PrimAITE

Release 2.0.0

Defence Science and Technology Laboratory UK

Aug 15, 2023

CONTENTS:

CHAPTER

ONE

WHAT IS PRIMAITE?

PrimAITE (Primary-level AI Training Environment) is a simulation environment for training AI under the ARCD programme. It incorporates the functionality required of a Primary-level environment, as specified in the Dstl ARCD Training Environment Matrix document:

- The ability to model a relevant platform / system context;
- The ability to model key characteristics of a platform / system by representing connections, IP addresses, ports, traffic loading, operating systems, file system, services and processes;
- Operates at machine-speed to enable fast training cycles.

CHAPTER

TWO

WHAT IS PRIMAITE BUILT WITH

- OpenAI's Gym is used as the basis for AI blue agent interaction with the PrimAITE environment
- Networkx is used as the underlying data structure used for the PrimAITE environment
- Stable Baselines 3 is used as a default source of RL algorithms (although PrimAITE is not limited to SB3 agents)
- Ray RLlib is used as an additional source of RL algorithms
- Typer is used for building CLIs (Command Line Interface applications)
- Jupyterlab is used as an extensible environment for interactive and reproducible computing, based on the Jupyter Notebook Architecture
- Platformdirs is used for finding the right location to store user data and configuration but varies per platform
- Plotly is used for building high level charts

CHAPTER

THREE

GETTING STARTED WITH PRIMAITE

Head over to the *Getting Started* page to install and setup PrimAITE!

3.1 Getting Started

Getting Started with PrimAITE

Pre-Requisites

In order to get **PrimAITE** installed, you will need to have a python version between 3.8 and 3.10 installed. If you don't already have it, this is how to install it:

Listing 1: Unix

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.10
sudo apt-get install python3-pip
sudo apt-get install python3-venv
```

Listing 2: Windows (Powershell)

```
- Manual install from: https://www.python.org/downloads/release/python-31011/
```

PrimAITE is designed to be OS-agnostic, and thus should work on most variations/distros of Linux, Windows, and MacOS.

3.1.1 Install PrimAITE

1. Create a primaite directory in your home directory:

Listing 3: Unix

```
mkdir ~/primaite/2.0.0
```

Listing 4: Windows (Powershell)

```
mkdir ~\primaite\2.0.0
```

2. Navigate to the primaite directory and create a new python virtual environment (venv)

Listing 5: Unix

```
cd ~/primaite/2.0.0
python3 -m venv .venv
```

Listing 6: Windows (Powershell)

```
cd ~\primaite\2.0.0
python3 -m venv .venv
attrib +h .venv /s /d # Hides the .venv directory
```

3. Activate the venv

Listing 7: Unix

```
source .venv/bin/activate
```

Listing 8: Windows (Powershell)

```
.\.venv\Scripts\activate
```

4. Install PrimAITE using pip from PyPi

Listing 9: Unix

```
pip install primaite
```

Listing 10: Windows (Powershell)

```
pip install primaite
```

5. Perform the PrimAITE setup

Listing 11: Unix

```
primaite setup
```

Listing 12: Windows (Powershell)

```
primaite setup
```

3.1.2 Clone & Install PrimAITE for Development

To be able to extend PrimAITE further, or to build wheels manually before install, clone the repository to a location of your choice:

```
git clone <repo path>
cd primaite
```

Create and activate your Python virtual environment (venv)

Listing 13: Unix

```
python3 -m venv venv
source venv/bin/activate
```

Listing 14: Windows (Powershell)

```
python3 -m venv venv
.\venv\Scripts\activate
```

Install PrimAITE with the dev extra

Listing 15: Unix

```
pip install -e .[dev]
```

Listing 16: Windows (Powershell)

