Reference Manual

Generated by Doxygen 1.8.6

Tue Feb 16 2016 02:42:00

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

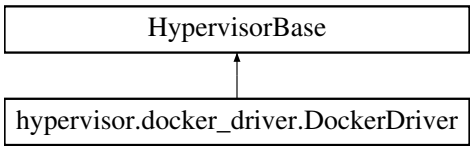
Chapter 3

Class Documentation

3.1 hypervisor.docker_driver.DockerDriver Class Reference

docker driver for nfio.

Inheritance diagram for hypervisor.docker_driver.DockerDriver:



Public Member Functions

- def **__init__**
- def get_id

  Returns a container's ID.
- def get_ip

  Returns a container's IP address.
- def deploy

  Deploys a docker container.
- def start

  Starts a docker container.
- def restart

  Restarts a docker container.
- def stop

  Stops a docker container.
- def pause

  Pauses a docker container.
- def unpause

  Unpauses a docker container.
- def destroy

  Destroys a docker container.
- def execute_in_guest

  Executed commands inside a docker container.
- def guest_status

  Returns the status of a docker container.

3.1.1 Detailed Description

docker driver for nfio.

This class provides methods for managing docker containers.

Definition at line 20 of file docker_driver.py.

3.1.2 Member Function Documentation

3.1.2.1 def hypervisor.docker_driver.DockerDriver.deploy (*self, host, user, image_name, vnf_name, is_privileged =* `True`)

Deploys a docker container.

Parameters

host	IP address or hostname of the machine where the docker container is to be deployed
user	name of the user who owns the VNF
image_name	docker image name for the VNF
vnf_name	name of the VNF instance
is_privileged	if True then the container is deployed in privileged mode

Returns

docker container ID

Definition at line 166 of file docker_driver.py.

```
166
167     def deploy(self, host, user, image_name, vnf_name, is_privileged=True):
168         self._validate_host(host)
169         self._validate_image_name(image_name)
170         vnf_fullname = user + '-' + vnf_name
171         self._validate_cont_name(vnf_fullname)
172         dcx = self._get_client(host)
173         host_config = dict()
174         if is_privileged:
175             host_config['Privileged'] = True
176         with self._error_handling(errors.VNFDeployError):
177             container = dcx.create_container(
178                 image=image_name,
179                 hostname=host,
180                 name=vnf_fullname,
181                 host_config=host_config)
182             return container['Id']
```

3.1.2.2 def hypervisor.docker_driver.DockerDriver.destroy (*self, host, user, vnf_name, force =* `True`)

Destroys a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF
force	if set to False then a running VNF will not be destroyed. default is True

Definition at line 267 of file docker_driver.py.

```
267
268     def destroy(self, host, user, vnf_name, force=True):
269         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
270         with self._error_handling(errors.VNFDestroyError):
271             dcx.remove_container(container=vnf_fullname, force=force)
```

3.1.2.3 def hypervisor.docker_driver.DockerDriver.execute_in_guest (*self, host, user, vnf_name, cmd*)

Executed commands inside a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF
cmd	the command to execute inside the container

Returns

The output of the command passes as cmd

Definition at line 284 of file docker_driver.py.

```
284
285     def execute_in_guest(self, host, user, vnf_name, cmd):
286         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
287         if self.guest_status(host, user, vnf_name) != 'running':
288             raise errors.VNFNotRunningError
289         with self._error_handling(errors.VNFCommandExecutionError
      ):
290             response = dcx.execute(vnf_fullname,
291                 ["/bin/bash", "-c", cmd], stdout=True, stderr=False)
292             return response
```

3.1.2.4 def hypervisor.docker_driver.DockerDriver.get_id (*self, host, user, vnf_name*)

Returns a container's ID.

Parameters

host	IP address or hostname of the machine where the docker container is deployed
user	name of the user who owns the VNF
vnf_name	name of the VNF instance whose ID is being queried

Returns

docker container ID.

Definition at line 130 of file docker_driver.py.

```
130
131     def get_id(self, host, user, vnf_name):
132         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
133         return inspect_data['Id'].encode('ascii')
```

3.1.2.5 def hypervisor.docker_driver.DockerDriver.get_ip (*self, host, user, vnf_name*)

Returns a container's IP address.

Parameters

host	IP address or hostname of the machine where the docker container is deployed

user	name of the user who owns the VNF
vnf_name	name of the VNF instance whose ID is being queried

Returns

docker container's IP.

Definition at line 145 of file docker_driver.py.

```
145
146     def get_ip(self, host, user, vnf_name):
147         if self.guest_status(host, user, vnf_name) != 'running':
148             raise errors.VNFNotRunningError
149         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
150         return inspect_data['NetworkSettings']['IPAddress'].encode('ascii')
```

3.1.2.6 def hypervisor.docker_driver.DockerDriver.guest_status (*self, host, user, vnf_name*)

Returns the status of a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF

Returns

current state of the docker container

Definition at line 304 of file docker_driver.py.

```
304
305     def guest_status(self, host, user, vnf_name):
306         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
307         return inspect_data['State']['Status'].encode('ascii')
```

3.1.2.7 def hypervisor.docker_driver.DockerDriver.pause (*self, host, user, vnf_name*)

Pauses a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF

Definition at line 238 of file docker_driver.py.

```
238
239     def pause(self, host, user, vnf_name):
240         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
241         with self._error_handling(errors.VNFPauseError):
242             dcx.pause(container=vnf_fullname)
```

3.1.2.8 def hypervisor.docker_driver.DockerDriver.restart (*self, host, user, vnf_name*)

Restarts a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF

Definition at line 210 of file docker_driver.py.

```
210
211     def restart(self, host, user, vnf_name):
212         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
213         with self._error_handling(errors.VNFRestartError):
214             dcx.restart(container=vnf_fullname)
```

3.1.2.9 def hypervisor.docker_driver.DockerDriver.start (*self, host, user, vnf_name, is_privileged* = `True`)

Starts a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF
is_privileged	if True then the container is started in privileged mode

Definition at line 194 of file docker_driver.py.

```
194
195     def start(self, host, user, vnf_name, is_privileged=True):
196         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
197         with self._error_handling(errors.VNFStartError):
198             dcx.start(container=vnf_fullname,
199                 dns=self.__dns_list,
200                 privileged=is_privileged)
```

3.1.2.10 def hypervisor.docker_driver.DockerDriver.stop (*self, host, user, vnf_name*)

Stops a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF

Definition at line 224 of file docker_driver.py.

```
224
225     def stop(self, host, user, vnf_name):
226         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
227         with self._error_handling(errors.VNFStopError):
228             dcx.stop(container=vnf_fullname)
```

3.1.2.11 def hypervisor.docker_driver.DockerDriver.unpause (*self, host, user, vnf_name*)

Unpauses a docker container.

