# ESCAPEv2 Documentation

***Release 2.0.0***

**János Czentye**

June 29, 2015

Contents

Welcome! This is the API documentation for **ESCAPEv2**.

CHAPTER 1

# Overview

Mininet is a great prototyping tool which takes existing SDN-related software components (e.g. Open vSwitch, OpenFlow controllers, network namespaces, cgroups, etc.) and combines them into a framework, which can automatically set up and configure customized OpenFlow testbeds scaling up to hundreds of nodes. Standing on the shoulders of Mininet, we have implemented a similar prototyping system called ESCAPE, which can be used to develop and test various components of the service chaining architecture. Our framework incorporates Click for implementing Virtual Network Functions (VNF), NETCONF (**RFC 6241**) for managing Click-based VNFs and POX for taking care of traffic steering. We also add our extensible Orchestrator module, which can accommodate mapping algorithms from abstract service descriptions to deployed and running service chains.

**See also:**

The source code of previous ESCAPE version is available at our github page. For more information we first suggest to read our paper:

Attila Csoma, Balazs Sonkoly, Levente Csikor, Felician Nemeth, Andras Gulyas, Wouter Tavernier, and Sahel Sahhaf: **ESCAPE: Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX**. Demo paper at Sigcomm'14.

- Download the paper
- Accompanying poster

For further information contact csoma@tmit.bme.hu, sonkoly@tmit.bme.hu

CHAPTER 2

# ESCAPEv2 structure

## 2.1 Dependencies:

```
$ sudo apt-get install libxml2 libxslt1-dev python-setuptools python-pip \
python-paramiko python-lxml python-libxml2 python-libxslt1

sudo pip install networkx ncclient requests
```

## 2.2 Class structure

### 2.2.1 The *escape* package

#### *escape* package

Unifying package for ESCAPEv2 functions

*CONFIG* contains the ESCAPEv2 dependent configuration such as the concrete RequestHandler and strategy classes, the initial Adapter classes, etc.

##### Submodules

***escape.service* package** Subpackage for classes related mostly to Service (Graph) Adaptation sublayer

**Submodules**

***element_mgmt.py* module** *AbstractElementManager* is an abstract class for element managers

*ClickManager* represent the interface to Click elements

**Module contents** Contains classes relevant to element management

**class** escape.service.element_mgmt.**AbstractElementManager**
Bases: object

Abstract class for element management components (EM)

> **Warning:** Not implemented yet!

**__init__**()
Init

**class** escape.service.element_mgmt.**ClickManager**
Bases: *escape.service.element_mgmt.AbstractElementManager*

Manager class for specific VNF management based on Clicky

> **Warning:** Not implemented yet!

**__init__**()
Init

***sas_mapping.py* module** *DefaultServiceMappingStrategy* implements a default mapping algorithm which map given SG on a single Bis-Bis

*ServiceGraphMapper* perform the supplementary tasks for SG mapping

**Module contents** Contains classes which implement SG mapping functionality

**class** escape.service.sas_mapping.**DefaultServiceMappingStrategy**
Bases: *escape.util.mapping.AbstractMappingStrategy*

Mapping class which maps given Service Graph into a single BiS-BiS

**__init__**()
Init

**classmethod map**(*graph*, *resource*)
Default mapping algorithm which maps given Service Graph on one BiS-BiS

**Parameters**

- **graph** (NFFG) – Service Graph
- **resource** (NFFG) – virtual resource

**Returns** Network Function Forwarding Graph

**Return type** *NFFG*

**class** escape.service.sas_mapping.**SGMappingFinishedEvent**(*nffg*)
Bases: pox.lib.revent.revent.Event

Event for signaling the end of SG mapping

**__init__**(*nffg*)
Init

**Parameters** **nffg** (NFFG) – NF-FG need to be initiated

**class** escape.service.sas_mapping.**ServiceGraphMapper**
Bases: *escape.util.mapping.AbstractMapper*

Helper class for mapping Service Graph to NF-FG

**_eventMixin_events = set([<class 'escape.service.sas_mapping.SGMappingFinishedEvent'>])**

**__init__**()
Init Service mapper

**Returns** None

**orchestrate**(*input_graph*, *resource_view*)
Orchestrate mapping of given service graph on given virtual resource

**Parameters**

- **input_graph** (NFFG) – Service Graph
- **resource_view** – virtual resource view
- **resource_view** – ESCAPEVirtualizer

**Returns** Network Function Forwarding Graph

**Return type** *NFFG*

**_mapping_finished**(*nffg*)
Called from a separate thread when the mapping process is finished

**Parameters** **nffg** (NFFG) – geenrated NF-FG

**Returns** None

***sas_API.py* module** *InstantiateNFFGEvent* can send NF-FG to the lower layer

*GetVirtResInfoEvent* can request virtual resource info from lower layer

*ServiceRequestHandler* implement the specific RESTful API functionality thereby realizes the UNIFY's U - Sl API

*ServiceLayerAPI* represents the SAS layer and implement all related functionality

**Module contents** Implements the platform and POX dependent logic for the Service Adaptation Sublayer

**class** escape.service.sas_API.**InstantiateNFFGEvent**(*nffg*)
Bases: pox.lib.revent.revent.Event

Event for passing NFFG (mapped SG) to Orchestration layer

**__init__**(*nffg*)
Init

**Parameters** **nffg** (NFFG) – NF-FG need to be initiated

**class** escape.service.sas_API.**GetVirtResInfoEvent**(*sid*)
Bases: pox.lib.revent.revent.Event

Event for requesting virtual resource info from Orchestration layer

**__init__**(*sid*)
Init

**Parameters** **sid** (*int*) – Service layer ID

**class** escape.service.sas_API.**ServiceRequestHandler**(*request*, *client_address*, *server*)
Bases: *escape.util.api.AbstractRequestHandler*

Request Handler for Service Adaptation SubLayer

> **Warning:** This class is out of the context of the recoco's co-operative thread context! While you don't need to worry much about synchronization between recoco tasks, you do need to think about synchronization between recoco task and normal threads. Synchronisation is needed to take care manually: use relevant helper function of core object: *callLater*/*raiseLater* or use *schedule_as_coop_task* decorator defined in util.misc on the called function

**request_perm = {'PUT': ('echo',), 'POST': ('echo', 'sg'), 'DELETE': ('echo',), 'GET': ('echo', 'version', 'operations')}**

**bounded_layer = 'service'**

**log = <logging.Logger object>**

**echo**()
Test function for REST-API

**Returns** None

**sg**()
Main API function for Service Graph initiation

Bounded to POST HTTP verb

**version**()
Return with version

**Returns** None

**operations**()
Return with allowed operations

**Returns** None

**class** escape.service.sas_API.**ServiceLayerAPI**(*standalone=False*, ***kwargs*)
Bases: *escape.util.api.AbstractAPI*

Entry point for Service Adaptation Sublayer

Maintain the contact with other UNIFY layers

Implement the U - Sl reference point

**_core_name = 'service'**

**dependencies = ('orchestration',)**

**__init__**(*standalone=False*, ***kwargs*)

**See also:**

*AbstractAPI.__init__()*

**initialize**()

**See also:**

*AbstractAPI.initialze()*

**shutdown**(*event*)

**See also:**

*AbstractAPI.shutdown()*

**_initiate_rest_api**(*handler=<class escape.service.sas_API.ServiceRequestHandler>*, *address='localhost'*, *port=8008*)
Initialize and set up REST API in a different thread

**Parameters**

- **address** (*str*) – server address, default localhost
- **port** (*int*) – port number, default 8008

**Returns** None

**_initiate_gui**()
Initiate and set up GUI

**_handle_SGMappingFinishedEvent**(*event*)
Handle SGMappingFinishedEvent and proceed with *NFFG* instantiation

**Parameters** **event** (SGMappingFinishedEvent) – event object

**Returns** None

**request_service**(**args*, ***kwargs*)
Initiate a Service Graph (UNIFY U-Sl API)

**Parameters** **sg** (NFFG) – service graph instance

**Returns** None

**_instantiate_NFFG**(*nffg*)
Send NFFG to Resource Orchestration Sublayer in an implementation-specific way

General function which is used from microtask and Python thread also

**Parameters** **nffg** (NFFG) – mapped Service Graph

**Returns** None

**_handle_MissingVirtualViewEvent**(*event*)
Request virtual resource info from Orchestration layer (UNIFY Sl - Or API)

Invoked when a `MissingVirtualViewEvent` raised

Service layer is identified with the sid value automatically

**Parameters** **event** (MissingVirtualViewEvent) – event object

**Returns** None

**_handle_VirtResInfoEvent**(*event*)
Save requested virtual resource info as an *ESCAPEVirtualizer*

**Parameters** **event** (VirtResInfoEvent) – event object

**Returns** None

**_handle_InstantiationFinishedEvent**(*event*)

***sas_orchestration.py* module** *ServiceOrchestrator* orchestrates SG mapping and centralize layer logic

*SGManager* stores and handles Service Graphs

*VirtualResourceManager* contains the functionality tided to the layer's virtual view and virtual resources

*NFIBManager* handles the Network Function Information Base and hides implementation dependent logic

**Module contents** Contains classes relevant to Service Adaptation Sublayer functionality

**class** escape.service.sas_orchestration.**ServiceOrchestrator**(*layer_API*)
Bases: object

Main class for the actual Service Graph processing

**__init__**(*layer_API*)
Initialize main Service Layer components

**Parameters** **layer_API** (ServiceLayerAPI) – layer API instance

**Returns** None

**initiate_service_graph**(*sg*)
Main function for initiating Service Graphs

**Parameters** **sg** (NFFG) – service graph stored in NFFG instance

**Returns** NF-FG description

**Return type** *NFFG*

**class** escape.service.sas_orchestration.**SGManager**
Bases: object

Store, handle and organize Service Graphs

Currently it just stores SGs in one central place

**__init__**()
Init

**save**(*sg*)
Save SG in a dict

**Parameters** **sg** (NFFG) – Service Graph

**Returns** computed id of given Service Graph

**Return type** int

**get**(*graph_id*)
Return service graph with given id

**Parameters** **graph_id** (*int*) – graph ID

**Returns** stored Service Graph

**Return type** *NFFG*

**class** escape.service.sas_orchestration.**MissingVirtualViewEvent**
Bases: pox.lib.revent.revent.Event

Event for signaling missing virtual resource view

**class** escape.service.sas_orchestration.**VirtualResourceManager**
Bases: pox.lib.revent.revent.EventMixin

Support Service Graph mapping, follow the used virtual resources according to the Service Graph(s) in effect

Handles object derived from :class‘AbstractVirtualizer‘ and requested from lower layer

**_eventMixin_events = set([<class ‘escape.service.sas_orchestration.MissingVirtualViewEvent’>])**

**__init__**()
Initialize virtual resource manager

**Returns** None

**virtual_view**
Return resource info of actual layer as an *NFFG* instance

If it isn't exist requires it from Orchestration layer

**Returns** resource info as a Virtualizer

**Return type** *AbstractVirtualizer*

***escape.orchest* package** Subpackage for classes related to UNIFY's Resource Orchestration Sublayer (ROS)

**Submodules**

***policy_enforcement.py* module** *PolicyEnforcementError* represent a violation during the policy checking process

*PolicyEnforcementMetaClass* contains the main general logic which handles the Virtualizers and enforce policies

*PolicyEnforcement* implements the actual enforcement logic

**Module contents** Contains functionality related to policy enforcement

**exception** escape.orchest.policy_enforcement.**PolicyEnforcementError**
Bases: exceptions.RuntimeError

Exception class to signal policy enforcement error

**class** escape.orchest.policy_enforcement.**PolicyEnforcementMetaClass**
Bases: type

Meta class for handling policy enforcement in the context of classes inherited from *AbstractVirtualizer*

If the *PolicyEnforcement* class contains a function which name matches one in the actual Virtualizer then PolicyEnforcement's function will be called first.