```
pip install -e .[dev]
```

To view the complete list of packages installed during PrimAITE installation, go to the dependencies page (Dependencies).

3.2 About PrimAITE

3.2.1 Features

PrimAITE provides the following features:

- A flexible network / system laydown based on the Python networkx framework
- Nodes and links (edges) host Python classes in order to present attributes and methods (and hence, a more representative model of a platform / system)
- A 'green agent' Information Exchange Requirement (IER) function allows the representation of traffic (protocols and loading) on any / all links. Application of IERs is based on the status of node operating systems and services
- A 'green agent' node Pattern-of-Life (PoL) function allows the representation of core behaviours on nodes (e.g. changing the Hardware state, Software State, Service state, or File System state)
- An Access Control List (ACL) function, mimicking the behaviour of a network firewall, is applied across the model, following standard ACL rule format (e.g. DENY/ALLOW, source IP, destination IP, protocol and port). Application of IERs adheres to any ACL restrictions
- Presents an OpenAI Gym interface to the environment, allowing integration with any OpenAI Gym compliant defensive agents
- Red agent activity based on 'red' IERs and 'red' PoL
- Defined reward function for use with RL agents (based on nodes status, and green / red IER success)
- Fully configurable (network / system laydown, IERs, node PoL, ACL, episode step period, episode max steps) and repeatable to suit the training requirements of agents. Therefore, not bound to a representation of any particular platform, system or technology
- Full capture of discrete metrics relating to agent training (full system state, agent actions taken, average reward)

- Networkx provides laydown visualisation capability

3.2.2 Architecture - Nodes and Links

Nodes

An inheritance model has been adopted in order to model nodes. All nodes have the following base attributes (Class: Node):

- ID
- Name
- Type (e.g. computer, switch, RTU - enumeration)
- Priority (P1, P2, P3, P4 or P5 - enumeration)
- Hardware State (ON, OFF, RESETTING, SHUTTING_DOWN, BOOTING - enumeration)

Active Nodes also have the following attributes (Class: Active Node):

- IP Address
- Software State (GOOD, PATCHING, COMPROMISED - enumeration)
- File System State (GOOD, CORRUPT, DESTROYED, REPAIRING, RESTORING - enumeration)

Service Nodes also have the following attributes (Class: Service Node):

- List of Services (where service is composed of service name and port). There is no theoretical limit on the number of services that can be modelled. Services and protocols are currently intrinsically linked (i.e. a service is an application on a node transmitting traffic of this protocol type)
- Service state (GOOD, PATCHING, COMPROMISED, OVERWHELMED - enumeration)

Passive Nodes are currently not used (but may be employed for non IP-based components such as machinery actuators in future releases).

Links

Links are modelled both as network edges (networkx) and as Python classes, in order to extend their functionality. Links include the following attributes:

- ID
- Name
- Bandwidth (bits/s)
- Source node ID
- Destination node ID
- Protocol list (containing the loading of protocols currently running on the link)

When the simulation runs, IERs are applied to the links in order to model traffic loading, individually assigned to each protocol. This allows green (background) and red agent behaviour to be modelled, and defensive agents to identify suspicious traffic patterns at a protocol / traffic loading level of fidelity.

3.2.3 Information Exchange Requirements (IERs)

PrimAITE adopts the concept of Information Exchange Requirements (IERs) to model both green agent (background) and red agent (adversary) behaviour. IERs are used to initiate modelling of traffic loading on the network, and have the following attributes:

- ID
- Start step (i.e. which step in the training episode should the IER start)
- End step (i.e. which step in the training episode should the IER end)
- Source node ID
- Destination node ID
- Load (bits/s)
- Protocol
- Port
- Running status (i.e. on / off)

The application of green agent IERs between a source and destination follows a number of rules. Specifically:

1. Does the current simulation time step fall between IER start and end step
2. Is the source node operational (both physically and at an O/S level), and is the service (protocol / port) associated with the IER (a) present on this node, and (b) in an operational state (i.e. not PATCHING)
3. Is the destination node operational (both physically and at an O/S level), and is the service (protocol / port) associated with the IER (a) present on this node, and (b) in an operational state (i.e. not PATCHING)
4. Are there any Access Control List rules in place that prevent the application of this IER
5. Are all switches in the (OSPF) path between source and destination operational (both physically and at an O/S level)

For red agent IERs, the application of IERs between a source and destination follows a number of subtly different rules. Specifically:

1. Does the current simulation time step fall between IER start and end step
2. Is the source node operational, and is the service (protocol / port) associated with the IER (a) present on that node and (b) already in a compromised state
3. Is the destination node operational, and is the service (protocol / port) associated with the IER present on that node
4. Are there any Access Control List rules in place that prevent the application of this IER
5. Are all switches in the (OSPF) path between source and destination operational (both physically and at an O/S level)

Assuming the rules pass, the IER is applied to all relevant links (based on use of OSPF) between source and destination.

3.2.4 Node Pattern-of-Life

Every node can be impacted (i.e. have a status change applied to it) by either green agent pattern-of-life or red agent pattern-of-life. This is distinct from IERs, and allows for attacks (and defence) to be modelled purely within the confines of a node.

The status changes that can be made to a node are as follows:

- All Nodes:
  - Hardware State:
    - ON
    - OFF
    - RESETTING - when a status of resetting is entered, the node will automatically exit this state after a number of steps (as defined by the nodeResetDuration configuration item) after which it returns to an ON state
    - BOOTING
    - SHUTTING_DOWN
- Active Nodes and Service Nodes:
  - Software State:
    - GOOD
    - PATCHING - when a status of patching is entered, the node will automatically exit this state after a number of steps (as defined by the osPatchingDuration configuration item) after which it returns to a GOOD state
    - COMPROMISED
  - File System State:
    - GOOD
    - CORRUPT (can be resolved by repair or restore)
    - DESTROYED (can be resolved by restore only)
    - REPAIRING - when a status of repairing is entered, the node will automatically exit this state after a number of steps (as defined by the fileSystemRepairingLimit configuration item) after which it returns to a GOOD state
    - RESTORING - when a status of repairing is entered, the node will automatically exit this state after a number of steps (as defined by the fileSystemRestoringLimit configuration item) after which it returns to a GOOD state
- Service Nodes only:
  - Service State (for any associated service):
    - GOOD
    - PATCHING - when a status of patching is entered, the service will automatically exit this state after a number of steps (as defined by the servicePatchingDuration configuration item) after which it returns to a GOOD state
    - COMPROMISED
    - OVERWHELMED

Red agent pattern-of-life has an additional feature not found in the green pattern-of-life. This is the ability to influence the state of the attributes of a node via a number of different conditions:

- DIRECT:

The pattern-of-life described by the configuration file item will be applied regardless of any other conditions in the network. This is particularly useful for direct red agent entry into the network.

- IER:

The pattern-of-life described by the configuration file item will be applied to the service on the node, only if there is an IER of the same protocol / service type incoming at the specified timestep.

- SERVICE:

The pattern-of-life described by the configuration file item will be applied to the node based on the state of a service. The service can either be on the same node, or a different node within the network.

3.2.5 Access Control List modelling

An Access Control List (ACL) is modelled to provide the means to manage traffic flows in the system. This will allow defensive agents the means to turn on / off rules, or potentially create new rules, to counter an attack.

The ACL follows a standard network firewall format. For example:

Table 1: ACL example

Permission	Source IP	Dest IP	Protocol	Port
DENY	192.168.1.2	192.168.1.3	HTTPS	443
ALLOW	192.168.1.4	ANY	SMTP	25
DENY	ANY	192.168.1.5	ANY	ANY

All ACL rules are considered when applying an IER. Logic follows the order of rules, so a DENY or ALLOW for the same parameters will override an earlier entry.

3.2.6 Observation Spaces

The observation space provides the blue agent with information about the current status of nodes and links.

PrimAITE builds on top of Gym Spaces to create an observation space that is easily configurable for users. It's made up of components which are managed by the *primaite.environment.observations.ObservationsHandler*. Each training scenario can define its own observation space, and the user can choose which information to inlude, and how it should be formatted.

3.2.6.1 NodeLinkTable component

For example, the *primaite.environment.observations.NodeLinkTable* component represents the status of nodes and links as a `gym.spaces.Box` with an example format shown below:

An example observation space is provided below:

Table 2: Observation Space example

	ID	Hardware State	Software State	File System State	Service / Protocol A	Service / Protocol B
Node A	1	1	1	1	1	1
Node B	2	1	3	1	1	1
Node C	3	2	1	1	3	2
Link 1	5	0	0	0	0	10000
Link 2	6	0	0	0	0	10000
Link 3	7	0	0	0	5000	0

For the nodes, the following values are represented:

```
[
  ID
  Hardware State            (1=ON,   2=OFF,  3=RESETTING,  4=SHUTTING_DOWN, 5=BOOTING)
  Operating System State    (0=none, 1=GOOD, 2=PATCHING,   3=COMPROMISED)
  File System State         (0=none, 1=GOOD, 2=CORRUPT,    3=DESTROYED,  4=REPAIRING,␣
↪5=RESTORING)
  Service1/Protocol1 state  (0=none, 1=GOOD, 2=PATCHING,   3=COMPROMISED)
  Service2/Protocol2 state  (0=none, 1=GOOD, 2=PATCHING,   3=COMPROMISED)
]
```

(Note that each service available in the network is provided as a column, although not all nodes may utilise all services)

For the links, the following statuses are represented:

