer class. It is using the dates provided in an execStepList to start the execution of a command at the required time and wait for the date corresponding to the next event to occur in the list.

The granularity of this mechanism is the second. But as Python timers are precise up to the millisecond, it could be possible to extent this class to be more precise.

The rest of the thread code is very similar with the one used in the NXEThreadCommand class except that a part of the command string used is generated there instead of being just a parameter. It breaks a little bit the isolation between the different layers of the application but we are forced to do so as some of the script parameters depends on the current execution step we are in.