Parameters

host	IP address or hostname of the machine/VM where the docker container is deployed
user	name of the user
vnf_name	name of the VNF

Definition at line 251 of file docker_driver.py.

```
251
252     def unpause(self, host, user, vnf_name):
253         dcx, vnf_fullname, inspect_data = self._lookup_vnf(host, user, vnf_name)
254         with self._error_handling(errors.VNFUnpauseError):
255             dcx.unpause(container=vnf_fullname)
```

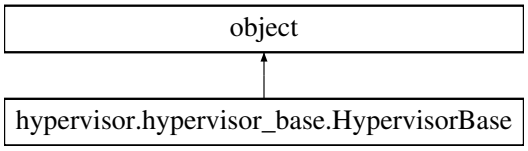
The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/hypervisor/docker_driver.py

3.2 hypervisor.hypervisor_base.HypervisorBase Class Reference

Base class for hypervisors.

Inheritance diagram for hypervisor.hypervisor_base.HypervisorBase:



Public Member Functions

- def get_id

  Returns the hypervisor specific ID of the VM or container.
- def deploy

  Deploys a VM or continer.
- def pause

  Pauses a VM or continer.
- def destroy

  Destroys a VM or continer.
- def execute_in_guest

  Executes a command in the VM or continer.
- def guest_status

  Returns the current status of a VM or continer.

3.2.1 Detailed Description

Base class for hypervisors.

This class must be extended by a hypervisor driver.

Definition at line 8 of file hypervisor_base.py.

3.2.2 Member Function Documentation

3.2.2.1 def hypervisor.hypervisor_base.HypervisorBase.deploy (*self*)

Deploys a VM or continer.

Args: Defined in derived class.

Returns: Hypervisor specific return code.

Definition at line 34 of file hypervisor_base.py.

```
34
35     def deploy(self):
36       pass
```

3.2.2.2 def hypervisor.hypervisor_base.HypervisorBase.destroy (*self*)

Destroys a VM or continer.

Args: Defined in derived class.

Returns: Hypervisor specific return code.

Definition at line 60 of file hypervisor_base.py.

```
60
61     def destroy(self):
62       pass
```

3.2.2.3 def hypervisor.hypervisor_base.HypervisorBase.execute_in_guest (*self*)

Executes a command in the VM or continer.

Args: Defined in derived class.

Returns: Hypervisor specific return code.

Definition at line 73 of file hypervisor_base.py.

```
73
74     def execute_in_guest(self):
75       pass
```

3.2.2.4 def hypervisor.hypervisor_base.HypervisorBase.get_id (*self*)

Returns the hypervisor specific ID of the VM or container.

Args: Defined in derived class.

Returns: Hypervisor specific ID for a VM or container.

Definition at line 21 of file hypervisor_base.py.

```
21
22     def get_id(self):
23       pass
```

3.2.2.5 def hypervisor.hypervisor_base.HypervisorBase.guest_status (*self*)

Returns the current status of a VM or continer.

Args: Defined in derived class.

Returns: Current status of a VM or container.

Definition at line 86 of file hypervisor_base.py.

```
86
87     def guest_status(self):
88       pass
```

3.2.2.6 def hypervisor.hypervisor_base.HypervisorBase.pause (*self*)

Pauses a VM or continer.

Args: Defined in derived class.

Returns: Hypervisor specific return code.

Definition at line 47 of file hypervisor_base.py.

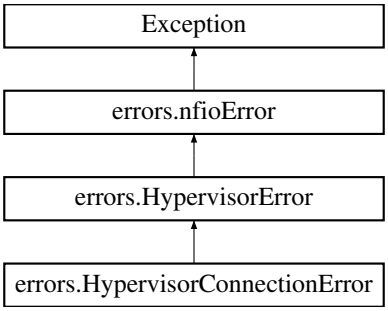
```
47
48     def pause(self):
49       pass
```

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/hypervisor/hypervisor_base.py

3.3 errors.HypervisorConnectionError Class Reference

Inheritance diagram for errors.HypervisorConnectionError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.3.1 Detailed Description

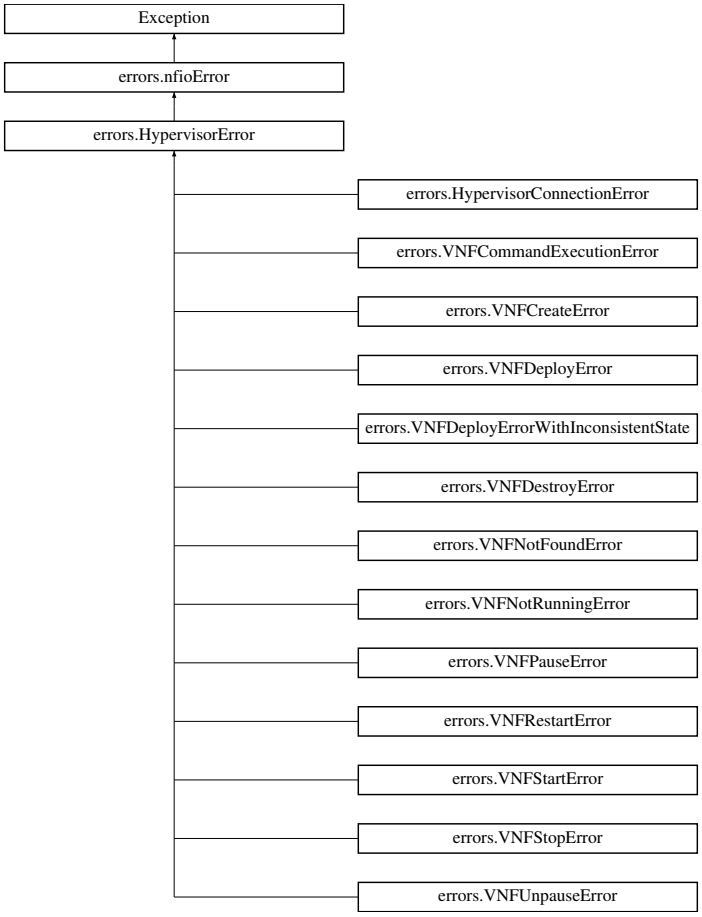
Definition at line 15 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.4 errors.HypervisorError Class Reference

Inheritance diagram for errors.HypervisorError:



3.4.1 Detailed Description

Definition at line 9 of file errors.py.