```
[
  ID
  Hardware State            (0=not applicable)
  Operating System State    (0=not applicable)
  File System State         (0=not applicable)
  Service1/Protocol1 state  (Traffic load from this protocol on this link)
  Service2/Protocol2 state  (Traffic load from this protocol on this link)
]
```

3.2.6.2 NodeStatus component

This is a MultiDiscrete observation space that can be though of as a one-dimensional vector of discrete states. The example above would have the following structure:

```
[
  node1_info
  node2_info
  node3_info
]
```

Each `node_info` contains the following:

```
[
  hardware_state    (0=none, 1=ON,   2=OFF,      3=RESETTING, 4=SHUTTING_DOWN, 5=BOOTING)
  software_state    (0=none, 1=GOOD, 2=PATCHING, 3=COMPROMISED)
```

(continues on next page)

(continued from previous page)

```
  file_system_state (0=none, 1=GOOD, 2=CORRUPT,  3=DESTROYED, 4=REPAIRING, 5=RESTORING)
  service1_state    (0=none, 1=GOOD, 2=PATCHING, 3=COMPROMISED)
  service2_state    (0=none, 1=GOOD, 2=PATCHING, 3=COMPROMISED)
]
```

In a network with three nodes and two services, the full observation space would have 15 elements. It can be written with `gym` notation to indicate the number of discrete options for each of the elements of the observation space. For example:

```
gym.spaces.MultiDiscrete([4,5,6,4,4,4,5,6,4,4,4,5,6,4,4])
```

Note: NodeStatus observation component provides information only about nodes. Links are not considered.

3.2.6.3 LinkTrafficLevels

This component is a MultiDiscrete space showing the traffic flow levels on the links in the network, after applying a threshold to convert it from a continuous to a discrete value. There are two configurable parameters: * `quantisation_levels` determines how many discrete bins to use for converting the continuous traffic value to discrete (default is 5). * `combine_service_traffic` determines whether to separately output traffic use for each network protocol or whether to combine them into an overall value for the link. (default is `True`)

For example, with default parameters and a network with three links, the structure of this component would be:

```
[
  link1_status
  link2_status
  link3_status
]
```

Each `link_status` is a number from 0-4 representing the network load in relation to bandwidth.

```
0 = No traffic (0%)
1 = low traffic (1%-33%)
2 = medium traffic (33%-66%)
3 = high traffic (66%-99%)
4 = max traffic/ overwhelmed (100%)
```

Using `gym` notation, the shape of the obs space is: `gym.spaces.MultiDiscrete([5,5,5])`.

3.2.7 Action Spaces

The action space available to the blue agent comes in two types:

1. Node-based
2. Access Control List
3. Any (Agent can take both node-based and ACL-based actions)

The choice of action space used during a training session is determined in the config_[name].yaml file.

Node-Based

The agent is able to influence the status of nodes by switching them off, resetting, or patching operating systems and services. In this instance, the action space is an OpenAI Gym spaces.Discrete type, as follows:

- Dictionary item {... ,1: [x1, x2, x3,x4] ...} The placeholders inside the list under the key '1' mean the following:
  - [0, num nodes] - Node ID (0 = nothing, node ID)
  - [0, 4] - What property it's acting on (0 = nothing, 1 = state, 2 = SoftwareState, 3 = service state, 4 = file system state)
  - [0, 3] - Action on property (0 = nothing, 1 = on / scan, 2 = off / repair, 3 = reset / patch / restore)
  - [0, num services] - Resolves to service ID (0 = nothing, resolves to service)

Access Control List

The blue agent is able to influence the configuration of the Access Control List rule set (which implements a system-wide firewall). In this instance, the action space is an OpenAI spaces.Discrete type, as follows:

- Dictionary item {... ,1: [x1, x2, x3, x4, x5, x6] ...}

The placeholders inside the list under the key '1' mean the following:

- [0, 2] - Action (0 = do nothing, 1 = create rule, 2 = delete rule)
- [0, 1] - Permission (0 = DENY, 1 = ALLOW)
- [0, num nodes] - Source IP (0 = any, then 1 -> x resolving to IP addresses)
- [0, num nodes] - Dest IP (0 = any, then 1 -> x resolving to IP addresses)
- [0, num services] - Protocol (0 = any, then 1 -> x resolving to protocol)
- [0, num ports] - Port (0 = any, then 1 -> x resolving to port)

ANY The agent is able to carry out both **Node-Based** and **Access Control List** operations.

This means the dictionary will contain key-value pairs in the format of BOTH Node-Based and Access Control List as seen above.

3.2.8 Rewards

A reward value is presented back to the blue agent on the conclusion of every step. The reward value is calculated via two methods which combine to give the total value:

1. Node and service status
2. IER status

Node and service status

On every step, the status of each node is compared against both a reference environment (simulating the situation if the red and blue agents had not impacted the environment) and the before and after state of the environment. If the comparison against the reference environment shows no difference, then the score provided is "AllOK". If there is a difference with respect to the reference environment, the before and after states are compared, and a score determined. See *The Config Files Explained* for details of reward values.

IER status

On every step, the full IER set is examined to determine whether green and red agent IERs are being permitted to run. Any red agent IERs running incur a penalty; any green agent IERs not permitted to run also incur a penalty. See *The Config Files Explained* for details of reward values.

3.2.9 Future Enhancements

The PrimAITE project has an ambition to include the following enhancements in future releases:

- Integration with a suitable standardised framework to allow multi-agent integration
- Integration with external threat emulation tools, either using off-line data, or integrating at runtime

3.3 The Config Files Explained

PrimAITE uses two configuration files for its operation:

- **The Training Config**

  Used to define the top-level settings of the PrimAITE environment, the reward values, and the session that is to be run.

- **The Lay Down Config**

  Used to define the low-level settings of a session, including the network laydown, green / red agent information exchange requirements (IERSs) and Access Control Rules.

3.3.1 Training Config:

The Training Config file consists of the following attributes:

Generic Config Values

- **agent_framework** [enum]

  This identifies the agent framework to be used to instantiate the agent algorithm. Select from one of the following:

  - NONE - Where a user developed agent is to be used
  - SB3 - Stable Baselines3
  - RLLIB - Ray RLlib.

- **agent_identifier**

  This identifies the agent to use for the session. Select from one of the following:

  - A2C - Advantage Actor Critic
  - PPO - Proximal Policy Optimization
  - HARDCODED - A custom built deterministic agent
  - RANDOM - A Stochastic random agent

- **random_red_agent** [bool]

  Determines if the session should be run with a random red agent

- **action_type** [enum]

  Determines whether a NODE, ACL, or ANY (combined NODE & ACL) action space format is adopted for the session

- **OBSERVATION_SPACE** [dict]

Allows for user to configure observation space by combining one or more observation components. List of available components is in *primaite.environment.observations*.

The observation space config item should have a `components` key which is a list of components. Each component config must have a `name` key, and can optionally have an `options` key. The `options` are passed to the component while it is being initialised.

This example illustrates the correct format for the observation space config item