**Warning:** Therefore the function names must be identical!

**Note:** If policy checking fails a *PolicyEnforcementError* should be raised and handled in a higher layer..

To use policy checking set the following class attribute:

```
__metaclass__ = PolicyEnforcementMetaClass
```

**static __new__**(*mcs*, *name*, *bases*, *attrs*)
Magic function called before subordinated class even created

**Parameters**

- **name** (*str*) – given class name
- **bases** (*tuple*) – bases of the class
- **attrs** (*dict*) – given attributes

**Returns** inferred class instance

**Return type** *AbstractVirtualizer*

**classmethod** **get_wrapper** (*mcs*, *orig_func*, *hooks*)
Return a decorator function which do the policy enforcement check

**Parameters**

- **orig_func** (*func*) – original function
- **hooks** (*tuple*) – tuple of pre and post checking functions

**Raise** PolicyEnforcementError

**Returns** decorator function

**Return type** func

**class** escape.orchest.policy_enforcement.**PolicyEnforcement**
Bases: object

Proxy class for policy checking

Contains the policy checking function

Binding is based on function name (checking function have to exist in this class and its name have to stand for the *pre_* or *post_* prefix and the name of the checked function)

**Warning:** Every PRE policy checking function is classmethod and need to have two parameter for nameless (args) and named(kwargs) params:

Example:

```
def pre_sanity_check (cls, args, kwargs):
```

**Warning:** Every POST policy checking function is classmethod and need to have three parameter for nameless (args), named (kwargs) params and return value:

Example:

```
def post_sanity_check (cls, args, kwargs, ret_value):
```

**Note:** The first element of args is the supervised Virtualizer ('self' param in the original function)

**__init__** ()
Init

**classmethod** **pre_sanity_check** (*args*, *kwargs*)
Implements the the sanity check before virtualizer's sanity check is called

**Parameters**

- **args** (*tuple*) – original nameless arguments
- **kwargs** (*dict*) – original named arguments

**Returns** None

**classmethod** **post_sanity_check** (*args*, *kwargs*, *ret_value*)
Implements the the sanity check after virtualizer's sanity check is called

**Parameters**

- **args** (*tuple*) – original nameless arguments

- **kwargs** (*dict*) – original named arguments
- **ret_value** – return value of Virtualizer's policy check function

**Returns** None

***ros_orchestration.py* module** *ResourceOrchestrator* orchestrates *NFFG* mapping and centralize layer logic *NFFGManager* stores and handles Network Function Forwarding Graphs

**Module contents** Contains classes relevant to Resource Orchestration Sublayer functionality

**class** escape.orchest.ros_orchestration.**ResourceOrchestrator**(*layer_API*)
Bases: object

Main class for the handling of the ROS-level mapping functions

**__init__**(*layer_API*)
Initialize main Resource Orchestration Layer components

**Parameters** **layer_API** (*ResourceOrchestrationAPI*) – layer API instance

**Returns** None

**instantiate_nffg**(*nffg*)
Main API function for NF-FG instantiation

**Parameters** **nffg** (NFFG) – NFFG instance

**Returns** mapped NFFG instance

**Return type** *NFFG*

**class** escape.orchest.ros_orchestration.**NFFGManager**
Bases: object

Store, handle and organize Network Function Forwarding Graphs

**__init__**()
Init

**save**(*nffg*)
Save NF-FG in a dict

**Parameters** **nffg** (NFFG) – Network Function Forwarding Graph

**Returns** generated ID of given NF-FG

**Return type** int

**get**(*nffg_id*)
Return NF-FG with given id

**Parameters** **nffg_id** (*int*) – ID of NF-FG

**Returns** NF-Fg instance

**Return type** *NFFG*

**class** escape.orchest.ros_orchestration.**NFIBManager**
Bases: object

Manage the handling of Network Function Information Base

**__init__**()
Init

**add**(*nf*)

**remove**(*nf_id*)

**getNF**(*nf_id*)

***ros_API.py* module** *InstallNFFGEvent* can send mapped NF-FG to the lower layer

*VirtResInfoEvent* can send back virtual resource info requested from upper layer

*GetGlobalResInfoEvent* can request global resource info from lower layer

*ResourceOrchestrationAPI* represents the ROS layer and implement all related functionality

**Module contents** Implements the platform and POX dependent logic for the Resource Orchestration Sublayer

**class** escape.orchest.ros_API.**InstallNFFGEvent**(*mapped_nffg*)
Bases: pox.lib.revent.revent.Event

Event for passing mapped *NFFG* to Controller Adaptation Sublayer

**__init__**(*mapped_nffg*)
Init

**Parameters** **mapped_nffg** (NFFG) – NF-FG graph need to be installed

**class** escape.orchest.ros_API.**VirtResInfoEvent**(*resource_info*)
Bases: pox.lib.revent.revent.Event

Event for sending back requested virtual resource info

**__init__**(*resource_info*)
Init

**Parameters** **resource_info** (ESCAPEVirtualizer) – virtual resource info

**class** escape.orchest.ros_API.**GetGlobalResInfoEvent**
Bases: pox.lib.revent.revent.Event

Event for requesting DomainVirtualizer from CAS

**class** escape.orchest.ros_API.**InstantiationFinishedEvent**(*success*, *error=None*)
Bases: pox.lib.revent.revent.Event

Event for signalling end of mapping process finished with success

**__init__**(*success*, *error=None*)

**class** escape.orchest.ros_API.**ResourceOrchestrationAPI**(*standalone=False*, ***kwargs*)
Bases: *escape.util.api.AbstractAPI*

Entry point for Resource Orchestration Sublayer (ROS)

Maintain the contact with other UNIFY layers

Implement the Sl - Or reference point

**_core_name = 'orchestration'**

**dependencies = ('adaptation',)**

**__init__**(*standalone=False*, ***kwargs*)

**See also:**

*AbstractAPI.__init__()*

**initialize**()

**See also:**

*AbstractAPI.initialze()*

**shutdown**(*event*)

**See also:**

*AbstractAPI.shutdown()*

**_handle_NFFGMappingFinishedEvent**(*event*)
Handle NFFGMappingFinishedEvent and proceed with *NFFG* installation

**Parameters** **event** (NFFGMappingFinishedEvent) – event object

**Returns** None

**_handle_InstantiateNFFGEvent**(**args*, ***kwargs*)
Instantiate given NF-FG (UNIFY Sl - Or API)

**Parameters** **event** (InstantiateNFFGEvent) – event object contains NF-FG

**Returns** None

**_install_NFFG**(*mapped_nffg*)
Send mapped *NFFG* to Controller Adaptation Sublayer in an implementation-specific way

General function which is used from microtask and Python thread also

**Parameters** **mapped_nffg** (NFFG) – mapped NF-FG

**Returns** None

**_handle_GetVirtResInfoEvent**(*event*)
Generate virtual resource info and send back to SAS

**Parameters** **event** (GetVirtResInfoEvent) – event object contains service layer id

**Returns** None

**_handle_MissingGlobalViewEvent**(*event*)
Request global resource info from CAS (UNIFY Or - CA API)

Invoked when a `MissingGlobalViewEvent` raised

**Parameters** **event** (MissingGlobalViewEvent) – event object

**Returns** None

**_handle_GlobalResInfoEvent**(*event*)
Save requested global resource info as the `DomainVirtualizer`

**Parameters** **event** (GlobalResInfoEvent) – event object contains resource info

**Returns** None

**_handle_InstallationFinishedEvent**(*event*)

***ros_mapping.py* module** *ESCAPEMappingStrategy* implements a default *NFFG* mapping algorithm of ESCAPEv2

*ResourceOrchestrationMapper* perform the supplementary tasks for *NFFG* mapping

**Module contents** Contains classes which implement *NFFG* mapping functionality

**class** escape.orchest.ros_mapping.**ESCAPEMappingStrategy**
Bases: *escape.util.mapping.AbstractMappingStrategy*

Implement a strategy to map initial *NFFG* into extended *NFFG*

**__init__**()
Init

**classmethod map**(*graph*, *resource*)
Default mapping algorithm of ESCAPEv2

**Parameters**

- **graph** (NFFG) – Network Function forwarding Graph
- **resource** (NFFG) – global virtual resource info

**Returns** mapped Network Function Forwarding Graph

**Return type** *NFFG*

**class** escape.orchest.ros_mapping.**NFFGMappingFinishedEvent**(*nffg*)
Bases: pox.lib.revent.revent.Event

Event for signaling the end of NF-FG mapping

**__init__**(*nffg*)
Init

**Parameters** **nffg** (NFFG) – NF-FG need to be installed

**class** escape.orchest.ros_mapping.**ResourceOrchestrationMapper**
Bases: *escape.util.mapping.AbstractMapper*

Helper class for mapping NF-FG on global virtual view

**_eventMixin_events = set([<class 'escape.orchest.ros_mapping.NFFGMappingFinishedEvent'>])**

**__init__**()
Init Resource Orchestrator mapper

**Returns** None

**orchestrate**(*input_graph*, *resource_view*)
Orchestrate mapping of given NF-FG on given global resource

**Parameters**

- **input_graph** (NFFG) – Network Function Forwarding Graph
- **resource_view** (DomainVirtualizer) – global resource view

**Returns** mapped Network Function Forwarding Graph

**Return type** *NFFG*

**_mapping_finished**(*nffg*)
Called from a separate thread when the mapping process is finished

**Parameters** **nffg** (NFFG) – mapped NF-FG

**Returns** None

***virtualization_mgmt.py* module** *AbstractVirtualizer* contains the central logic of Virtualizers

*ESCAPEVirtualizer* implement the standard virtualization logic of the Resource Orchestration Sublayer