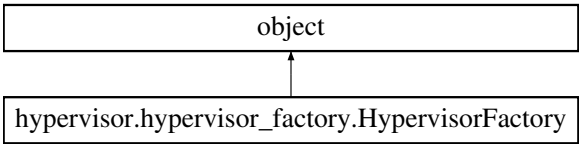
The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.5 hypervisor.hypervisor_factory.HypervisorFactory Class Reference

A singletone class for creating hypervisor driver objects.

Inheritance diagram for hypervisor.hypervisor_factory.HypervisorFactory:



Public Member Functions

- def __init__

  Instantiates a HypervisorFactory object.

Static Public Member Functions

- def get_hypervisor_instance

  Returns the hypervisor driver nstance.

3.5.1 Detailed Description

A singletone class for creating hypervisor driver objects.

For an instantiation of nf.io there can be exactly one object of only one type of hyperviosr. HyervisorFactory takes care of the creation logic.

Definition at line 11 of file hypervisor_factory.py.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 def hypervisor.hypervisor_factory.HypervisorFactory.__init__ (*self, hypervisor_type =* "DockerDriver")

Instantiates a HypervisorFactory object.

Args: hypervisor_type: The type of hypervisor object to instantiate. Valid hypervisor types are 'DockerDriver' and 'Libvirt' for the time being.

Returns: Nothing. Initializaes the factory object.

Note: If this factory class is instantiated multiple times with different types of hypervisor_type argument then it raises a ValueError.

If this factory class is instantiated with a hypervisor type other than Docker or Libvirt it raises a TypeError.

Definition at line 34 of file hypervisor_factory.py.

```
34
35     def __init__(self, hypervisor_type="DockerDriver"):
36         if not HypervisorFactory.__hyp_instance:
37             if hypervisor_type == "DockerDriver":
38                 HypervisorFactory.__hyp_instance_type = hypervisor_type
39                 HypervisorFactory.__hyp_instance = DockerDriver()
40             elif hypervisor_type == "Libvirt":
41                 HypervisorFactory.__hyp_instance_type = hypervisor_type
42                 HypervisorFactory.__hyp_instance = Libvirt()
43             else:
44                 raise TypeError(
45                     "Invalid hypervisor type. Valid types are: Docker, Libvirt")
46         elif HypervisorFactory.__hyp_instance_type != hypervisor_type:
47             raise ValueError(
48                 "An instantiation of type " +
49                 HypervisorFactory.__hyp_instance_type +
50                 " already exists.")
```

3.5.3 Member Function Documentation

3.5.3.1 def hypervisor.hypervisor_factory.HypervisorFactory.get_hypervisor_instance ( ) [static]

Returns the hypervisor driver nstance.

If the instance is not initialized then a RuntimeError is raised.

Definition at line 57 of file hypervisor_factory.py.

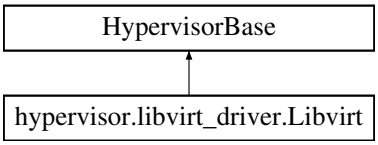
```
57
58     def get_hypervisor_instance():
59         if HypervisorFactory.__hyp_instance is not None:
60             return HypervisorFactory.__hyp_instance
61         else:
62             raise RuntimeError("Hypervisor not initialized.")
```

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/hypervisor/hypervisor_factory.py

3.6 hypervisor.libvirt_driver.Libvirt Class Reference

Inheritance diagram for hypervisor.libvirt_driver.Libvirt:



Public Member Functions

- def **deploy**
- def **pause**
- def **destroy**

3.6.1 Detailed Description

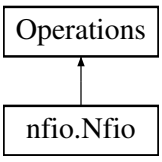
Definition at line 4 of file libvirt_driver.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/hypervisor/libvirt_driver.py

3.7 nfio.Nfio Class Reference

Inheritance diagram for nfio.Nfio:



Public Member Functions

- def __init__

  Instantiates a Nfio object.
- def **access**
- def **chmod**
- def **chown**
- def getattr

  Returns the file attributes of the file specified by path Args: path: Path of the file fh: Open file handle to the file Returns: A dictionary containing file attributes.
- def **readdir**
- def **readlink**
- def **mknod**
- def **rmdir**
- def mkdir

  The semantics have been redefined to create a new VNF instance when a directory is created under a specific type of VNF directory.
- def **statfs**
- def **unlink**
- def **symlink**
- def **rename**
- def **link**
- def **utimens**
- def **open**
- def **create**
- def read

  Reads an open file.
- def write

  Write to an open file.
- def **truncate**
- def **flush**
- def **release**
- def **fsync**

Public Attributes

- **root**
- **mountpoint**
- **hypervisor**
- **vnfs_ops**
- **module_root**

3.7.1 Detailed Description

Definition at line 22 of file nfio.py.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 def nfio.Nfio.__init__ (*self, root, mountpoint, hypervisor = 'Docker', module_root = 'middleboxes'*)

Instantiates a Nfio object.

Args: root: The root directory of nfio file system. The root directory stores persistent state about the system. mountpoint: The mountpoint of nfio file system. The mountpoint is required to intercept the file system calls via fuse. All the file system calls for fuse mounted files/directories are intercepted by libfuse and our provided implementation is executed. hypervisor: The type of hypervisor to use for deploying VNFs. The default is to use Docker containers. However, we also plan to add support for Libvirt. module_root: Root directory of the middlebox modules. Each middlebox provides it's own implementation of certain system calls in a separate module. module_root points to the root of that module. If nothing is provided a default of 'middleboxes' will be assumed. Returns: Nothing. Mounts nf.io file system at the specified mountpoint and creates a loop to act upon different file system calls.

Definition at line 52 of file nfio.py.

```
52
53             module_root='middleboxes'):
54         self.root = root
55         self.mountpoint = mountpoint
56         self.hypervisor = hypervisor
57         self.vnfs_ops = VNFSOperations(root)
58         self.module_root = module_root
```

3.7.3 Member Function Documentation

3.7.3.1 def nfio.Nfio.getattr (*self, path, fh = None*)

Returns the file attributes of the file specified by path Args: path: Path of the file fh: Open file handle to the file Returns: A dictionary containing file attributes.

The dictionary contains the following keys: st_atime: Last access time st_ctime: File creation time st_gid: Group id of the owner group st_mode: File access mode st_mtime: Last modification time st_nlink: Number of symbolic links to the file st_size: Size of the file in bytes st_uid: User id of the file owner Note: For special placeholder files for VNFs, st_size is set to a constant 1000. This is to make sure read utilities such as cat work for these special placeholder files.