```
observation_space:
components:
  - name: NODE_LINK_TABLE
  - name: NODE_STATUSES
  - name: LINK_TRAFFIC_LEVELS
  - name: ACCESS_CONTROL_LIST
    options:
      combine_service_traffic : False
      quantisation_levels: 99
```

Currently available components are:

- *NODE_LINK_TABLE* this does not accept any additional options
- *NODE_STATUSES*, this does not accept any additional options
- *ACCESS_CONTROL_LIST*, this does not accept additional options
- *LINK_TRAFFIC_LEVELS*, this accepts the following options:
  - `combine_service_traffic` - whether to consider bandwidth use separately for each network protocol or combine them into a single bandwidth reading (boolean)
  - `quantisation_levels` - how many discrete bandwidth usage levels to use for encoding. This can be an integer equal to or greater than 3.

The other configurable item is `flatten` which is false by default. When set to true, the observation space is flattened (turned into a 1-D vector). You should use this if your RL agent does not natively support observation space types like `gym.Spaces.Tuple`.

- **num_train_episodes** [int]

  This defines the number of episodes that the agent will train for.

- **num_train_steps** [int]

  Determines the number of steps to run in each episode of the training session.

- **num_eval_episodes** [int]

  This defines the number of episodes that the agent will be evaluated over.

- **num_eval_steps** [int]

  Determines the number of steps to run in each episode of the evaluation session.

- **time_delay** [int]

  The time delay (in milliseconds) to take between each step when running a GENERIC agent session

- **session_type** [text]

  Type of session to be run (TRAINING, EVALUATION, or BOTH)

- **load_agent** [bool]

  Determine whether to load an agent from file

- **agent_load_file** [text]

  File path and file name of agent if you're loading one in

- **observation_space_high_value** [int]

  The high value to use for values in the observation space. This is set to 1000000000 by default, and should not need changing in most cases

- **implicit_acl_rule** [str]

  Determines which Explicit rule the ACL list has - two options are: DENY or ALLOW.

- **max_number_acl_rules** [int]

  Sets a limit on how many ACL rules there can be in the ACL list throughout the training session.

Reward-Based Config Values

Rewards are calculated based on the difference between the current state and reference state (the 'should be' state) of the environment.

- **Generic [all_ok]** [float]

  The score to give when the current situation (for a given component) is no different from that expected in the baseline (i.e. as though no blue or red agent actions had been undertaken)

- **Node Hardware State [off_should_be_on]** [float]

  The score to give when the node should be on, but is off

- **Node Hardware State [off_should_be_resetting]** [float]

  The score to give when the node should be resetting, but is off

- **Node Hardware State [on_should_be_off]** [float]

  The score to give when the node should be off, but is on

- **Node Hardware State [on_should_be_resetting]** [float]

  The score to give when the node should be resetting, but is on

- **Node Hardware State [resetting_should_be_on]** [float]

  The score to give when the node should be on, but is resetting

- **Node Hardware State [resetting_should_be_off]** [float]

  The score to give when the node should be off, but is resetting

- **Node Hardware State [resetting]** [float]

  The score to give when the node is resetting

- **Node Operating System or Service State [good_should_be_patching]** [float]

  The score to give when the state should be patching, but is good

- **Node Operating System or Service State [good_should_be_compromised]** [float]

  The score to give when the state should be compromised, but is good

- **Node Operating System or Service State [good_should_be_overwhelmed]** [float]

  The score to give when the state should be overwhelmed, but is good

- **Node Operating System or Service State [patching_should_be_good]** [float]

  The score to give when the state should be good, but is patching

- **Node Operating System or Service State [patching_should_be_compromised]** [float]

  The score to give when the state should be compromised, but is patching
- **Node Operating System or Service State [patching_should_be_overwhelmed]** [float]

  The score to give when the state should be overwhelmed, but is patching
- **Node Operating System or Service State [patching]** [float]

  The score to give when the state is patching
- **Node Operating System or Service State [compromised_should_be_good]** [float]

  The score to give when the state should be good, but is compromised
- **Node Operating System or Service State [compromised_should_be_patching]** [float]

  The score to give when the state should be patching, but is compromised
- **Node Operating System or Service State [compromised_should_be_overwhelmed]** [float]

  The score to give when the state should be overwhelmed, but is compromised
- **Node Operating System or Service State [compromised]** [float]

  The score to give when the state is compromised
- **Node Operating System or Service State [overwhelmed_should_be_good]** [float]

  The score to give when the state should be good, but is overwhelmed
- **Node Operating System or Service State [overwhelmed_should_be_patching]** [float]

  The score to give when the state should be patching, but is overwhelmed
- **Node Operating System or Service State [overwhelmed_should_be_compromised]** [float]

  The score to give when the state should be compromised, but is overwhelmed
- **Node Operating System or Service State [overwhelmed]** [float]

  The score to give when the state is overwhelmed
- **Node File System State [good_should_be_repairing]** [float]

  The score to give when the state should be repairing, but is good
- **Node File System State [good_should_be_restoring]** [float]

  The score to give when the state should be restoring, but is good
- **Node File System State [good_should_be_corrupt]** [float]

  The score to give when the state should be corrupt, but is good
- **Node File System State [good_should_be_destroyed]** [float]

  The score to give when the state should be destroyed, but is good
- **Node File System State [repairing_should_be_good]** [float]

  The score to give when the state should be good, but is repairing
- **Node File System State [repairing_should_be_restoring]** [float]

  The score to give when the state should be restoring, but is repairing
- **Node File System State [repairing_should_be_corrupt]** [float]

  The score to give when the state should be corrupt, but is repairing

- **Node File System State [repairing_should_be_destroyed]** [float]

  The score to give when the state should be destroyed, but is repairing
- **Node File System State [repairing]** [float]

  The score to give when the state is repairing
- **Node File System State [restoring_should_be_good]** [float]

  The score to give when the state should be good, but is restoring
- **Node File System State [restoring_should_be_repairing]** [float]

  The score to give when the state should be repairing, but is restoring
- **Node File System State [restoring_should_be_corrupt]** [float]

  The score to give when the state should be corrupt, but is restoring
- **Node File System State [restoring_should_be_destroyed]** [float]

  The score to give when the state should be destroyed, but is restoring
- **Node File System State [restoring]** [float]

  The score to give when the state is restoring
- **Node File System State [corrupt_should_be_good]** [float]

  The score to give when the state should be good, but is corrupt
- **Node File System State [corrupt_should_be_repairing]** [float]

  The score to give when the state should be repairing, but is corrupt
- **Node File System State [corrupt_should_be_restoring]** [float]

  The score to give when the state should be restoring, but is corrupt
- **Node File System State [corrupt_should_be_destroyed]** [float]

  The score to give when the state should be destroyed, but is corrupt
- **Node File System State [corrupt]** [float]

  The score to give when the state is corrupt
- **Node File System State [destroyed_should_be_good]** [float]

  The score to give when the state should be good, but is destroyed
- **Node File System State [destroyed_should_be_repairing]** [float]

  The score to give when the state should be repairing, but is destroyed
- **Node File System State [destroyed_should_be_restoring]** [float]

  The score to give when the state should be restoring, but is destroyed
- **Node File System State [destroyed_should_be_corrupt]** [float]