*VirtualizerManager* stores and handles the virtualizers

**Module contents** Contains components relevant to virtualization of resources and views

**class** escape.orchest.virtualization_mgmt.**AbstractVirtualizer**
Bases: object

Abstract class for actual Virtualizers

Follows the Proxy design pattern

**__metaclass__**
alias of PolicyEnforcementMetaClass

**__init__**()
Init

**get_resource_info**()
Hides object's mechanism and return with a resource object derived from *NFFG*

**Warning:** Derived class have to override this function

**Raise** NotImplementedError

**Returns** resource info

**Return type** *NFFG*

**sanity_check**(**args*, ***kwargs*)
Place-holder for sanity check which implemented in PolicyEnforcement

**Parameters** **nffg** (NFFG) – NFFG instance

**Returns** None

**class** escape.orchest.virtualization_mgmt.**ESCAPEVirtualizer**
Bases: *escape.orchest.virtualization_mgmt.AbstractVirtualizer*

Actual Virtualizer class for ESCAPEv2

**__init__**()
Init

**get_resource_info**()
Hides object's mechanism and return with a resource object derived from *NFFG*

**Returns** virtual resource info

**Return type** *NFFG*

**sanity_check**(**args*, ***kwargs*)
Placeholder method for policy checking.

Return the virtual resource info for the post checker function

**Returns** virtual resource info

**Return type** *NFFG*

**_generate_resource_info**()
Private method to return with resouce info

**class** escape.orchest.virtualization_mgmt.**MissingGlobalViewEvent**
Bases: pox.lib.revent.revent.Event

Event for signaling missing global resource view

**class** escape.orchest.virtualization_mgmt.**VirtualizerManager**
Bases: pox.lib.revent.revent.EventMixin

Store, handle and organize instances of derived classes of *AbstractVirtualizer*

**_eventMixin_events = set([<class 'escape.orchest.virtualization_mgmt.MissingGlobalViewEvent'>])**

**__init__**()
Initialize virtualizer manager

**Returns** None

**dov**
Getter method for the DomainVirtualizer

Request DoV from Adaptation if it hasn't set yet

Use: *virtualizerManager.dov*

**Returns** Domain Virtualizer (DoV)

**Return type** *DomainVirtualizer*

**get_virtual_view**(*layer_id*)
Return the virtual view as a derived class of *AbstractVirtualizer*

**Parameters** **layer_id** (*int*) – layer ID

**Returns** virtual view

**Return type** *ESCAPEVirtualizer*

**_generate_virtual_view**(*layer_id*)
Generate a missing *ESCAPEVirtualizer* for other layer using global view (DoV) and a given layer id

**Parameters** **layer_id** (*int*) – layer ID

**Returns** generated Virtualizer derived from AbstractVirtualizer

**Return type** *ESCAPEVirtualizer*

***escape.adapt* package** Sublayer for classes related to UNIFY's Controller Adaptation Sublayer (CAS)

### Submodules

***adaptation.py* module** *ControllerAdapter* implements the centralized functionality of high-level adaptation and installation of *NFFG*

*DomainVirtualizer* implement the standard virtualization/generalization logic of the Resource Orchestration Sublayer

*DomainResourceManager* stores and handles the global Virtualizer

**Module contents** Contains classes relevant to the main adaptation function of the Controller Adaptation Sublayer

**class** escape.adapt.adaptation.**DomainConfigurator**(*ca*, *lazy_load=True*, *remote=True*)
Bases: object

Initialize, configure and store Domain Manager objects

Use global config to create managers and adapters

Follows Componenet Configurator design pattern

**__init__**(*ca*, *lazy_load=True*, *remote=True*)
For domain adapters the configurator checks the CONFIG first.

> **Warning:** Adapter classes must be subclass of AbstractDomainAdapter

> **Note:** Arbitrary domain adapters is searched in *escape.adapt.domain_adapters*

**Parameters**

- **ca** (*ControllerAdapter*) – ControllerAdapter instance
- **lazy_load** (*bool*) – load adapters only at first reference (default: True)
- **remote** (*bool*) – use NETCONF RPCs or direct access (default: True)

**get**(*domain_name*)
Get Domain maganger with given name.

**Parameters** **domain_name** (*str*) – name of domain manager

**Returns** None

**start**(*domain_name*)
Initialize and start a Domain manager.

**Parameters** **domain_name** (*str*) – name of domain manager

**Returns** None

**stop**(*domain_name*)
Stop and derefer a Domain manager.

**Parameters** **domain_name** (*str*) – name of domain manager

**Returns** None

**components**
Return the dict of initiated Domain managers.

**Returns** managers

**Return type** dict

**__iter__**()
Return with an iterator rely on initiated managers

**load_default_mgrs**()
Init default adapters.

**load_internal_mgr**(*remote=True*)
Init Domain Manager for internal domain.

**Parameters** **remote** (*bool*) – use NETCONF RPCs or direct access (default: True)

**Returns** None

**_DomainConfigurator__load_component**(*component_name*, *from_config=True*, ***kwargs*)
Load given component from config.

**Parameters**

- **component_name** (*str*) – adapter's name
- **kwargs** (*dict*) – adapter's initial parameters

**Returns** initiated adapter

**Return type** *AbstractDomainAdapter*

**class** escape.adapt.adaptation.**ControllerAdapter**(*layer_API*, *with_infr=False*)
Bases: object

Higher-level class for *NFFG* adaptation between multiple domains

**__init__**(*layer_API*, *with_infr=False*)
Initialize Controller adapter

For domain adapters the ControllerAdapter checks the CONFIG first If there is no adapter defined explicitly then initialize the default Adapter class stored in *_defaults*

**Warning:** Adapter classes must be subclass of AbstractDomainAdapter

**Note:** Arbitrary domain adapters is searched in *escape.adapt.domain_adapters*

**Parameters**

- **layer_API** (*ControllerAdaptationAPI*) – layer API instance
- **with_infr** (*bool*) – using emulated infrastructure (default: False)

**install_nffg**(*mapped_nffg*)
Start NF-FG installation

Process given *NFFG*, slice information based on domains an invoke domain adapters to install domain specific parts

**Parameters** **mapped_nffg** (NFFG) – mapped NF-FG instance which need to be installed

**Returns** None or internal domain NFFG part

**_handle_DomainChangedEvent**(*event*)
Handle DomainChangedEvents, process changes and store relevant information in DomainResourceManager

**_slice_into_domains**(*nffg*)
Slice given *NFFG* into separate parts

**Parameters** **nffg** (NFFG) – mapped NFFG object

**Returns** sliced parts

**Return type** dict

**class** escape.adapt.adaptation.**DomainVirtualizer**(*domainResManager*)
Bases: *escape.orchest.virtualization_mgmt.AbstractVirtualizer*

Specific Virtualizer class for global domain virtualization

Implement the same interface as *AbstractVirtualizer*

**__init__**(*domainResManager*)
Init

**Parameters** **domainResManager** (DomainResourceManager) – domain resource manager

**Returns** None

**get_resource_info**()
Return the global resource info represented this class

**Returns** global resource info

**Return type** *NFFG*

**class** escape.adapt.adaptation.**DomainResourceManager**
Bases: object

Handle and store global resources

**__init__**()
Init

**dov**
Getter for *DomainVirtualizer*

**Returns** Domain Virtualizer

**Return type** *ESCAPEVirtualizer*

**update_resource_usage**(*data*)
Update global resource database with resource usage relevant to installed components, routes, VNFs, etc.

**Parameters** **data** (*dict*) – usage data

**Returns** None

***cas_API.py* module** *GlobalResInfoEvent* can send back global resource info requested from upper layer

*ControllerAdaptationAPI* represents the CAS layer and implement all related functionality

**Module contents** Implements the platform and POX dependent logic for the Controller Adaptation Sublayer

**class** escape.adapt.cas_API.**GlobalResInfoEvent**(*resource_info*)
Bases: pox.lib.revent.revent.Event

Event for sending back requested global resource info

**__init__**(*resource_info*)
Init

**Parameters** **resource_info** (*ESCAPEVirtualizer*) – resource info

**class** escape.adapt.cas_API.**InstallationFinishedEvent**(*success*, *error=None*)
Bases: pox.lib.revent.revent.Event

Event for signalling end of mapping process

**__init__**(*success*, *error=None*)

**class** escape.adapt.cas_API.**DeployNFFGEvent**(*nffg_part*)
Bases: pox.lib.revent.revent.Event

Event for passing mapped *NFFG* to internally emulated network based on Mininet for testing

**__init__**(*nffg_part*)

**class** escape.adapt.cas_API.**ControllerAdaptationAPI**(*standalone=False*, ***kwargs*)
Bases: *escape.util.api.AbstractAPI*

Entry point for Controller Adaptation Sublayer (CAS)

Maintain the contact with other UNIFY layers

Implement the Or - Ca reference point

**_core_name = 'adaptation'**

**__init__**(*standalone=False*, ***kwargs*)

**See also:**

*AbstractAPI.__init__()*

**initialize**()

**See also:**

*AbstractAPI.initialze()*

**shutdown**(*event*)

**See also:**

*AbstractAPI.shutdown()*

**_handle_InstallNFFGEvent**(**args*, ***kwargs*)
Install mapped NF-FG (UNIFY Or - Ca API)

**Parameters** **event** (*InstallNFFGEvent*) – event object contains mapped NF-FG

**Returns** None

**_handle_GetGlobalResInfoEvent**(*event*)
Generate global resource info and send back to ROS

**Parameters** **event** (*GetGlobalResInfoEvent*) – event object

**Returns** None

**_handle_DeployEvent**(*event*)
Receive processed NF-FG from domain adapter(s) and forward to Infrastructure

**Parameters** **event** (*DeployNFFGEvent*) – event object

**Returns** None

**_handle_DeploymentFinishedEvent**(*event*)
Receive successfull NF-FG deployment event and propagate upwards

**Parameters** **event** (*DeploymentFinishedEvent*) – event object

**Returns** None

***domain_adapters.py* module** *POXDomainAdapter* implements POX related functionality.

*MininetDomainAdapter* implements Mininet related functionality transparently.

*VNFStarterAdapter* is a wrapper class for vnf_starter NETCONF module.

*OpenStackRESTAdapter* is a wrapper class for OpenStack-related functions.

*InternalDomainManager* represent the top class for interacting with emulated infrastructure.

*OpenStackDomainManager* implements OpenStack related functionality.

*DockerDomainManager* implements Docker related functionality.

**Module contents** Contains Adapter classes which represent the connections between ESCAPEv2 and other different domains

**class** escape.adapt.domain_adapters.**POXDomainAdapter**(*name=None*, *address='127.0.0.1'*, *port=6653*)

Bases: *escape.util.adapter.AbstractDomainAdapter*

Adapter class to handle communication with internal POX OpenFlow controller

Can be used to define a controller (based on POX) for other external domains

**name = 'POX'**

**__init__**(*name=None*, *address='127.0.0.1'*, *port=6653*)

Initialize attributes, register specific connection Arbiter if needed and set up listening of OpenFlow events.