Definition at line 119 of file nfio.py.

```
119
120     def getattr(self, path, fh=None):
121         opcode = self.vnfs_ops.vnfs_get_opcode(path)
122         if opcode == VNFSOperations.OP_NF:
123             nf_type = self.vnfs_ops.vnfs_get_nf_type(path)
124             if len(nf_type) > 0:
125                 try:
126                     mbox_module = importlib.import_module(
127                         self.module_root +
128                         "." +
129                         nf_type)
130                 except ImportError:
131                     logger.error('VNF module file missing. Add "' + nf_type
132                         + '.py" under the directory ' + self.module_root)
133                     ## TODO: raise an custom exception and handle it in a OS
134                     ## specific way
135                     raise OSError(errno.ENOSYS)
136                 return mbox_module._getattr(self.root, path, fh)
137         full_path = self._full_path(path)
138         st = os.lstat(full_path)
139         file_name = self.vnfs_ops.vnfs_get_file_name(full_path)
140         return dict(
141             (key,
142              getattr(
143                  st,
144                  key)) for key in (
145                 'st_atime',
146                 'st_ctime',
147                 'st_gid',
148                 'st_mode',
149                 'st_mtime',
```

```
150                 'st_nlink',
151                 'st_size',
152                 'st_uid'))
```

3.7.3.2 def nfio.Nfio.mkdir (*self, path, mode*)

The semantics have been redefined to create a new VNF instance when a directory is created under a specific type of VNF directory.

Args: path: path of the directory to create. The path also represents the name of the new VNF instance to be created. mode: File access mode for the new directory. Returns: If path does not correspond to a directory under a specific VNF type directory then errno.EPERM is returned. Otherwise the return code is same as os.mkdir()'s return code.

Definition at line 188 of file nfio.py.

```
188
189     def mkdir(self, path, mode):
190         opcode = self.vnfs_ops.vnfs_get_opcode(path)
191         if opcode == VNFSOperations.OP_NF:
192             nf_type = self.vnfs_ops.vnfs_get_nf_type(path)
193             # Check if this directory is an instance directory or a type
194             # directory
195             path_tokens = path.split("/")
196             if path_tokens.index("nf-types") == len(path_tokens) - 2:
197                 return os.mkdir(self._full_path(path), mode)
198             mbox_module = importlib.import_module(
199                 self.module_root +
200                 "." +
201                 nf_type)
202             result = mbox_module._mkdir(self.root, path, mode)
203         elif opcode == VNFSOperations.OP_UNDEFINED:
204             result = errno.EPERM
205         return result
```

3.7.3.3 def nfio.Nfio.read (*self, path, length, offset, fh*)

Reads an open file.

This nfio specific implementation parses path to see if the read is from any VNF or not. In case the read is from a VNF, the corresponding VNF module is loaded and the module's _read function is invoked to complete the read system call.

Args: path: path represents the path of the file to read from length: number of bytes to read from the file offset: byte offset indicating the starting byte to read from fh: file descriptor of the open file represented by path

Returns: length bytes from offset byte of the file represented by fh and path

Notes: VNFs can have special files which are placeholders for statistics such as number of received/sent bytes etc. VNFs provide their own implementation of read and handle reading of these special placeholder files.

Definition at line 273 of file nfio.py.

```
273
274     def read(self, path, length, offset, fh):
275         full_path = self._full_path(path)
276         opcode = self.vnfs_ops.vnfs_get_opcode(full_path)
277         file_name = self.vnfs_ops.vnfs_get_file_name(full_path)
278         if opcode == self.vnfs_ops.OP_NF:
279             nf_type = self.vnfs_ops.vnfs_get_nf_type(path)
280             mbox_module = importlib.import_module(
281                 self.module_root +
282                 "." +
283                 nf_type)
284             return mbox_module._read(self.root, path, length, offset, fh)
285         os.lseek(fh, offset, os.SEEK_SET)
286         return os.read(fh, length)
```

3.7.3.4 def nfio.Nfio.write (*self, path, buf, offset, fh*)

Write to an open file.

In this nfio specific implementation the path is parsed to see if the write is for any specific VNF or not. If the write is for any file under a VNF directory then the corresponding VNF module is loaded and the module's _write function is invoked.

Args: path: path to the file to write buf: the data to write offset: the byte offset at which the write should begin fh: file descriptor of the open file represented by path

Returns: Returns the number of bytes written to the file starting at offset

Note: VNFs can have special files where writing specific strings trigger a specific function. For example, writing 'activate' to the 'action' file of a VNF will start the VNF. VNF specific modules handle such special cases of writing.

Definition at line 309 of file nfio.py.

```
309
310     def write(self, path, buf, offset, fh):
311         opcode = self.vnfs_ops.vnfs_get_opcode(path)
312         full_path = self._full_path(path)
313         file_name = self.vnfs_ops.vnfs_get_file_name(path)
314         if opcode == VNFSOperations.OP_NF:
315             nf_type = self.vnfs_ops.vnfs_get_nf_type(full_path)
316             mbox_module = importlib.import_module(
317                 self.module_root +
318                 "." +
319                 nf_type)
320             return mbox_module._write(self.root, path, buf, offset, fh)
321
322         os.lseek(fh, offset, os.SEEK_SET)
323         return os.write(fh, buf)
```

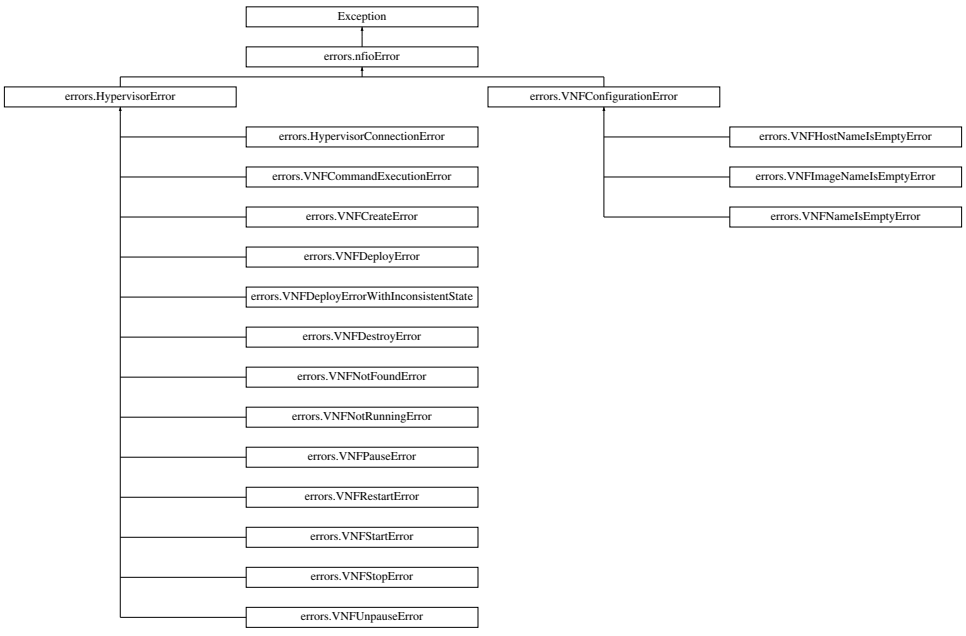
The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/nfio.py