  The score to give when the state should be corrupt, but is destroyed
- **Node File System State [destroyed]** [float]

  The score to give when the state is destroyed
- **Node File System State [scanning]** [float]

  The score to give when the state is scanning

- **IER Status [red_ier_running]** [float]

  The score to give when a red agent IER is permitted to run

- **IER Status [green_ier_blocked]** [float]

  The score to give when a green agent IER is prevented from running

Patching / Reset Durations

- **os_patching_duration** [int]

  The number of steps to take when patching an Operating System

- **node_reset_duration** [int]

  The number of steps to take when resetting a node's hardware state

- **service_patching_duration** [int]

  The number of steps to take when patching a service

- **file_system_repairing_limit** [int]:

  The number of steps to take when repairing the file system

- **file_system_restoring_limit** [int]

  The number of steps to take when restoring the file system

- **file_system_scanning_limit** [int]

  The number of steps to take when scanning the file system

- **deterministic** [bool]

  Set to true if the agent evaluation should be deterministic. Default is `False`

- **seed** [int]

  Seed used in the randomisation in agent training. Default is `None`

3.3.2 The Lay Down Config

The lay down config file consists of the following attributes:

- **itemType: STEPS** [int]
- **item_type: PORTS** [int]

  Provides a list of ports modelled in this session

- **item_type: SERVICES** [freetext]

  Provides a list of services modelled in this session

- **item_type: NODE**

  Defines a node included in the system laydown being simulated. It should consist of the following attributes:

  - **id** [int]: Unique ID for this YAML item
  - **name** [freetext]: Human-readable name of the component

  - **node_class** [enum]: Relates to the base type of the node. Can be SERVICE, ACTIVE or PASSIVE. PASSIVE nodes do not have an operating system or services. ACTIVE nodes have an operating system, but no services. SERVICE nodes have both an operating system and one or more services
  - **node_type** [enum]: Relates to the component type. Can be one of CCTV, SWITCH, COMPUTER, LINK, MONITOR, PRINTER, LOP, RTU, ACTUATOR or SERVER
  - **priority** [enum]: Provides a priority for each node. Can be one of P1, P2, P3, P4 or P5 (which P1 being the highest)
  - **hardware_state** [enum]: The initial hardware state of the node. Can be one of ON, OFF or RESETTING
  - **ip_address** [IP address]: The IP address of the component in format xxx.xxx.xxx.xxx
  - **software_state** [enum]: The intial state of the node operating system. Can be GOOD, PATCHING or COMPROMISED
  - **file_system_state** [enum]: The initial state of the node file system. Can be GOOD, CORRUPT, DESTROYED, REPAIRING or RESTORING
  - **services**: For each service associated with the node:
    * **name** [freetext]: Free-text name of the service, but must match one of the services defined for the system in the services list
    * **port** [int]: Integer value of the port related to this service, but must match one of the ports defined for the system in the ports list
    * **state** [enum]: The initial state of the service. Can be one of GOOD, PATCHING, COMPROMISED or OVERWHELMED

- **item_type: LINK**

  Defines a link included in the system laydown being simulated. It should consist of the following attributes:

  - **id** [int]: Unique ID for this YAML item
  - **name** [freetext]: Human-readable name of the component
  - **bandwidth** [int]: The bandwidth (in bits/s) of the link
  - **source** [int]: The ID of the source node
  - **destination** [int]: The ID of the destination node

- **item_type: GREEN_IER**

  Defines a green agent Information Exchange Requirement (IER). It should consist of:

  - **id** [int]: Unique ID for this YAML item
  - **start_step** [int]: The start step (in the episode) for this IER to begin
  - **end_step** [int]: The end step (in the episode) for this IER to finish
  - **load** [int]: The load (in bits/s) for this IER to apply to links
  - **protocol** [freetext]: The protocol to apply to the links. This must match a value in the services list
  - **port** [int]: The port that the protocol is running on. This must match a value in the ports list
  - **source** [int]: The ID of the source node

  - **destination** [int]: The ID of the destination node
  - **mission_criticality** [enum]: The mission criticality of this IER (with 5 being highest, 1 lowest)

- **item_type: RED_IER**

  Defines a red agent Information Exchange Requirement (IER). It should consist of:

  - **id** [int]: Unique ID for this YAML item
  - **start_step** [int]: The start step (in the episode) for this IER to begin
  - **end_step** [int]: The end step (in the episode) for this IER to finish
  - **load** [int]: The load (in bits/s) for this IER to apply to links
  - **protocol** [freetext]: The protocol to apply to the links. This must match a value in the services list
  - **port** [int]: The port that the protocol is running on. This must match a value in the ports list
  - **source** [int]: The ID of the source node
  - **destination** [int]: The ID of the destination node
  - **mission_criticality** [enum]: Not currently used. Default to 0

- **item_type: GREEN_POL**

  Defines a green agent pattern-of-life instruction. It should consist of:

  - **id** [int]: Unique ID for this YAML item
  - **start_step** [int]: The start step (in the episode) for this PoL to begin
  - **end_step** [int]: Not currently used. Default to same as start step
  - **nodeId** [int]: The ID of the node to apply the PoL to
  - **type** [enum]: The type of PoL to apply. Can be one of OPERATING, OS or SERVICE
  - **protocol** [freetext]: The protocol to be affected if SERVICE type is chosen. Must match a value in the services list
  - **state** [enuum]: The state to apply to the node (which represents the PoL change). Can be one of ON, OFF or RESETTING (for node state) or GOOD, PATCHING or COMPROMISED (for Software State) or GOOD, PATCHING, COMPROMISED or OVERWHELMED (for service state)

- **item_type: RED_POL**

  Defines a red agent pattern-of-life instruction. It should consist of:

  - **id** [int]: Unique ID for this YAML item
  - **start_step** [int]: The start step (in the episode) for this PoL to begin
  - **end_step** [int]: Not currently used. Default to same as start step
  - **targetNodeId** [int]: The ID of the node to apply the PoL to
  - **initiator** [enum]: What initiates the PoL. Can be DIRECT, IER or SERVICE
  - **type** [enum]: The type of PoL to apply. Can be one of OPERATING, OS or SERVICE
  - **protocol** [freetext]: The protocol to be affected if SERVICE type is chosen. Must match a value in the services list

  - **state** [enum]: The state to apply to the node (which represents the PoL change). Can be one of ON, OFF or RESETTING (for node state) or GOOD, PATCHING or COMPROMISED (for Software State) or GOOD, PATCHING, COMPROMISED or OVERWHELMED (for service state) or GOOD, CORRUPT, DESTROYED, REPAIRING or RESTORING (for file system state)
  - **sourceNodeId** [int] The ID of the source node containing the service to check (used for SERVICE initiator)
  - **sourceNodeService** [freetext]: The service on the source node to check (used for SERVICE initiator). Must match a value in the services list for this node
  - **sourceNodeServiceState** [enum]: The state of the source node service to check (used for SERVICE initiator). Can be one of GOOD, PATCHING, COMPROMISED or OVERWHELMED

- **item_type: ACL_RULE**

  Defines an initial Access Control List (ACL) rule. It should consist of:

  - **id** [int]: Unique ID for this YAML item
  - **permission** [enum]: Defines either an allow or deny rule. Value must be either DENY or ALLOW
  - **source** [IP address]: Defines the source IP address for the rule in xxx.xxx.xxx.xxx format
  - **destination** [IP address]: Defines the destination IP address for the rule in xxx.xxx.xxx.xxx format
  - **protocol** [freetext]: Defines the protocol for the rule. Must match a value in the services list
  - **port** [int]: Defines the port for the rule. Must match a value in the ports list
  - **position** [int]: Defines where to place the ACL rule in the list. Lower index or (higher up in the list) means they are checked first. Index starts at 0 (Python indexes).

3.4 Run a PrimAITE Session

3.4.1 Run

A PrimAITE session can be ran either with the `primaite session` command from the cli (See *primaite.cli.session()*), or by calling *primaite.main.run()* from a Python terminal or Jupyter Notebook. Both the `primaite session` and *primaite.main.run()* take a training config and a lay down config as parameters.

Listing 17: Unix CLI