**Parameters**

- **name** (*str*) – name used to register component ito pox.core
- **address** (*str*) – socket address (default: 127.0.0.1)
- **port** (*int*) – socket port (default: 6653)

**filter_connections**(*event*)

Handle which connection should be handled by this Adapter class.

This adapter accept every OpenFlow connection by default.

**Parameters** **event** (pox.openflow.ConnectionUp) – POX internal ConnectionUp event (event.dpid, event.connection)

**Returns** True os False obviously

**Return type** bool

**_handle_ConnectionUp**(*event*)

Handle incoming OpenFlow connections

**_handle_ConnectionDown**(*event*)

Handle disconnected device

**install_routes**(*routes*)

Install routes related to the managed domain. Translates the generic format of the routes into OpenFlow flow rules.

Routes are computed by the ControllerAdapter's main adaptation algorithm

**Parameters** **routes** (*NFFG*) – list of routes

**Returns** None

**class** escape.adapt.domain_adapters.**MininetDomainAdapter**(*mininet=None*)

Bases: *escape.util.adapter.AbstractDomainAdapter*, *escape.util.adapter.VNFStarterAPI*

Adapter class to handle communication with Mininet domain

Implement VNF managing API using direct access to the mininet.net.Mininet object

**_eventMixin_events = set([<class 'escape.util.adapter.DeployEvent'>, <class 'escape.util.adapter.DomainChangedEv**

**name = 'MININET'**

**__init__**(*mininet=None*)
Init

**Parameters** **mininet** (*:any'mininet.net.Mininet'*) – set pre-defined network (optional)

**initiate_VNFs**(*nffg_part*)

**stopVNF**(*vnf_id*)

**getVNFInfo**(*vnf_id=None*)

**disconnectVNF**(*vnf_id*, *vnf_port*)

**startVNF**(*vnf_id*)

**connectVNF**(*vnf_id*, *vnf_port*, *switch_id*)

**initiateVNF**(*vnf_type=None*, *vnf_description=None*, *options=None*)

**class** escape.adapt.domain_adapters.**VNFStarterAdapter**(***kwargs*)
Bases: *escape.util.netconf.AbstractNETCONFAdapter*, *escape.util.adapter.AbstractDomainAdapt*
*escape.util.adapter.VNFStarterAPI*

This class is devoted to provide NETCONF specific functions for vnf_starter module. Documentation is transferred from vnf_starter.yang

This class is devoted to start and stop CLICK-based VNFs that will be connected to a mininet switch.

Follows the MixIn design patteran approach to support NETCONF functionality

**RPC_NAMESPACE = u'http://csikor.tmit.bme.hu/netconf/unify/vnf_starter'**

**name = 'VNFStarter'**

**__init__**(***kwargs*)

**initiateVNF**(*vnf_type=None*, *vnf_description=None*, *options=None*)
This RCP will start a VNF.

0.initiate new VNF (initiate datastructure, generate unique ID)

1.set its arguments (control port, control ip, and VNF type/command)

2.returns the connection data, which from the vnf_id is the most important

**Parameters**

- **vnf_type** (*str*) – pre-defined VNF type (see in vnf_starter/available_vnfs)
- **vnf_description** (*str*) – Click description if there are no pre-defined type
- **options** (*collections.OrderedDict*) – unlimited list of additional options as name-value pairs

**Returns** RPC reply data

**Raises** RPCError, OperationError, TransportError

**connectVNF**(*vnf_id*, *vnf_port*, *switch_id*)
This RPC will practically start and connect the initiated VNF/CLICK to the switch.

0.create virtualEthernet pair(s)

1.connect either end of it (them) to the given switch(es)

This RPC is also used for reconnecting a VNF. In this case, however, if the input fields are not correctly set an error occurs

**Parameters**

- **vnf_id** (*str*) – VNF ID (mandatory)
- **vnf_port** (*str*) – VNF port (mandatory)
- **switch_id** (*str*) – switch ID (mandatory)

**Returns** Returns the connected port(s) with the corresponding switch(es).

**Raises** RPCError, OperationError, TransportError

**disconnectVNF**(*vnf_id*, *vnf_port*)

This RPC will disconnect the VNF(s)/CLICK(s) from the switch(es).

0.ip link set uny_0 down

1.ip link set uny_1 down

2.(if more ports) repeat 1. and 2. with the corresponding data

**Parameters**

- **vnf_id** (*str*) – VNF ID (mandatory)
- **vnf_port** (*str*) – VNF port (mandatory)

**Returns** reply data

**Raises** RPCError, OperationError, TransportError

**startVNF**(*vnf_id*)

This RPC will actually start the VNF/CLICK instance.

**Parameters** **vnf_id** (*str*) – VNF ID (mandatory)

**Returns** reply data

**Raises** RPCError, OperationError, TransportError

**stopVNF**(*vnf_id*)

This RPC will gracefully shut down the VNF/CLICK instance.

0.if disconnect() was not called before, we call it

1.delete virtual ethernet pairs

2.stop (kill) click

3.remove vnf's data from the data structure

**Parameters** **vnf_id** (*str*) – VNF ID (mandatory)

**Returns** reply data

**Raises** RPCError, OperationError, TransportError

**getVNFInfo**(*vnf_id=None*)

This RPC will send back all data of all VNFs that have been initiated by this NETCONF Agent. If an input of vnf_id is set, only that VNF's data will be sent back. Most of the data this RPC replies is used for DEBUG, however 'status' is useful for indicating to upper layers whether a VNF is UP_AND_RUNNING

**Parameters** **vnf_id** (*str*) – VNF ID

**Returns** reply data

**Raises** RPCError, OperationError, TransportError

**class** escape.adapt.domain_adapters.**OpenStackRESTAdapter**(*url*)
Bases: *escape.util.adapter.AbstractRESTAdapter*, *escape.util.adapter.AbstractDomainAdapter*, *escape.util.adapter.OpenStackAPI*

**__init__**(*url*)
Init

**Parameters** **url** (*str*) – OpenStack RESTful API URL

**class** escape.adapt.domain_adapters.**InternalDomainManager**(*controller=None*, *network=None*, *remote=None*)
Bases: *escape.util.adapter.AbstractDomainManager*

Manager class to handle communication with internally emulated network

---

**Note:** Uses *MininetDomainAdapter* for managing the emulated network and *POXDomainAdapter* for controlling the network

---

**name = 'INTERNAL'**

**__init__**(*controller=None*, *network=None*, *remote=None*)
Init

**controller_name**

**install_nffg**(*nffg_part*)
Install an *NFFG* related to the internal domain

Split given *NFFG* to a set of NFs need to be initiated and a set of routes/connections between the NFs

**Parameters** **nffg_part** (*NFFG*) – NF-FG need to be deployed

**Returns** None

**class** escape.adapt.domain_adapters.**OpenStackDomainManager**(*url*)
Bases: *escape.util.adapter.AbstractDomainManager*

Adapter class to handle communication with OpenStack domain

**Warning:** Not implemented yet!

**name = 'OPENSTACK'**

**__init__**(*url*)
Init

**Parameters** **url** (*str*) – OpenStack RESTful API URL

**install_nffg**(*nffg_part*)

**class** escape.adapt.domain_adapters.**DockerDomainManager**
Bases: *escape.util.adapter.AbstractDomainManager*

Adapter class to handle communication component in a Docker domain

**Warning:** Not implemented yet!

**name = 'DOCKER'**

**__init__**()
Init

**install_nffg**(*nffg_part*)

***escape.infr* package** Sublayer for classes related to UNIFY's Infrastructure Layer (IL)

**Submodules**

***il_API.py* module** *InfrastructureLayerAPI* represents the IL layer and implement all related functionality

**Module contents** Emulate UNIFY's Infrastructure Layer for testing purposes based on Mininet

**class** escape.infr.il_API.**DeploymentFinishedEvent**(*success*, *error=None*)
Bases: pox.lib.revent.revent.Event

Event for signaling NF-FG deployment

**__init__**(*success*, *error=None*)

**class** escape.infr.il_API.**InfrastructureLayerAPI**(*standalone=False*, ***kwargs*)
Bases: *escape.util.api.AbstractAPI*

Entry point for Infrastructure Layer (IL)

Maintain the contact with other UNIFY layers

Implement a specific part of the Co - Rm reference point

**_core_name = 'infrastructure'**

**_eventMixin_events = set([<class 'escape.infr.il_API.DeploymentFinishedEvent'>])**

**__init__**(*standalone=False*, ***kwargs*)

**See also:**

*AbstractAPI.__init__()*

**initialize**()

**See also:**

*AbstractAPI.initialize()*

**shutdown**(*event*)

**See also:**

*AbstractAPI.shutdown()*

**_handle_ComponentRegistered**(*event*)
Wait for controller (internal POX module)

**Parameters** **event** (ComponentRegistered) – registered component event

**Returns** None

**_handle_DeployNFFGEvent**(**args*, ***kwargs*)
Install mapped NFFG part into the emulated network

:param event:event object :return: *DeployNFFGEvent*

**install_route**()

***topology.py* module** *InternalControllerProxy* represents the connection between the internal controller and the emulated network

*AbstractTopology* represents the emulated topology for the high-level API

**Module contents** Wrapper module for handling emulated test topology based on Mininet

**class** escape.infr.topology.**AbstractTopology**(*hopts=None*, *sopts=None*, *lopts=None*, *eopts=None*)
Bases: mininet.topo.Topo

Abstract class for representing emulated topology.

Have the functions to build a ESCAPE-specific topology.

Can be used to define reusable topology similar to Mininet's high-level API. Reusable, convenient and predefined way to define a topology, but less flexible and powerful.

**default_host_opts = None**

**default_switch_opts = None**

**default_link_opts = None**

**default_EE_opts = None**

**__init__**(*hopts=None*, *sopts=None*, *lopts=None*, *eopts=None*)

**class** escape.infr.topology.**BackupTopology**
Bases: *escape.infr.topology.AbstractTopology*

Topology class for testing purposes and serve as a fallback topology

**__init__**()
Init and build test topology

**class** escape.infr.topology.**InternalControllerProxy**(*name='InternalPOXController'*, *ip='127.0.0.1'*, *port=6653*, ***kwargs*)
Bases: mininet.node.RemoteController

Controller class for emulated Mininet network. Making connection with internal controller initiated by POXDomainAdapter.

**__init__**(*name='InternalPOXController'*, *ip='127.0.0.1'*, *port=6653*, ***kwargs*)
Init

**Parameters**

- **name** (*str*) – name of the controller (default: InternalPOXController)
- **ip** (*str*) – IP address (default: 127.0.0.1)
- **port** (*int*) – port number (default 6633)

**checkListening**()
Check the controller port is open

**class** escape.infr.topology.**ESCAPENetworkBridge**(*network=None*)
Bases: object

Internal class for representing the emulated topology.