3.8 errors.nfioError Class Reference

This module contains all the custom exceptions defined for nf.io.

Inheritance diagram for errors.nfioError:



3.8.1 Detailed Description

This module contains all the custom exceptions defined for nf.io.

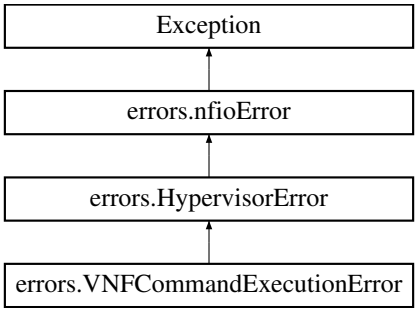
Definition at line 6 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.9 errors.VNFCommandExecutionError Class Reference

Inheritance diagram for errors.VNFCommandExecutionError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.9.1 Detailed Description

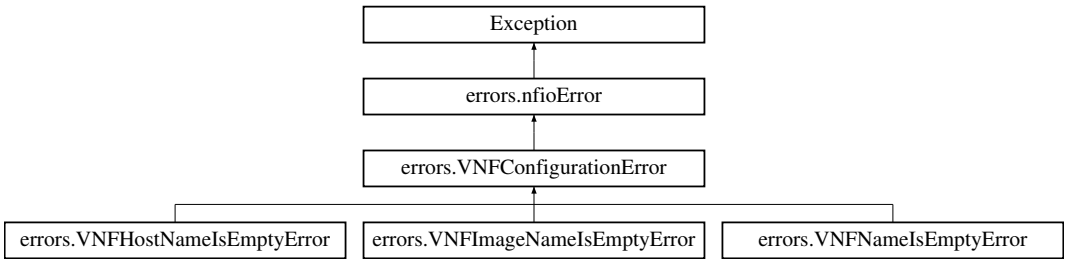
Definition at line 23 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.10 errors.VNFConfigurationError Class Reference

Inheritance diagram for errors.VNFConfigurationError:



3.10.1 Detailed Description

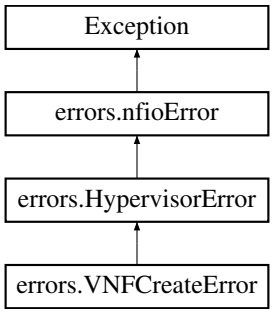
Definition at line 12 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.11 errors.VNFCreateError Class Reference

Inheritance diagram for errors.VNFCreateError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.11.1 Detailed Description

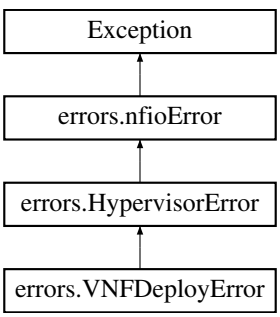
Definition at line 27 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.12 errors.VNFDeployError Class Reference

Inheritance diagram for errors.VNFDeployError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.12.1 Detailed Description

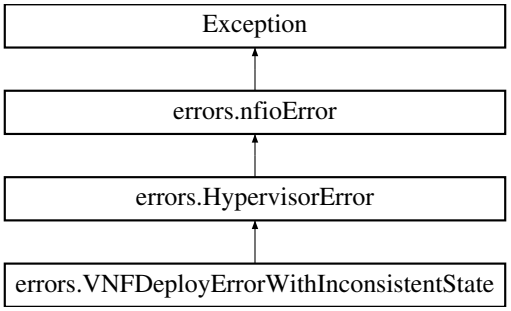
Definition at line 31 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.13 errors.VNFDeployErrorWithInconsistentState Class Reference

Inheritance diagram for errors.VNFDeployErrorWithInconsistentState:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.13.1 Detailed Description

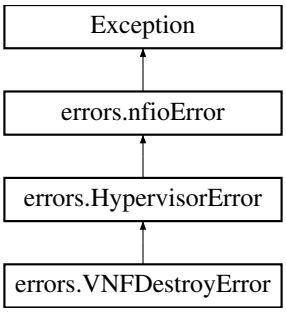
Definition at line 59 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.14 errors.VNFDestroyError Class Reference

Inheritance diagram for errors.VNFDestroyError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.14.1 Detailed Description

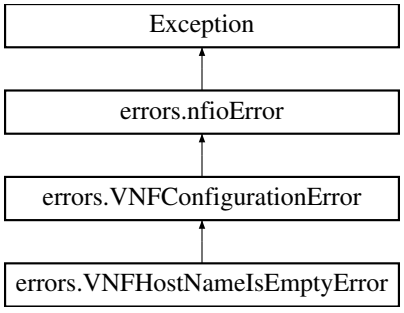
Definition at line 35 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.15 errors.VNFHostNameIsEmptyError Class Reference

Inheritance diagram for errors.VNFHostNameIsEmptyError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.15.1 Detailed Description

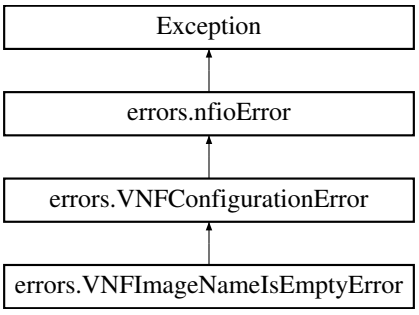
Definition at line 67 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.16 errors.VNFImageNameIsEmptyError Class Reference

Inheritance diagram for errors.VNFImageNameIsEmptyError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.16.1 Detailed Description