```
cd ~/primaite/2.0.0
source ./.venv/bin/activate
primaite session --tc ./config/my_training_config.yaml --ldc ./config/my_lay_down_config.
↪yaml
```

Listing 18: Powershell CLI

```
cd ~\primaite\2.0.0
.\.venv\Scripts\activate
primaite session --tc .\config\my_training_config.yaml --ldc .\config\my_lay_down_config.
↪yaml
```

Listing 19: Python

```
from primaite.main import run

training_config = <path to training config yaml file>
lay_down_config = <path to lay down config yaml file>
run(training_config, lay_down_config)
```

When a session is ran, a session output sub-directory is created in the users app sessions directory (`~/primaite/2.0.0/sessions`). The sub-directory is formatted as such: `~/primaite/2.0.0/sessions/<yyyy-mm-dd>/<yyyy-mm-dd>_<hh-mm-dd>/`

For example, when running a session at 17:30:00 on 31st January 2023, the session will output to: `~/primaite/2.0.0/sessions/2023-01-31/2023-01-31_17-30-00/`.

`primaite session` can be ran in the terminal/command prompt without arguments. It will use the default configs in the directory `primaite/config/example_config`.

To run a PrimAITE session using legacy training or laydown config files, add the `--legacy-tc` and/or `legacy-ldc` options.

Listing 20: Unix CLI

```
cd ~/primaite/2.0.0
source ./.venv/bin/activate
primaite session --tc ./config/my_legacy_training_config.yaml --legacy-tc --ldc ./config/
↪my_legacy_lay_down_config.yaml --legacy-ldc
```

Listing 21: Powershell CLI

```
cd ~\primaite\2.0.0
.\.venv\Scripts\activate
primaite session --tc .\config\my_legacy_training_config.yaml --legacy-tc --ldc .\config\
↪my_legacy_lay_down_config.yaml --legacy-ldc
```

Listing 22: Python

```
from primaite.main import run

training_config = <path to legacy training config yaml file>
lay_down_config = <path to legacy lay down config yaml file>
run(training_config, lay_down_config, legacy_training_config=True, legacy_lay_down_
↪config=True)
```

3.4.2 Outputs

PrimAITE produces four types of outputs:

- Session Metadata
- Results
- Diagrams
- Saved agents (training checkpoints and a final trained agent)

Session Metadata

PrimAITE creates a `session_metadata.json` file that contains the following metadata:

- **uuid** - The UUID assigned to the session upon instantiation.
- **start_datetime** - The date & time the session started in iso format.
- **end_datetime** - The date & time the session ended in iso format.
- **learning**
  - **total_episodes** - The total number of training episodes completed.
  - **total_time_steps** - The total number of training time steps completed.
- **evaluation**
  - **total_episodes** - The total number of evaluation episodes completed.
  - **total_time_steps** - The total number of evaluation time steps completed.
- **env**
  - **training_config**
    - **All training config items**
  - **lay_down_config**
    - **All lay down config items**

Results

PrimAITE automatically creates two sets of results from each learning and evaluation session:

- Average reward per episode - a csv file listing the average reward for each episode of the session. This provides, for example, an indication of the change over a training session of the reward value
- All transactions - a csv file listing the following values for every step of every episode:
  - Timestamp
  - Episode number

- Step number
- Reward value
- Action taken (as presented by the blue agent on this step). Individual elements of the action space are presented in the format AS_X
- Initial observation space (what the blue agent observed when it decided its action)

Diagrams

- For each session, PrimAITE automatically creates a visualisation of the system / network lay down configuration.
- For each learning and evaluation task within the session, PrimAITE automatically plots the average reward per episode using PlotLY and saves it to the learning or evaluation subdirectory in the session directory.

Saved agents

For each training session, assuming the agent being trained implements the *save()* function and this function is called by the code, PrimAITE automatically saves the agent state.

Example Session Directory Structure