Represents a container class for network elements such as switches, nodes, execution environments, links etc. Contains network management functions similar to Mininet's mid-level API extended with ESCAPEv2 related capabilities

Separate the interface using internally from original Mininet object to implement loose coupling and avoid changes caused by Mininet API changes e.g. 2.1.0 -> 2.2.0

Follows Bridge design pattern.

**__init__**(*network=None*)
Initialize Mininet implementation with proper attributes.

**network**
Internal network representation

**Returns** network representation

**Return type** mininet.net.Mininet

**start_network**()
Start network

**stop_network**()
Stop network

**cleanup**()
Clean up junk which might be left over from old runs.

**..seealso::** mininet.clean.cleanup()

**exception** escape.infr.topology.**TopologyBuilderException**
Bases: exceptions.Exception

Exception class for topology errors.

**class** escape.infr.topology.**ESCAPENetworkBuilder**(*net=None*, *opts=None*, *run_dry=True*)
Bases: object

Builder class for topology.

Update the network object based on the parameters if it's given or create an empty instance.

Always return with an ESCAPENetworkBridge instance which offer a generic interface for created :any::*Mininet* object and hide implementation's nature.

Follows Builder design pattern.

**default_opts = {'listenPort': None, 'autoSetMacs': True, 'inNamespace': False, 'autoStaticArp': True, 'controller': <**

**topology_config_name = 'FALLBACK-TOPO'**

**__init__**(*net=None*, *opts=None*, *run_dry=True*)
Initialize NetworkBuilder.

If the topology definition is not found, an exception will be raised or an empty :any::*Mininet* topology will be created if run_dry is set.

**Parameters**

- **net** (:any::*Mininet*) – update given Mininet object instead of creating a new one
- **opts** (*dict*) – update default options with the given opts

- **run_dry** (*bool*) – do not raise an Exception and return with bare Mininet obj.

**_ESCAPENetworkBuilder__init_from_AbstractTopology**(*topo_class*)
Build topology from pre-defined Topology class.

**Parameters** **topo** (*AbstractTopology*) – topology

**Returns** None

**_ESCAPENetworkBuilder__init_from_CONFIG**()
Build a pre-defined topology stored in CONFIG.

**Returns** None

**_ESCAPENetworkBuilder__init_from_NFFG**(*net*, *nffg*)
Initialize topology from *NFFG*

**Parameters** **nffg** (*NFFG*) – topology

**Returns** None

**_ESCAPENetworkBuilder__init_from_dict**(*dict*)
Initialize topology from a dictionary.

Keywords for network elements: controllers, ee, saps, switches, links

Option keywords: netopts

**Parameters** **dict** (*NFFG*) – topology

**Returns** None

**_ESCAPENetworkBuilder__init_from_file**(*path='escape.topo'*)
Build a pre-defined topology stored in a file in JSON format.

**Parameters** **path** (*str*) – file path

**Returns** None

**build**(*topology=None*)
Initialize network

**Parameters** **topology** (*NFFG* or Dictionary displays or *AbstractTopology*) – topology representation

**Returns** None

***escape.util* package** Additional functions, classes, components

### Submodules

***adapter.py* module** *DomainChangedEvent* signals changes for *ControllerAdapter* in an unified way.

*AbstractDomainManager* contains general logic for top domain managers

*AbstractDomainAdapter* contains general logic for actual Adapters.

*VNFStarterAPI* defines the interface for VNF management based on VNFStarter YANG description.

*OpenStackAPI* defines the interface for communication with OpenStack domain.

Requirements:

```
sudo pip install requests
```

*AbstractRESTAdapter* contains the general functions for communication through an HTTP/RESTful API

**Module contents** Implement the supporting classes for doamin adapters

**class** escape.util.adapter.**DomainChangedEvent**(*domain*, *cause*, *data=None*)
Bases: pox.lib.revent.revent.Event

Event class for signaling all kind of change(s) in specific domain

This event's purpose is to hide the domain specific operations and give a general and unified way to signal domain changes to ControllerAdapter in order to handle all the changes in the same function/algorithm

**type**
alias of enum

**__init__**(*domain*, *cause*, *data=None*)
Init event object

**Parameters**

- **domain** (*str*) – domain name. Should be *AbstractDomainAdapter.name*
- **cause** (*str*) – type of the domain change: *DomainChangedEvent.type*
- **data** (*object*) – data connected to the change (optional)

**Returns** None

**class** escape.util.adapter.**DeployEvent**(*nffg_part*)
Bases: pox.lib.revent.revent.Event

Event class for signaling NF-FG deployment to infrastructure layer API

Used by DirectMininetAdapter for internal NF-FG deployment

**__init__**(*nffg_part*)

**class** escape.util.adapter.**AbstractDomainManager**
Bases: pox.lib.revent.revent.EventMixin

Abstract class for different domain managers

Domain managers is top level classes to handle and manage domains transparently

Follows the MixIn design pattern approach to support general manager functionality for topmost Controller-Adapter class

Follows the Component Configurator design pattern as base component class

**init**()
Abstract function for component initialization

**run**()
Abstract function for starting component

**finit**()
Abstract function for starting component

**suspend**()
Abstract class for suspending a running component

**resume**()
Abstract function for resuming a suspended component

**info**()
Abstract function for requesting information about the component

**install_nffg**(*nffg_part*)
Install an *NFFG* related to the specific domain

**Parameters** **nffg_part** (*NFFG*) – NF-FG need to be deployed

**Returns** None

**class** escape.util.adapter.**AbstractDomainAdapter**
Bases: pox.lib.revent.revent.EventMixin

Abstract class for different domain adapters.

Domain adapters can handle domains as a whole or well-separated parts of a domain e.g. control part of an SDN network, infrastructure containers or other entities through a specific protocol (NETCONF, HTTP/REST).

Follows the Adapter design pattern (Adaptor base class).

Follows the MixIn design patteran approach to support general adapter functionality for manager classes mostly.

**_eventMixin_events = set([<class 'escape.util.adapter.DomainChangedEvent'>])**

**name = None**

**__init__**()
Init

**start_polling**(*wait=1*)
Initialize and start a Timer co-op task for polling.

**Parameters** **wait** (*int*) – polling period (default: 1)

**stop_polling**()
Stop timer.

**poll**()
Template fuction to poll domain state. Called by a Timer co-op multitask. If the function return with False the timer will be cancelled.

**class** escape.util.adapter.**VNFStarterAPI**
Bases: object

Define interface for managing VNFs.

**See also:**

vnf_starter.yang

Follows the MixIn design pattern approach to support VNFStarter functionality.

**__init__**()

**initiateVNF**(*vnf_type=None*, *vnf_description=None*, *options=None*)
Initiate a VNF.

**Parameters**

- **vnf_type** (*str*) – pre-defined VNF type (see in vnf_starter/available_vnfs)
- **vnf_description** (*str*) – Click description if there are no pre-defined type
- **options** (*collections.OrderedDict*) – unlimited list of additional options as name-value pairs

**connectVNF** (*vnf_id*, *vnf_port*, *switch_id*)
Connect a VNF to a switch.

**Parameters**

- **vnf_id** (*str*) – VNF ID (mandatory)
- **vnf_port** (*str*) – VNF port (mandatory)
- **switch_id** (*str*) – switch ID (mandatory)

**Returns** Returns the connected port(s) with the corresponding switch(es).

**disconnectVNF** (*vnf_id*, *vnf_port*)
Disconnect VNF from a switch.

**Parameters**

- **vnf_id** (*str*) – VNF ID (mandatory)
- **vnf_port** (*str*) – VNF port (mandatory)

**Returns** reply data

**startVNF** (*vnf_id*)
Start VNF.

**Parameters** **vnf_id** (*str*) – VNF ID (mandatory)

**Returns** reply data

**stopVNF** (*vnf_id*)
Stop VNF.

**Parameters** **vnf_id** (*str*) – VNF ID (mandatory)

**Returns** reply data

**getVNFInfo** (*vnf_id=None*)

**class** escape.util.adapter.**OpenStackAPI**
Bases: object

Define interface for managing OpenStack domain.

**Note:** Based on separated REST API which need to be discussed!

Follows the MixIn design pattern approach to support OpenStack functionality.

**class** escape.util.adapter.**AbstractRESTAdapter** (*base_url*, *auth=None*)
Bases: requests.sessions.Session

Abstract class for various adapters rely on a RESTful API.

Contains basic functions for managing connections.

Inhereted from requests.Session. Provided features: coockie persistence, connection-pooling and configuration.

**Implements Context Manager Python protocol::**

```
>>> with AbstractRESTAdapter as a:
>>>   a.<method>()
```

**See also:**

http://docs.python-requests.org/en/latest/api/#requests.Session

Follows Adapter design pattern.

**custom_headers = {'user-agent': 'ESCAPE/2.0.0'}**

**__init__**(*base_url*, *auth=None*)

**_send_request**(*method*, *url=None*, *body=None*, ***kwargs*)

Prepare the request and send it. If valid URL is given that value will be used else it will be append to the end of the `base_url`. If `url` is not given only the `base_url` will be used.

**Parameters**

- **method** (*str*) – HTTP method
- **url** (*str*) – valid URL or relevent part follows `self.base_url`
- **body** (*NFFG* or dict or bytes or str) – request body
- **kwargs** – additional params. See `requests.Session.request`

**Returns** response text as JSON

**Return type** str

**Raises**

- **HTTPError** – if responde code is between 400 and 600
- **ConnectionError** – connection error
- **Timeout** – many error occured when request timed out

***api.py* module** *AbstractAPI* contains the register mechanism into the POX core for layer APIs, the event handling/registering logic and defines the general functions for initialization and finalization steps

*RESTServer* is a general HTTP server which parse HTTP request and forward to explicitly given request handler

*AbstractRequestHandler* is a base class for concrete request handling. It implements the general URL and request body parsing functions

**Module contents** Contains abstract classes for concrete layer API modules

**class** `escape.util.api.`**AbstractAPI**(*standalone=False*, ***kwargs*)

Bases: `pox.lib.revent.revent.EventMixin`

Abstract class for UNIFY's API

Contain common functions

Follows Facade design pattern -> simplified enty/exit point ot the layers

**_core_name = 'AbstractAPI'**

**dependencies = ()**

**__init__**(*standalone=False*, ***kwargs*)

Abstract class constructor

Handle core registration along with *_all_dependencies_met()*

Set given parameters (standalone parameter is mandatory) automatically as:

```
self._<param_name> = <param_value>
```

Base constructor functions have to be called as the last step in derived classes. Same situation with *_all_dependencies_met()* respectively. Must not override these function, just use *initialize()* for init steps. Actual API classes must only call `super()` in their constructor with the form:

```
super(<API Class name>, self).__init__(standalone=standalone, **kwargs)
```

**Warning:** Do not use prefixes in the name of event handlers, because of automatic dependency discovery considers that as a dependent component and this situation cause a dead lock (component will be waiting to each other to set up)

**Parameters** **standalone** (*bool*) – started in standalone mode or not

**_all_dependencies_met**()
Called when every component on which depends are initialized on POX core

Contain dependency relevant initialization.