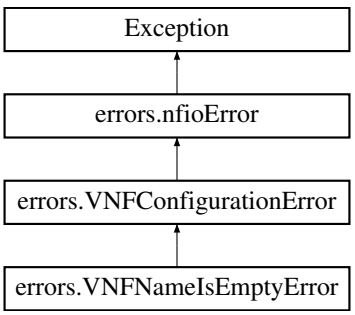
Definition at line 63 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.17 errors.VNFNameIsEmptyError Class Reference

Inheritance diagram for errors.VNFNameIsEmptyError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.17.1 Detailed Description

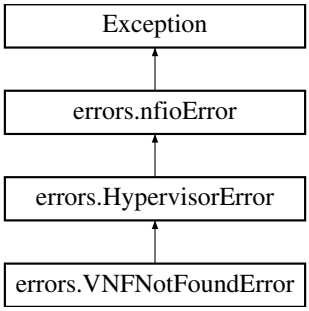
Definition at line 71 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.18 errors.VNFNotFoundError Class Reference

Inheritance diagram for errors.VNFNotFoundError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.18.1 Detailed Description

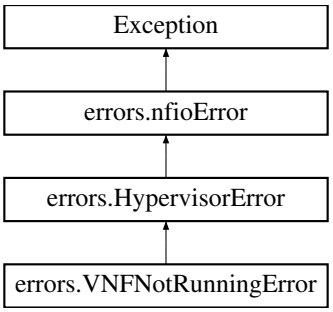
Definition at line 19 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.19 errors.VNFNotRunningError Class Reference

Inheritance diagram for errors.VNFNotRunningError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.19.1 Detailed Description

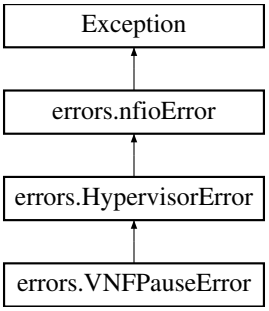
Definition at line 75 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.20 errors.VNFPauseError Class Reference

Inheritance diagram for errors.VNFPauseError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.20.1 Detailed Description

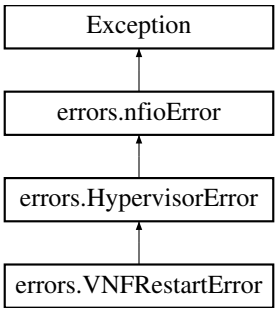
Definition at line 51 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.21 errors.VNFRestartError Class Reference

Inheritance diagram for errors.VNFRestartError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.21.1 Detailed Description

Definition at line 43 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.22 vnfs_operations.VNFSOperations Class Reference

Provides a common set of operations for nfio.

Public Member Functions

- def **__init__**
- def vnfs_create_vnf_instance

  Create the file system structure for a VNF.
- def vnfs_get_opcode

  Determinse the type of operation based on the path.
- def vnfs_get_nf_type

Parse the type of VNF from path.

- def vnfs_get_file_name

  Return the name of the file represented by a path.
- def vnfs_is_nf_instance

  Determines if a path represents an nf instance directory.
- def vnfs_get_instance_configuration

  Return the configuration parameters related to a VNF instance.
- def vnfs_deploy_nf

  Deploys and STARTS a VNF instance.
- def vnfs_stop_vnf

  Stops a VNF instance.
- def vnfs_start_vnf

  Starts a deployed VNF instance.
- def vnfs_destroy_vnf

  Destroys a deployed VNF instance.
- def vnfs_get_rx_bytes

  Reads the number of bytes received by a VNF instance.
- def vnfs_get_tx_bytes

  Reads the number of bytes sent by a VNF instance.
- def vnfs_get_pkt_drops

  Reads the number of packets dropped by a VNF instance.
- def vnfs_get_status

  Get the status of a VNF instance, e.g., the VNF is running/suspended/stopped etc.
- def vnfs_get_ip

  Get the status of a VNF instance, e.g., the VNF is running/suspended/stopped etc.

Public Attributes

- **vnfs_root**

Static Public Attributes

- int **OP_UNDEFINED** = 0xFF
- int **OP_NF** = 0x01

3.22.1 Detailed Description

Provides a common set of operations for nfio.

These operations act as a helper.

Definition at line 25 of file vnfs_operations.py.

3.22.2 Member Function Documentation

3.22.2.1 def vnfs_operations.VNFSOperations.vnfs_create_vnf_instance (*self, path, mode*)

Create the file system structure for a VNF.

Args: path: path of the new VNF instance. mode: file creation mode for the new VNF instance directory.

Returns: returns the return code of os.mkdir

Definition at line 52 of file vnfs_operations.py.

```
52
53     def vnfs_create_vnf_instance(self, path, mode):
54         logger.info('Creating file/directory structure in ' + path)
55         full_path = self._full_path(path)
56         result = os.mkdir(full_path)
57         default_file_mode = 0o644
58         os.open(
59             full_path +
60             "/status",
61             os.O_WRONLY | os.O_CREAT,
62             default_file_mode)
63
64         os.mkdir(full_path + "/config", mode)
65         os.open(full_path + "/config/boot.conf", os.O_WRONLY | os.O_CREAT,
66                 default_file_mode)
67         os.mkdir(full_path + "/machine", mode)
68         os.open(full_path + "/machine/ip", os.O_WRONLY | os.O_CREAT,
69                 default_file_mode)
70         os.open(full_path + "/machine/vm.vcpu", os.O_WRONLY | os.O_CREAT,
71                 default_file_mode)
72         os.open(full_path + "/machine/vm.memory", os.O_WRONLY | os.O_CREAT,
73                 default_file_mode)
74         os.open(full_path + "/machine/vm.image", os.O_WRONLY | os.O_CREAT,
75                 default_file_mode)
76         os.open(full_path + "/machine/vm.ip", os.O_WRONLY | os.O_CREAT,
77                 default_file_mode)
78         os.open(full_path + "/action", os.O_WRONLY | os.O_CREAT,
79                 default_file_mode)
80
81         default_file_mode = 0o444
82         os.mkdir(full_path + "/stats", mode)
83         os.open(full_path + "/stats/rx_bytes", os.O_WRONLY | os.O_CREAT,
84                 default_file_mode)
85         os.open(full_path + "/stats/tx_bytes", os.O_WRONLY | os.O_CREAT,
86                 default_file_mode)
87         os.open(full_path + "/stats/pkt_drops", os.O_WRONLY | os.O_CREAT,
88                 default_file_mode)
89         logger.info('Finished creating file/directory structure in ' + path)
90         return result
```

3.22.2.2 def vnfs_operations.VNFSOperations.vnfs_deploy_nf (*self, nf_path*)

Deploys and STARTS a VNF instance.