```
~/
└── primaite/
    └── 2.0.0/
        └── sessions/
            └── 2023-07-18/
                └── 2023-07-18_11-06-04/
                    ├── evaluation/
                    │   ├── all_transactions_2023-07-18_11-06-04.csv
                    │   ├── average_reward_per_episode_2023-07-18_11-06-04.csv
                    │   └── average_reward_per_episode_2023-07-18_11-06-04.png
                    ├── learning/
                    │   ├── all_transactions_2023-07-18_11-06-04.csv
                    │   ├── average_reward_per_episode_2023-07-18_11-06-04.csv
                    │   ├── average_reward_per_episode_2023-07-18_11-06-04.png
                    │   ├── checkpoints/
                    │   │   └── sb3ppo_10.zip
                    │   ├── SB3_PPO.zip
                    │   └── tensorboard_logs/
                    │       ├── PPO_1/
                    │       │   └── events.out.tfevents.1689674765.METD-9PMRFB3.42960.0
                    │       ├── PPO_2/
                    │       │   └── events.out.tfevents.1689674766.METD-9PMRFB3.42960.1
                    │       ├── PPO_3/
                    │       │   └── events.out.tfevents.1689674766.METD-9PMRFB3.42960.2
                    │       ├── PPO_4/
                    │       │   └── events.out.tfevents.1689674767.METD-9PMRFB3.42960.3
                    │       ├── PPO_5/
                    │       │   └── events.out.tfevents.1689674767.METD-9PMRFB3.42960.4
                    │       ├── PPO_6/
                    │       │   └── events.out.tfevents.1689674768.METD-9PMRFB3.42960.5
                    │       ├── PPO_7/
                    │       │   └── events.out.tfevents.1689674768.METD-9PMRFB3.42960.6
                    │       ├── PPO_8/
                    │       │   └── events.out.tfevents.1689674769.METD-9PMRFB3.42960.7
                    │       ├── PPO_9/
```

(continues on next page)

(continued from previous page)

```
                        │       │   └── events.out.tfevents.1689674770.METD-9PMRFB3.42960.8
                        │       └── PPO_10/
                        │           └── events.out.tfevents.1689674770.METD-9PMRFB3.42960.9
                        ├── network_2023-07-18_11-06-04.png
                        └── session_metadata.json
```

3.4.3 Loading a session

A previous session can be loaded by providing the **directory** of the previous session to either the `primaite session` command from the cli (See *primaite.cli.session()*), or by calling *primaite.main.run()* with session_path.

Listing 23: Unix CLI

```
cd ~/primaite/2.0.0
source ./.venv/bin/activate
primaite session --load "path/to/session"
```

Listing 24: Powershell CLI

```
cd ~\primaite\2.0.0
.\.venv\Scripts\activate
primaite session --load "path\to\session"
```

Listing 25: Python

```
from primaite.main import run

run(session_path=<previous session directory>)
```

When PrimAITE runs a loaded session, PrimAITE will output in the provided session directory

3.5 Custom Agents

3.5.1 Integrating a user defined blue agent

Note: If you are planning to implement custom RL agents into PrimAITE, you must use the project as a repository. If you install PrimAITE as a python package from wheel, custom agents are not supported.

PrimAITE has integration with Ray RLLib and StableBaselines3 agents. All agents interface with PrimAITE through an `primaite.agents.agent_abc.AgentSessionABC` which provides Input/Output of agent savefiles, as well as capturing and plotting performance metrics during training and evaluation. If you wish to integrate a custom blue agent, it is recommended to create a subclass of the *primaite.agents.agent_abc.AgentSessionABC* and implement the `__init__()`, `_setup()`, `_save_checkpoint()`, `learn()`, `evaluate()`, `_get_latest_checkpoint`, `load()`, and `save()` methods.

Below is a barebones example of a custom agent implementation:

```python
# src/primaite/agents/my_custom_agent.py

from primaite.agents.agent_abc import AgentSessionABC
from primaite.common.enums import AgentFramework, AgentIdentifier

class CustomAgent(AgentSessionABC):
    def __init__(self, training_config_path, lay_down_config_path):
        super().__init__(training_config_path, lay_down_config_path)
        assert self._training_config.agent_framework == AgentFramework.CUSTOM
        assert self._training_config.agent_identifier == AgentIdentifier.MY_AGENT
        self._setup()

    def _setup(self):
        super()._setup()
        self._env = Primaite(
            training_config_path=self._training_config_path,
            lay_down_config_path=self._lay_down_config_path,
            session_path=self.session_path,
            timestamp_str=self.timestamp_str,
    )
        self._agent = ... # your code to setup agent

    def _save_checkpoint(self):
        checkpoint_num = self._training_config.checkpoint_every_n_episodes
        episode_count = self._env.episode_count
        save_checkpoint = False
        if checkpoint_num:
            save_checkpoint = episode_count % checkpoint_num == 0
        # saves checkpoint if the episode count is not 0 and save_checkpoint flag was␣
↪set to true
        if episode_count and save_checkpoint:
            ...
            # your code to save checkpoint goes here.
            # The path should start with self.checkpoints_path and include the episode␣
↪number.

    def learn(self):
        ...
        # call your agent's learning function here.

        super().learn() # this will finalise learning and output session metadata
        self.save()

    def evaluate(self):
        ...
        # call your agent's evaluation function here.

        self._env.close()
        super().evaluate()

    def _get_latest_checkpoint(self):
        ...
```

(continues on next page)

(continued from previous page)

```
        # Load an agent from file.

    @classmethod
    def load(cls, path):
        ...
        # Create a CustomAgent object which loads model weights from file.

    def save(self):
        ...
        # Call your agent's function that saves it to a file
```

You will also need to modify `primaite.primaite_session.PrimaiteSession` and *primaite.common.enums* to capture your new agent identifiers.

```
# src/primaite/common/enums.py

class AgentIdentifier(Enum):
    """The Red Agent algo/class."""
    A2C = 1
    "Advantage Actor Critic"
    PPO = 2
    "Proximal Policy Optimization"
    HARDCODED = 3
    "The Hardcoded agents"
    DO_NOTHING = 4
    "The DoNothing agents"
    RANDOM = 5
    "The RandomAgent"
    DUMMY = 6
    "The DummyAgent"
    CUSTOM_AGENT = 7
    "Your custom agent"
```

```
# src/primaite_session.py

from primaite.agents.my_custom_agent import CustomAgent

# ...

    def setup(self):
    """Performs the session setup."""
    if self._training_config.agent_framework == AgentFramework.CUSTOM:
        _LOGGER.debug(f"PrimaiteSession Setup: Agent Framework = {AgentFramework.CUSTOM}
↪")
        if self._training_config.agent_identifier == AgentIdentifier.CUSTOM_AGENT:
            self._agent_session = CustomAgent(self._training_config_path, self._lay_down_
↪config_path)
        if self._training_config.agent_identifier == AgentIdentifier.HARDCODED:
            _LOGGER.debug(f"PrimaiteSession Setup: Agent Identifier =" f"
↪{AgentIdentifier.HARDCODED}")
            if self._training_config.action_type == ActionType.NODE:
                # Deterministic Hardcoded Agent with Node Action Space
```

(continues on next page)

(continued from previous page)

```
                self._agent_session = HardCodedNodeAgent(self._training_config_path,␣
↪self._lay_down_config_path)
```

Finally, specify your agent in your training config.