**Returns** None

**initialize**()
Init function for child API classes to simplify dynamic initialization

Called when every component on which depends are initialized and registered in POX core

This function should be overwritten by child classes.

**Returns** None

**shutdown**(*event*)
Finalization, deallocation, etc. of actual component

Should be overwritten by child classes

**Parameters** **event** (*GoingDownEvent*) – shutdown event raised by POX core

**Returns** None

**static _read_json_from_file**(*graph_file*)
Read the given file and return a string formatted as JSON

**Parameters** **graph_file** (*str*) – file path

**Returns** JSON data

**Return type** str

**__str__**()
Print class type and non-private attributes with their types for debugging

**Returns** specific string

**Return type** str

**class** escape.util.api.**RESTServer**(*RequestHandlerClass*, *address='127.0.0.1'*, *port=8008*)
Bases: BaseHTTPServer.HTTPServer, SocketServer.ThreadingMixIn

Base HTTP server for RESTful API

Initiate an HTTPServer and run it in different thread

**__init__**(*RequestHandlerClass*, *address='127.0.0.1'*, *port=8008*)
Set up an BaseHTTPServer.HTTPServer in a different thread

**Parameters**

- **RequestHandlerClass** (AbstractRequestHandler) – Class of a handler which handles HTTP request
- **address** (*str*) – Used IP address
- **port** (*int*) – Used port number

**start**()
Start RESTServer thread

**stop**()
Stop RESTServer thread

**run**()
Handle one request at a time until shutdown.

**exception** escape.util.api.**RESTError**(*msg=None*, *code=0*)
Bases: exceptions.Exception

Exception class for REST errors

**__init__**(*msg=None*, *code=0*)

**msg**

**code**

**__str__**()

**class** escape.util.api.**AbstractRequestHandler**(*request*, *client_address*, *server*)
Bases: BaseHTTPServer.BaseHTTPRequestHandler

Minimalistic RESTful API for Layer APIs

Handle /escape/* URLs.

Method calling permissions represented in escape_intf dictionary

> **Warning:** This class is out of the context of the recoco's co-operative thread context! While you don't need to worry much about synchronization between recoco tasks, you do need to think about synchronization between recoco task and normal threads. Synchronisation is needed to take care manually - use relevant helper function of core object: callLater() / raiseLater() or use *schedule_as_coop_task()* decorator defined in *escape.util.misc* on the called function

**server_version = 'ESCAPE/2.0.0'**

**static_prefix = 'escape'**

**request_perm = {'PUT': (), 'POST': (), 'DELETE': (), 'GET': ()}**

**bounded_layer = None**

**log = <logging.Logger object>**

**do_GET**()
Get information about an entity. R for CRUD convention.

**do_POST**()
Create an entity. C for CRUD convention.

**do_PUT**()
Update an entity. U for CRUD convention.

**do_DELETE**()
Delete an entity. D for CRUD convention.

**do_OPTIONS**()
Handling unsupported HTTP verbs

**Returns** None

**do_HEAD**()
Handling unsupported HTTP verbs

**Returns** None

**do_TRACE**()
Handling unsupported HTTP verbs

**Returns** None

**do_CONNECT**()
Handling unsupported HTTP verbs

**Returns** None

**_process_url**()
Split HTTP path and call the carved function if it is defined in this class and in request_perm

**Returns** None

**_parse_json_body**()
Parse HTTP request body in JSON format

---

**Note:** Call only once by HTTP request

---

---

**Note:** Parsed JSON object is Unicode

---

GET, DELETE messages don't have body - return empty dict by default

**Returns** request body in JSON format

**Return type** dict

**send_REST_headers**()
Set the allowed REST verbs as an HTTP header (Allow)

**Returns** None

**send_acknowledge**(*msg='{"result": "Accepted"}'*)
Send back acknowlede message

**Parameters**

- **msg** – response body
- **msg** – dict

**Returns** None

**_send_json_response**(*data*, *encoding='utf-8'*)
Send requested data in JSON format

**Parameters**

- **data** (*dict*) – data in JSON format
- **encoding** (*str*) – Set data encoding (optional)

**Returns** None

**error_content_type = 'text/json'**

**send_error**(*code*, *message=None*)
Override original function to send back allowed HTTP verbs and set format to JSON

**log_error**(*mformat*, **args*)
Overwritten to use POX logging mechanism

**log_message**(*mformat*, **args*)
Disable logging of incoming messages

**log_full_message**(*mformat*, **args*)
Overwritten to use POX logging mechanism

**_proceed_API_call**(*function*, **args*, ***kwargs*)
Fail-safe method to call API function

The cooperative micro-task context is handled by actual APIs

Should call this with params, not directly the function of actual API

**Parameters**

- **function** (*str*) – function name
- **args** (*tuple*) – Optional params
- **kwargs** (*dict*) – Optional named params

**Returns** None

***mapping.py* module** *AbstractMapper* is an abstract class for orchestration method which should implement mapping preparations and invoke actual mapping algorithm

*AbstractMappingStrategy* is an abstract class for containing entirely the mapping algorithm as a class method

**Module contents** Contains abstract classes for NFFG mapping

**class** escape.util.mapping.**AbstractMappingStrategy**
Bases: object

Abstract class for the mapping strategies

Follows the Strategy design pattern

**__init__**()
Init

**classmethod map**(*graph*, *resource*)
Abstract function for mapping algorithm

> **Warning:** Derived class have to override this function

**Parameters**

- **graph** (NFFG) – Input graph which need to be mapped
- **resource** (NFFG) – resource info

**Raise** NotImplementedError

**Returns** mapped graph

**Return type** *NFFG*

**class** escape.util.mapping.**AbstractMapper**(*layer_name*, *strategy=None*, *threaded=None*)
Bases: pox.lib.revent.revent.EventMixin

Abstract class for graph mapping function

Inherited from :class‘EventMixin‘ to implement internal event-based communication

Contain common functions and initialization

**_defaults = {‘orchestration’: ‘ESCAPEMappingStrategy’, ‘service’: ‘DefaultServiceMappingStrategy’}**

**__init__**(*layer_name*, *strategy=None*, *threaded=None*)
Initialize Mapper class

Set given strategy class and threaded value or check in *CONFIG*

If no valid value is found for arguments set the default params defined in *_default*

> **Warning:** Strategy classes must be a subclass of AbstractMappingStrategy

> **Note:** SAS strategy is searched in *escape.service.sas_mapping*

> **Note:** ROS strategy is searched in *escape.orchest.ros_mapping*

**Parameters**

- **layer_name** (*str*) – name of the layer which initialize this class. This value is used to search the layer configuration in *CONFIG*
- **strategy** (AbstractMappingStrategy) – strategy class (optional)
- **threaded** (*bool*) – run mapping algorithm in separate Python thread instead of in the coop microtask environment (optional)

**Returns** None

**orchestrate**(*input_graph*, *resource_view*)
Abstract function for wrapping optional steps connected to orchestration

Implemented function call the mapping algorithm

> **Warning:** Derived class have to override this function

**Parameters**

- **input_graph** (NFFG) – graph representation which need to be mapped
- **resource_view** (AbstractVirtualizer) – resource information

**Raise** NotImplementedError

**Returns** mapped graph

**Return type** *NFFG*

**_start_mapping**(*graph*, *resource*)
Run mapping algorithm in a separate Python thread

**Parameters**

- **graph** (NFFG) – Network Function Forwarding Graph

- **resource** (NFFG) – global resource

**Returns** None

**_mapping_finished**(*nffg*)

Called from a separate thread when the mapping process is finished

**Warning:** Derived class have to override this function

**Parameters** **nffg** (NFFG) – generated NF-FG

**Returns** None

***misc.py* module** *schedule_as_coop_task()* helps invoking a function in POX's cooperative microtask environment

*call_as_coop_task()* hides POC core functionality and schedule a function in the coop microtask environment directly

*SimpleStandaloneHelper* is a helper class for mimic a minimal layer API as a dependency for other layer APIs to handles events

*enum()* is a helper function to generate Pythonic enumeration

*ESCAPEConfig* is a wrapper class for config

**Module contents** Contains miscellaneous helper functions

escape.util.misc.**schedule_as_coop_task**(*func*)

Decorator functions for running functions in an asynchronous way as a microtask in recoco's cooperative multitasking context (in which POX was written)

**Parameters** **func** (*func*) – decorated function

**Returns** decorator function

**Return type** func

escape.util.misc.**call_as_coop_task**(*func*, **args*, ***kwargs*)

Schedule a coop microtask and run the given function with parameters in it

Use POX core logic directly

**Parameters**

- **func** (*func*) – function need to run
- **args** (*tuple*) – nameless arguments
- **kwargs** (*dict*) – named arguments

**Returns** None

escape.util.misc.**enum**(**sequential*, ***named*)

Helper function to define enumeration. E.g.:

```
>>> Numbers = enum(ONE=1, TWO=2, THREE='three')
>>> Numbers = enum('ZERO', 'ONE', 'TWO')
>>> Numbers.ONE
1
>>> Numbers.reversed[2]
'TWO'
```

**Parameters**

- **sequential** (*list*) – support automatic enumeration
- **named** (*dict*) – support definition with unique keys

**Returns** Enum object

**Return type** dict

escape.util.misc.**quit_with_error**(*msg=None*, *logger='core'*)
Helper function for quitting in case of an error

**Parameters**

- **msg** (*str*) – error message (optional)
- **logger** (*str*) – logger name (default: core)

**Returns** None

**class** escape.util.misc.**SimpleStandaloneHelper**(*container*, *cover_name*)
Bases: object

Helper class for layer APIs to catch events and handle these in separate handler functions

**__init__**(*container*, *cover_name*)
Init

**Parameters**

- **container** – Container class reference
- **cover_name** (*str*) – Container's name for logging

**Type** EventMixin

**_register_listeners**()
Register event listeners

If a listener is explicitly defined in the class use this function otherwise use the common logger function

**Returns** None

**_log_event**(*event*)
Log given event

**Parameters** **event** (*Event*) – Event object which need to be logged

**Returns** None

**class** escape.util.misc.**Singleton**
Bases: type

Metaclass for classes need to be created only once.

Realize Singleton design pattern in a pythonic way.

**_instances = {<class 'escape.util.misc.ESCAPEConfig'>: <escape.util.misc.ESCAPEConfig object at 0x7f807e775d50>**

**__call__**(**args*, ***kwargs*)

**class** escape.util.misc.**ESCAPEConfig**(*default=None*)
Bases: object

Wrapper class for configuration to hide specialies with respect to storing, loading, parging and getting special data.