Args: nf_path: path of the VNF instance.

Returns

void

Definition at line 207 of file vnfs_operations.py.

```
207
208     def vnfs_deploy_nf(self, nf_path):
209         logger.info('Deploying new VNF at ' + nf_path)
210         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(nf_path)
211         try:
212             cont_id  = self._hypervisor.deploy(
213                 ip_address, getpass.getuser(), image_name, nf_instance_name)
214             logger.debug(cont_id)
215         except errors.VNFDeployError:
216             logger.info('Instance: ' + nf_instance_name  + ' deployment failed')
217         else:
218             logger.info('Instance: ' + nf_instance_name
219                     + ' successfully deployed')
220             try:
221                 logger.info('Starting the deployed VNF instance: '
222                     + nf_instance_name)
223                 self._hypervisor.start(ip_address, cont_id)
224             except errors.VNFStartError:
225                 logger.info('Instance: ' + nf_instance_name  + ' start failed')
226                 # destroy the deployed VNF
227                 self._hypervisor.destroy(ip_address, cont_id)
228                 logger.info('Instance: ' + nf_instance_name
229                     + ' destroyed')
230             else:
231                 logger.info('Instance: ' + nf_instance_name
232                     + ' successfully deployed and started')
```

3.22.2.3 def vnfs_operations.VNFSOperations.vnfs_destroy_vnf (*self, nf_path*)

Destroys a deployed VNF instance.

Args: nf_path: path of the VNF instance.

Returns: return codes are described in hypervisor.hypervisor_return_codes module.

Definition at line 278 of file vnfs_operations.py.

```
278
279     def vnfs_destroy_vnf(self, nf_path):
280         logger.info("Destroying VNF at " + nf_path)
281         nf_instance_name, nf_type, ip_address, image_name = self.
      vnfs_get_instance_configuration(
282             nf_path)
283         self._hypervisor.destroy(ip_address, getpass.getuser(), nf_instance_name)
284         logger.info('Instance: ' + nf_instance_name + ' successfully destroyed')
```

3.22.2.4 def vnfs_operations.VNFSOperations.vnfs_get_file_name (*self, path*)

Return the name of the file represented by a path.

Args: path: the path of the file in concern

Returns: returns the name of the file, i.e., last token after / in the path.

Definition at line 141 of file vnfs_operations.py.

```
141
142     def vnfs_get_file_name(self, path):
143
144         return path.split('/')[-1]
```

3.22.2.5 def vnfs_operations.VNFSOperations.vnfs_get_instance_configuration (*self, nf_path*)

Return the configuration parameters related to a VNF instance.

Args: nf_path: path of the VNF instance. e.g., /mnt/vnfsmnt/firewall/fw-alpha

Returns: A tuple representing the configuration of the VNF instance. The tuple is organized in the following order: nf_instance_name: name of the VNF instance. nf_type: type of the VNF. ip_address: IP address of the machine where this VNF will be deployed. image_name: name of the VM/container image for that VNF.

Definition at line 184 of file vnfs_operations.py.

```
184
185     def vnfs_get_instance_configuration(self, nf_path):
186         nf_instance_name = self.vnfs_get_file_name(nf_path)
187         nf_type = self.vnfs_get_nf_type(nf_path)
188         ip_address = ''
189         logger.debug(nf_path + '/machine/ip')
190         with open(nf_path + '/machine/ip') as ip_fd:
191             ip_address = ip_fd.readline().rstrip('\n')
192         image_name = ''
193         with open(nf_path + '/machine/vm.image') as img_fd:
194             image_name = img_fd.readline().rstrip('\n')
195         logger.info("Instance name: " + nf_instance_name + ", type: "
196             + nf_type + ", host-ip: " + ip_address + " VNF image: " + image_name)
197         return nf_instance_name, nf_type, ip_address, image_name
```

3.22.2.6 def vnfs_operations.VNFSOperations.vnfs_get_ip (*self, nf_path*)

Get the status of a VNF instance, e.g., the VNF is running/suspended/stopped etc.

Args: nf_path: path of the VNF instance.

Returns: Hypervisor specific status of the VNF. For example, if Docker is being used for VNF deployment then Docker specific container status message is returned.

Definition at line 390 of file vnfs_operations.py.

```
390
391     def vnfs_get_ip(self, nf_path):
392         logger.info('Reading ip at ' + nf_path)
393         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
394             nf_path)
395         cont_ip = self._hypervisor.get_ip(ip_address, getpass.getuser(), nf_instance_name)
396         logger.debug('cont_ip ' + cont_ip)
397         logger.info('Successfully read ip')
398         return cont_ip
399
```

3.22.2.7 def vnfs_operations.VNFSOperations.vnfs_get_nf_type (*self, path*)

Parse the type of VNF from path.

Args: path: the path of the file/directory on which some operation is being performed.

Returns: Returns the type of VNF parsed from the path, e.g., if the path is /mnt/vnfsroot/nf-types/firewall/fw-alpha/action then returns firewall.

Definition at line 122 of file vnfs_operations.py.

```
122
123     def vnfs_get_nf_type(self, path):
124         tokens = self._full_path(path).encode('ascii').split('/')
125         try:
126             return tokens[tokens.index("nf-types") + 1]
127         except ValueError:
128             return ""
129         except IndexError:
130             return ""
```

3.22.2.8 def vnfs_operations.VNFSOperations.vnfs_get_opcode (*self, path*)

Determinse the type of operation based on the path.

Args: path: path to the file/directory on which the operation is being performed

Returns: If the file is under nf-types subdirectory in the nfio mount, then returns OP_NF. Otherwise, returns OP_UNDEFINED.

Definition at line 103 of file vnfs_operations.py.

```
103
104     def vnfs_get_opcode(self, path):
105         tokens = self._full_path(path).encode('ascii').split('/')
106         if "nf-types" in tokens:
107             return VNFSOperations.OP_NF
108         return VNFSOperations.OP_UNDEFINED
```

3.22.2.9 def vnfs_operations.VNFSOperations.vnfs_get_pkt_drops (*self, nf_path*)

Reads the number of packets dropped by a VNF instance.

Args: nf_path: path of the VNF instance.

Returns: returns the number of packets dropped by a VNF instance.