Contains functions for config handling and manipulation.

Should be instantiated once!

**__metaclass__**
alias of *Singleton*

**LAYERS = ('service', 'orchestration', 'adaptation', 'infrastructure')**

**__init__**(*default=None*)
Init configuration from given data or an empty dict

**Parameters** **default** (*dict*) – default configuration

**add_cfg**(*cfg*)
Override configuration

**Parameters** **cfg** (*dict*) – new configuration

**Returns** None

**load_config**(*config='escape.config'*)
Load static configuration from file if it exist or leave the default intact.

---

**Note:** The CONFIG is updated per data under the Layer entries. This means that the minimal amount of data have to given is the hole sequence or collection unter the appropriate key. E.g. the hole data under the 'STRATEGY' key in 'orchestration' layer.

---

**Parameters** **config** (*str*) – config file name relative to pox.py (optional)

**Returns** self

**Return type** *ESCAPEConfig*

**dump**()
Return with the entire configuration in JSON.

**Returns** config

**Return type** str

**is_loaded**(*layer*)
Return the value given UNIFY's layer is loaded or not.

**Parameters** **layer** (*str*) – layer name

**Returns** layer condition

**Return type** bool

**set_loaded**(*layer*)
Set the given layer LOADED value.

**Parameters** **layer** (*str*) – layer name

**Returns** None

**__getitem__**(*item*)
Can be used the config as a dictionary: CONFIG[...]

**Parameters** **item** (*str*) – layer key

**Returns** layer config

**Return type** dict

**__setitem__**(*key*, *value*)
Disable explicit layer config modification.

**__delitem__**(*key*)
Disable explicit layer config deletion.

**get_strategy**(*layer*)
Return with the Strategy class of the given layer.

**Parameters** **layer** (*str*) – layer name

**Returns** Strategy class

**Return type** *AbstractMappingStrategy*

**get_threaded**(*layer*)
Return with the value if the mapping strategy is needed to run in separated thread or not. If value is not defined: return False.

**Parameters** **layer** (*str*) – layer name

**Returns** threading value

**Return type** bool

**get_domain_component**(*component*)
Return with the class of the adaptation component.

**Parameters** **component** (*str*) – component name

**Returns** component class

**get_default_mgrs**()
Return the default DomainManagers for initialization on start.

**Returns** list of *AbstractDomainManager*

**Return type** list

**get_fallback_topology**(*topo_name*)
Return the fallback topology class.

**Parameters** **topo_name** (*str*) – name of the topo in CONFIG

**Returns** fallback topo class

**Return type** :any::*AbstractTopology*

**get_clean_after_shutdown**()
Return with the value if a cleaning process need to be done or not.

**Returns** cleanup (default: False)

**Return type** bool

***netconf.py* module** Requirements:

```
sudo apt-get install python-setuptools python-paramiko python-lxml \
python-libxml2 python-libxslt1 libxml2 libxslt1-dev

sudo pip install ncclient
```

*AbstractNETCONFAdapter* contains the main function for communication over NETCONF such as managing SSH channel, handling configuration, assemble RPC request and parse RPC reply

**Module contents** Implement the supporting classes for communication over NETCONF

**class** escape.util.netconf.**AbstractNETCONFAdapter**(*server*, *port*, *username*, *password*, *timeout=30*, *debug=False*)

Bases: object

Abstract class for various Adapters rely on NETCONF protocol (**RFC 4741**)

Contains basic functions for managing connection and invoking RPC calls. Configuration management can be handled by the external ncclient.manager.Manager class exposed by the manager property

Follows the Adapter design pattern

**NETCONF_NAMESPACE = 'urn:ietf:params:xml:ns:netconf:base:1.0'**

**RPC_NAMESPACE = None**

**__init__**(*server*, *port*, *username*, *password*, *timeout=30*, *debug=False*)

Initialize connection parameters

**Parameters**

- **server** (*str*) – server address
- **port** (*int*) – port number
- **username** (*str*) – username
- **password** (*str*) – password
- **timeout** (*int*) – connection timeout (default=30)
- **debug** (*bool*) – print DEBUG infos, RPC messages ect. (default: False)

**Returns** None

**connected**

**Returns** Return connection state

**Return type** bool

**connection_data**

**Returns** Return connection data in (server, port, username) tuples

**Return type** tuple

**manager**

**Returns** Return the connection manager (wrapper for NETCONF commands)

**Return type** ncclient.manager.Manager

**connect**()

This function will connect to the netconf server.

**Returns** Also returns the NETCONF connection manager

**Return type** ncclient.manager.Manager

**disconnect**()

This function will close the connection.

**Returns** None

**get_config**(*source='running'*, *to_file=False*)

This function will download the configuration of the NETCONF agent in an XML format. If source is

None then the running config will be downloaded. Other configurations are netconf specific (**RFC 6241**) - running, candidate, startup

**Parameters**

- **source** (*str*) – NETCONF specific configuration source (defalut: running)
- **to_file** (*bool*) – save config to file

**Returns** None

**get**(*expr='/'*)

This process works as yangcli's GET function. A lot of information can be got from the running NETCONF agent. If an xpath-based expression is also set, the results can be filtered. The results are not printed out in a file, it's only printed to stdout

**Parameters** **expr** (*str*) – xpath-based expression

**Returns** result in XML

**Return type** str

**_create_rpc_request**(*rpc_name*, ***params*)

This function is devoted to create a raw RPC request message in XML format. Any further additional rpc-input can be passed towards, if netconf agent has this input list, called 'options'. Switches is used for connectVNF rpc in order to set the switches where the vnf should be connected.

**Parameters**

- **rpc_name** (*str*) – rpc name
- **options** (*dict*) – additional RPC input in the specific <options> tag
- **switches** (*list*) – set the switches where the vnf should be connected
- **params** (*dict*) – input params for the RPC using param's name as XML tag name

**Returns** raw RPC message in XML format (lxml library)

**Return type** `lxml.etree.ElementTree`

**_parse_rpc_response**(*data=None*)

Parse raw XML response and return params in dictionary. If data is given it is parsed instead of the lasr response and the result will not be saved.

**Parameters** **data** (`lxml.etree.ElementTree`) – raw data (uses last reply by default)

**Returns** return parsed params

**Return type** dict

**_invoke_rpc**(*request_data*)

This function is devoted to call an RPC, and parses the rpc-reply message (if needed) and returns every important parts of it in as a dictionary. Any further additional rpc-input can be passed towards, if netconf agent has this input list, called 'options'. Switches is used for connectVNF rpc in order to set the switches where the vnf should be connected.

**Parameters** **request_data** (*dict*) – data for RPC request body

**Returns** raw RPC response

**Return type** `lxml.etree.ElementTree`

**call_RPC**(*rpc_name*, *no_rpc_error=False*, ***params*)

Call an RPC given by rpc_name. If *no_rpc_error* is set returns with a dict instead of raising `RPCError`

**Parameters**

- **rpc_name** (*str*) – RPC name
- **no_rpc_error** (*bool*) – return with dict (RPC error) instead of exception

**Returns** RPC reply

**Return type** dict

**__enter__**()
Context manager setup action.

Usage:

```
with AbstractNETCONFAdapter() as adapter:
  ...
```

**__exit__**(*exc_type*, *exc_val*, *exc_tb*)
Context manager cleanup action

**_AbstractNETCONFAdapter__parse_rpc_params**(*rpc_request*, *params*)
Parse given keyword arguments and generate RPC body in proper XML format. The key value is used as the XML tag name. If the value is another dictionary the XML structure follows the hierarchy. The param values can be only simple types and dictionary for simplicity. Convertation example:

```
{
  'vnf_type': 'headerDecompressor',
  'options': {
              'name': 'ip',
              'value': 127.0.0.1
              }
}
```

will be generated into:

```
<rpc-call-name>
  <vnf_type>headerDecompressor</vnf_type>
  <options>
    <name>ip</name>
    <value>127.0.0.1</value>
  </options>
</rpc-call-name>
```

**Parameters**

- **rpc_request** (`lxml.etree.ElementTree`) – empty RPC request
- **params** (*dict*) – RPC call argument given in a dictionary

**Returns** parsed params in XML format (lxml library)

**Return type** `lxml.etree.ElementTree`

**_AbstractNETCONFAdapter__parse_xml_response**(*element*, *namespace=None*)
This is an inner function, which is devoted to automatically analyze the rpc-reply message and iterate through all the xml elements until the last child is found, and then create a dictionary. Return a dict with the parsed data. If the reply is OK the returned dict contains an *rcp-reply* element with value *OK*.

**Parameters**

- **element** (`lxml.etree.ElementTree`) – XML element
- **namespace** – namespace

**Type** str

**Returns** parsed XML data

**Return type** dict

**_AbstractNETCONFAdapter__remove_namespace**(*xml_element*, *namespace=None*)
Own function to remove the ncclient's namespace prefix, because it causes "definition not found error" if OWN modules and RPCs are being used

**Parameters**

- **xml_element** (lxml.etree.ElementTree) – XML element
- **namespace** (lxml.etree.ElementTree) – namespace

**Returns** cleaned XML elemenet

**Return type** lxml.etree.ElementTree

***nffg.py* module** *NFFG* represents the internal format of NF-FG, SG and RG

**Module contents** Abstract class and implementation for basic operations with a single NF-FG, such as building, parsing, processing NF-FG, helper functions, etc.

**class** escape.util.nffg.**AbstractNFFG**(*id*, *name=None*, *version='1.0'*, *json=None*, *file=None*)
Bases: object

Abstract class for managing single NF-FG data structure

The NF-FG data model is described in YANG. This class provides the interfaces with the high level data manipulation functions.

**__metaclass__**
alias of ABCMeta

**__init__**(*id*, *name=None*, *version='1.0'*, *json=None*, *file=None*)
Init

**add_nf**(*node_nf*)
Add a single NF node to the NF-FG

**add_sap**(*node_sap*)
Add a single SAP node to the NF-FG

**add_infra**(*node_infra*)
Add a single infrastructure node to the NF-FG

**add_edge**(*src*, *dst*, *params=None*)
Add an edge to the NF-FG

**Parameters**

- **src** (*(Node, Port) inherited Node classes: NodeNF, NodeSAP, NodeInfra*) – source (node, port) of the edge
- **dst** (*(Node, Port) inherited Node classes: NodeNF, NodeSAP, NodeInfra*) – destination (node, port) of the edge
- **params** (*ResOfEdge or Flowrule*) – attribute of the edge depending on the type

**Returns** None

**add_link**(*edge_link*)
Add a static or dynamic infrastructure link to the NF-FG

**add_sglink**(*edge_sglink*)
Add an SG link to the NF-FG

**add_req**(*edge_req*)
Add a requirement link to the NF-FG

**del_node**(*id*)
Delete a single node from the NF-FG

**_init_from_json**(*json_data*)
Initialize the NFFG object from JSON data

**Parameters** **json_data** (*str*) – NF-FG represented in JSON format

**Returns** None

**static load_from_file**(*filename*)

**to_json**()
Return a JSON string represent this instance

**Returns** JSON formatted string

**Return type** str

**__copy__**()
Magic class for creating a shallow copy of actual class using the copy.copy() function of Python standard library. This means that, while the instance itself is a new instance, all of its data is referenced.

**Returns** shallow copy of this instace

**Return type** *NFFG*

**__deepcopy__**(*memo={}*)
Magic class for creating a deep copy of actual class using the copy.deepcopy() function of Python standard library. The object and its data are both copied.

**Parameters** **memo** (*dict*) – is a cache of previously copied objects

**Returns** shallow copy of this instace

**Return type** *NFFG*

**__abstractmethods__ = frozenset(['add_sglink', 'add_req', 'add_link', 'del_node', 'add_infra', 'add_sap', 'add_nf']**

**_abc_cache = <_weakrefset.WeakSet object>**

**_abc_negative_cache = <_weakrefset.WeakSet object>**

**_abc_negative_cache_version = 25**

**_abc_registry = <_weakrefset.WeakSet object>**

**class** escape.util.nffg.**NFFG**(*id=None*, *name=None*, *version='1.0'*, *json=None*, *file=None*)
Bases: *escape.util.nffg.AbstractNFFG*, networkx.classes.multigraph.MultiGraph

NF-FG implementation based on NetworkX.

Implement the AbstractNFFG using NetworkX graph representation internally. Implement the high level functions and additionally expose NetworkX API.

**__init__**(*id=None*, *name=None*, *version='1.0'*, *json=None*, *file=None*)
Init

**add_nf**(*node_nf*)
Add a single NF node to the NF-FG

**add_sap**(*node_sap*)
Add a single SAP node to the NF-FG

**add_infra**(*node_infra*)
Add a single infrastructure node to the NF-FG

**add_link**(*edge_link*)
Add a static or dynamic infrastructure link to the NF-FG

**add_sglink**(*edge_sglink*)
Add an SG link to the NF-FG

**add_req**(*edge_req*)
Add a requirement link to the NF-FG

**del_node**(*id*)
Delete a single node from the NF-FG

**__abstractmethods__ = frozenset([])**

**_abc_cache = <_weakrefset.WeakSet object>**

**_abc_negative_cache = <_weakrefset.WeakSet object>**

**_abc_negative_cache_version = 25**

**_abc_registry = <_weakrefset.WeakSet object>**

escape.util.nffg.**main**(*argv=None*)

***pox_extension.py* module** *OpenFlowBridge* is a special version of OpenFlow event originator class

*ExtendedOFConnectionArbiter* dispatches incoming OpenFlow connections to fit ESCAPEv2

**Module contents** Override and extend internal POX components to achieve ESCAPE-desired behaviour

**class** escape.util.pox_extension.**OpenFlowBridge**
Bases: pox.openflow.OpenFlowNexus

Own class for listening OpenFlow event originated by one of the contained Connection and sending OpenFlow messages according to DPID

Purpose of the class mostly fits the Bride design pattern

**class** escape.util.pox_extension.**ExtendedOFConnectionArbiter**(*default=False*)
Bases: pox.openflow.OpenFlowConnectionArbiter

Extended connection arbiter class for dispatching incoming OpenFlow Connection between registered OF event originators ( OpenFlowNexus) according to the connection's listening address

**_core_name = 'OpenFlowConnectionArbiter'**

**__init__**(*default=False*)
Init

**Parameters** **default** (OpenFlowNexus) – inherited param

**add_connection_listener**(*address*, *nexus*)
Helper function to register connection listeners a.k.a. OpenFlowNexus

**Parameters**

- **address** (*tuple*) – listened socket name in form of (address, port)
- **nexus** (*OpenFlowBridge*) – registered object

**Returns** registered listener

**Return type** *OpenFlowBridge*

**classmethod activate**()
Register this component into `pox.core` and replace already registered Arbiter.

**Returns** registered component

**Return type** *ExtendedOFConnectionArbiter*

**getNexus**(*connection*)
Return registered connection listener or default `core.openflow`.

Fires ConnectionIn event.

**Parameters** **connection** (`Connection`) – incoming connection object

**Returns** OpenFlow event originator object

**Return type** `OpenFlowNexus`

## 2.3 Main modules for layers/sublayers

### 2.3.1 The *unify.py* top module

Basic POX module for ESCAPE

Initiate appropriate APIs

Follows POX module conventions

`unify.`**_start_components**(*event*)
Initiate and run POX with ESCAPE components

**Parameters** **event** (*GoingUpEvent*) – POX's going up event

**Returns** None

`unify.`**add_dependencies**()
Add dependency directories to PYTHONPATH. Dependencies are directories besides the escape.py initial script except pox.

**Returns** None

`unify.`**launch**(*sg_file=''*, *gui=False*, *full=False*, *debug=True*)
Launch function called by POX core when core is up

**Parameters**

- **sg_file** (*str*) – Path of the input Service graph (optional)
- **gui** (*bool*) – Signal for initiate GUI (optional)
- **full** (*bool*) – Initiate Infrastructure Layer also

**Returns** None

### Submodules

#### The *service.py* main module

Basic POX module for ESCAPE Service (Graph Adaptation) sublayer

Initiate appropriate API class which implements U-Sl reference point

Follows POX module conventions

service.**_start_layer**(*event*)
Initiate and run Service module

**Parameters** **event** (*GoingUpEvent*) – POX's going up event

**Returns** None

service.**launch**(*sg_file=''*, *gui=False*, *standalone=False*)
Launch function called by POX core when core is up

**Parameters**

- **sg_file** (*str*) – Path of the input Service graph (optional)
- **gui** (*bool*) – Initiate built-in GUI (optional)
- **standalone** (*bool*) – Run layer without dependency checking (optional)

**Returns** None

#### The *orchestration.py* main module

Basic POX module for ESCAPE Resource Orchestration Sublayer (ROS)

Initiate appropriate API class which implements Sl-Or reference point

Follows POX module conventions

orchestration.**_start_layer**(*event*)
Initiate and run Orchestration module

**Parameters** **event** (*GoingUpEvent*) – POX's going up event

**Returns** None

orchestration.**launch**(*nffg_file=''*, *standalone=False*)
Launch function called by POX core when core is up

**Parameters**

- **nffg_file** (*str*) – Path of the NF-FG graph (optional)
- **standalone** (*bool*) – Run layer without dependency checking (optional)

**Returns** None

#### The *adaptation.py* main module

Basic POX module for ESCAPE Controller Adaptation Sublayer (CAS)

Initiate appropriate API class which implements Or-Ca reference point

Follows POX module conventions

adaptation.**_start_layer**(*event*)
Initiate and run Adaptation module

**Parameters** **event** (*GoingUpEvent*) – POX's going up event

**Returns** None

adaptation.**launch**(*mapped_nffg_file=''*, *with_infr=False*, *standalone=False*)
Launch function called by POX core when core is up

**Parameters**

- **mapped_nffg_file** (*str*) – Path of the mapped NF-FG graph (optional)
- **with_infr** (*bool*) – Set Infrastructure as a dependency
- **standalone** (*bool*) – Run layer without dependency checking (optional)

**Returns** None

#### The *infrastructure.py* main module

Basic POX module for ESCAPE Infrastructure Layer

Initiate appropriate API class which emulate Co-Rm reference point

Follows POX module conventions

infrastructure.**_start_layer**(*event*)
Initiate and run Infrastructure module

**Parameters** **event** (*GoingUpEvent*) – POX's going up event

**Returns** None

infrastructure.**launch**(*standalone=False*)
Launch function called by POX core when core is up

**Parameters** **standalone** (*bool*) – Run layer without dependency checking (optional)

**Returns** None

## 2.4 README

ESCAPEv2 example commands

**The simpliest use-case:**

```
$ ./escape.py
```

Usage:

```
$ ./escape.py -h
usage: escape.py [-h] [-v] [-d] [-f] [-i]

ESCAPE: Extensible Service ChAin Prototyping Environment using Mininet, Click,
NETCONF and POX

optional arguments:
  -h, --help         show this help message and exit
  -v, --version      show program's version number and exit
```

```
ESCAPE arguments:
  -d, --debug        run the ESCAPE in debug mode
  -f, --full         run the infrastructure layer also
  -i, --interactive  run an interactive shell for observing internal states
```

**More advanced commands:**

Basic command:

```
$ ./pox.py unify
```

Basic command for debugging:

```
$ ./pox.py --verbose --no-openflow unify py
```

Basic command to initiate a built-in emulated network for testing:

```
# Infrastructure layer requires root privileges due to use of Mininet!
$ sudo ./pox.py unify --full
```

Minimal command with explicitly-defined components (components' order is irrelevant):

```
$ ./pox.py service orchestration adaptation
```

Without service layer:

```
$ ./pox.py orchestration adaptation
```

With infrastructure layer:

```
$ sudo ./pox.py service orchestration adaptation --with_infr infrastructure
```

Long version with debugging and explicitly-defined components (analogous with ./pox.py unify –full):

```
$./pox.py --verbose log.level --DEBUG samples.pretty_log service orchestration adaptation--with_infr
```

Start layers with graph-represented input contained in a specific file:

```
$ ./pox.py service --sg_file=<path> ...
$ ./pox.py unify --sg_file=<path>

$ ./pox.py orchestration --nffg_file=<path> ...
$ ./pox.py adaptation --mapped_nffg_file=<path> ...
```

Start ESCAPEv2 with built-in GUI:

```
$ ./pox.py service --gui ...
$ ./pox.py unify --gui
```

Start layer in standalone mode (no dependency handling) for test/debug:

```
$ ./pox.py service --standalone
$ ./pox.py orchestration --standalone
$ ./pox.py adaptation --standalone
$ sudo ./pox.py infrastructure --standalone

$ ./pox.py service orchestration --standalone
```

## 2.5 REST API

*Content Negotiation:* The Service layer's RESTful API accepts and returns data only in JSON format.

*Operations:* Every operation need to be called under the **escape/** path. E.g. *http://localhost/escape/version*

| Path | Params | HTTP verbs | Description |
|---|---|---|---|
| */version* | `None` | GET | Returns with the current version of ESCAPEv2 |
| */echo* | `ANY` | ALL | Returns with the given parameters |
| */operations* | `None` | GET | Returns with the implemented operations as a list |
| */sg* | `NFFG` | POST | Initiate given NFFG. Returns the given NFFG initiation is accepted or not. |

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

## a



## e



## i



## o



## s



## u

## Symbols

## A



## B



## C

## D



## E

## F



## G



## I

## L



## M



## N

## O



## P



## Q



## R



## S

## T



## U



## V