Definition at line 339 of file vnfs_operations.py.

```
339
340     def vnfs_get_pkt_drops(self, nf_path):
341         logger.info('Reading pkt_drops at ' + nf_path)
342         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
343             nf_path)
344         command = "ifconfig eth0 | grep -Eo 'RX .* dropped:[0-9]+' | cut -d':' -f 4"
345         response = self._hypervisor.execute_in_guest(
346             ip_address,
347             getpass.getuser(), nf_instance_name,
348             command)
349         logger.info('Successfully read pkt_drops')
350         return response
```

3.22.2.10 def vnfs_operations.VNFSOperations.vnfs_get_rx_bytes (*self, nf_path*)

Reads the number of bytes received by a VNF instance.

Args: nf_path: path of the VNF instance.

Returns: returns the number of bytes received by a VNF instance.

Definition at line 295 of file vnfs_operations.py.

```
295
296     def vnfs_get_rx_bytes(self, nf_path):
297         logger.info('Reading rx_bytes at ' + nf_path)
298         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
299             nf_path)
300         command = "ifconfig eth0 | grep -Eo 'RX bytes:[0-9]+' | cut -d':' -f 2"
301         response = self._hypervisor.execute_in_guest(
302             ip_address,
303             getpass.getuser(), nf_instance_name,
304             command)
305         logger.info('Successfully read rx_bytes')
306         return response
```

3.22.2.11 def vnfs_operations.VNFSOperations.vnfs_get_status (*self, nf_path*)

Get the status of a VNF instance, e.g., the VNF is running/suspended/stopped etc.

Args: nf_path: path of the VNF instance.

Returns: Hypervisor specific status of the VNF. For example, if Docker is being used for VNF deployment then Docker specific container status message is returned.

Definition at line 364 of file vnfs_operations.py.

```
364
365     def vnfs_get_status(self, nf_path):
366         logger.info('Reading status at ' + nf_path)
367         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
368             nf_path)
369         response = ''
370         try:
371             response = self._hypervisor.guest_status(ip_address, getpass.getuser(),
372                 nf_instance_name)
373         except errors.VNFNotFoundError:
374             logger.info('Instance: ' + nf_instance_name + ' does not exist')
375         logger.info('Successfully read status')
376         return response
```

3.22.2.12 def vnfs_operations.VNFSOperations.vnfs_get_tx_bytes (*self, nf_path*)

Reads the number of bytes sent by a VNF instance.

Args: nf_path: path of the VNF instance.

Returns: returns the number of bytes sent by a VNF instance.

Definition at line 317 of file vnfs_operations.py.

```
317
318     def vnfs_get_tx_bytes(self, nf_path):
319         logger.info('Reading tx_bytes at ' + nf_path)
320         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
321             nf_path)
322         command = "ifconfig eth0 | grep -Eo 'TX bytes:[0-9]+' | cut -d':' -f 2"
323         response = self._hypervisor.execute_in_guest(
324             ip_address,
325             getpass.getuser(), nf_instance_name,
326             command)
327         logger.info('Successfully read tx_bytes')
328         return response
```

3.22.2.13 def vnfs_operations.VNFSOperations.vnfs_is_nf_instance (*self, path*)

Determines if a path represents an nf instance directory.

Args: path: path of the file/directory in concern.

Returns: True: if path represents an nf instance directory. For example, if path is /mnt/vnfsmnt/nf-types/firewall/fw-alpha then returns True.

False: if the path does not represent an nf instance directory. For example, if path is /mnt/vnfsmnt/nf-types/firewall/fw-alpha/action then returns False.

Definition at line 160 of file vnfs_operations.py.

```
160
161     def vnfs_is_nf_instance(self, path):
162         tokens = self._full_path(path).encode('ascii').split('/')
163         if ('nf-types' in tokens) and len(tokens) > tokens.index('nf-types') + \
164                 1:
165             return True
166         return False
```

3.22.2.14 def vnfs_operations.VNFSOperations.vnfs_start_vnf (*self, nf_path*)

Starts a deployed VNF instance.

Args: nf_path: path of the VNF instance.

Returns: return codes are described in hypervisor.hypervisor_return_codes module.

Definition at line 260 of file vnfs_operations.py.

```
260
261     def vnfs_start_vnf(self, nf_path):
262         logger.info("Starting VNF at " + nf_path)
263         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
264             nf_path)
265         self._hypervisor.start(ip_address, getpass.getuser(), nf_instance_name)
266         logger.info('Instance: ' + nf_instance_name + ' successfully started')
```

3.22.2.15 def vnfs_operations.VNFSOperations.vnfs_stop_vnf (*self, nf_path*)

Stops a VNF instance.

Args: nf_path: path of the VNF instance.

Returns

void

Definition at line 242 of file vnfs_operations.py.

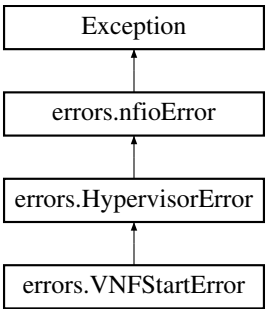
```
242
243     def vnfs_stop_vnf(self, nf_path):
244         logger.info("Stopping VNF at " + nf_path)
245         nf_instance_name, nf_type, ip_address, image_name = self.
    vnfs_get_instance_configuration(
246             nf_path)
247         self._hypervisor.stop(ip_address, getpass.getuser(), nf_instance_name )
248         logger.info('Instance: ' + nf_instance_name + ' successfully stopped')
```

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/vnfs_operations.py

3.23 errors.VNFStartError Class Reference

Inheritance diagram for errors.VNFStartError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.23.1 Detailed Description

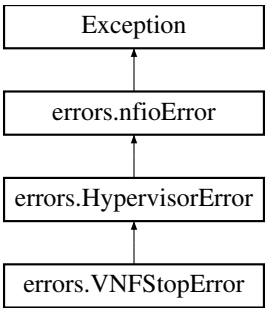
Definition at line 39 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.24 errors.VNFStopError Class Reference

Inheritance diagram for errors.VNFStopError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.24.1 Detailed Description

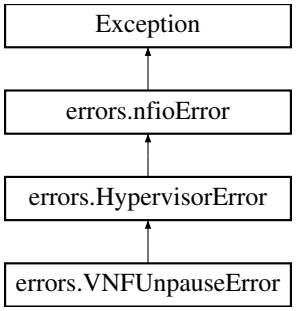
Definition at line 47 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

3.25 errors.VNFUnpauseError Class Reference

Inheritance diagram for errors.VNFUnpauseError:



Public Member Functions

- def **__init__**

Public Attributes

- **errno**

3.25.1 Detailed Description

Definition at line 55 of file errors.py.

The documentation for this class was generated from the following file:

- WatNFV/nf.io/src/errors.py

Index