```
# ~/primaite/2.0.0/config/path/to/your/config_main.yaml

# Training Config File

agent_framework: CUSTOM
agent_identifier: CUSTOM_AGENT
random_red_agent: False
# ...
```

Now you can *run a primaite session* with your custom agent by passing in the custom `config_main`.

3.6 Simulation

3.6.1 Contents

3.6.1.1 Simulation Structure

The simulation is made up of many smaller components which are related to each other in a tree-like structure. At the top level, there is an object called the `SimulationController` _(doesn't exist yet)_, which has a physical network and a software controller for managing software and users.

Each node of the simulation 'tree' has responsibility for creating, deleting, and updating its direct descendants.

3.6.1.2 Actions

Agents can interact with the simulation by using actions. Actions are standardised with the `primaite.simulation.core.Action` class, which just holds a reference to two special functions.

1. The action function itself, it must accept a *request* parameters which is a list of strings that describe what the action should do. It must also accept a *context* dict which can house additional information surrounding the action. For example, the context will typically include information about which entity intiated the action.
2. A validator function. This function should return a boolean value that decides if the request is permitted or not. It uses the same paramters as the action function.

3.6.1.2.1 Action Permissions

When an agent tries to perform an action on a simulation component, that action will only be executed if the request is validated. For example, some actions can require that an agent is logged into an admin account. Each action defines its own permissions using an instance of `primaite.simulation.core.ActionPermissionValidator`. The below code snippet demonstrates usage of the `ActionPermissionValidator`.

```
from primaite.simulator.core import Action, ActionManager, SimComponent
from primaite.simulator.domain.controller import AccountGroup, GroupMembershipValidator
```

(continues on next page)

(continued from previous page)

```
class Smartphone(SimComponent):
    name: str
    apps = []

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.action_manager = ActionManager()

        self.action_manager.add_action(
            "reset_factory_settings",
            Action(
                func = lambda request, context: self.reset_factory_settings(),
                validator = GroupMembershipValidator([AccountGroup.DOMAIN_ADMIN]),
            ),
        )

    def reset_factory_settings(self):
        self.apps = []

phone = Smartphone(name="phone1")

# try to wipe the phone as a domain user, this will have no effect
phone.apply_action(["reset_factory_settings"], context={"request_source":{"groups":[
↪"DOMAIN_USER"]}})

# try to wipe the phone as an admin user, this will wipe the phone
phone.apply_action(["reset_factory_settings"], context={"request_source":{"groups":[
↪"DOMAIN_ADMIN"]}})
```

3.6.1.3 Base Hardware

The physical layer components are models of a NIC (Network Interface Card), SwitchPort, Node, Switch, and a Link. These components allow modelling of layer 1 (physical layer) in the OSI model and the nodes that connect to and transmit across layer 1.

3.6.1.3.1 NIC

The NIC class provides a realistic model of a Network Interface Card. The NIC acts as the interface between a Node and a Link, handling IP and MAC addressing, status, and sending/receiving frames.

3.6.1.3.1.1 Addressing

A NIC has both an IPv4 address and MAC address assigned:

- **ip_address** - The IPv4 address assigned to the NIC for communication on an IP network.
- **subnet_mask** - The subnet mask that defines the network subnet.
- **gateway** - The default gateway IP address for routing traffic beyond the local network.
- **mac_address** - A unique MAC address assigned to the NIC by the manufacturer.

3.6.1.3.1.2 Status

The status of the NIC is represented by:

- **enabled** - Indicates if the NIC is active/enabled or disabled/down. It must be enabled to send/receive frames.
- **connected_node** - The Node instance the NIC is attached to.
- **connected_link** - The Link instance the NIC is wired to.

3.6.1.3.1.3 Packet Capture

- **pcap** - A PacketCapture instance attached to the NIC for capturing all frames sent and received. This allows packet

capture and analysis.

3.6.1.3.1.4 Sending/Receiving Frames

The NIC can send and receive Frames to/from the connected Link:

- **send_frame()** - Sends a Frame through the NIC onto the attached Link.
- **receive_frame()** - Receives a Frame from the attached Link and processes it.

This allows a NIC to handle sending, receiving, and forwarding of network traffic at layer 2 of the OSI model. The Frames contain network data encapsulated with various protocol headers.

3.6.1.3.1.5 Basic Usage

```
nic1 = NIC(
    ip_address="192.168.0.100",
    subnet_mask="255.255.255.0",
    gateway="192.168.0.1"
)
nic1.enable()
frame = Frame(...)
nic1.send_frame(frame)
```

3.6.1.3.2 SwitchPort

The SwitchPort models a port on a network switch. It has similar attributes and methods to NIC for addressing, status, packet capture, sending/receiving frames, etc.

Key attributes:

- **port_num**: The port number on the switch.
- **connected_switch**: The switch to which this port belongs.

3.6.1.3.3 Node

The Node class represents a base node that communicates on the Network.

3.6.1.3.3.1 Network Interfaces

A Node will typically have one or more NICs attached to it for network connectivity:

- **nics** - A dictionary containing the NIC instances attached to the Node. NICs can be added/removed.

3.6.1.3.3.2 Configuration

- **hostname** - Configured hostname of the Node.
- **operating_state** - Current operating state like ON or OFF. The NICs will be enabled/disabled based on this.

3.6.1.3.3.3 Network Services

A Node runs various network services and components for handling traffic:

- **session_manager** - Handles establishing sessions to/from the Node.
- **software_manager** - Manages software and applications on the Node.
- **arp** - ARP cache for resolving IP addresses to MAC addresses.
- **icmp** - ICMP service for responding to pings and echo requests.
- **sys_log** - System log service for logging internal events and messages.

The SysLog provides a logging mechanism for the Node:

The SysLog records informational, warning, and error events that occur on the Node during simulation. This allows debugging and tracing program execution and network activity for each simulated Node. Other Node services like ARP and ICMP, along with custom Applications, services, and Processes will log to the SysLog.

3.6.1.3.3.4 Sending/Receiving

The Node handles sending and receiving Frames via its attached NICs:

- **send_frame()** - Sends a Frame to the network through one of the Node's NICs.
- **receive_frame()** - Receives a Frame from the network through a NIC. The Node then processes it appropriately based

on the protocols and payload.

3.6.1.3.3.5 Basic Usage

```
node1 = Node(hostname='server1')
node1.operating_state = NodeOperatingState.ON

nic1 = NIC()
node1.connect_nic(nic1)

Send a frame
frame = Frame(...)
node1.send_frame(frame)
```

The Node class brings together the NICs, configuration, and services to model a full network node that can send, receive, process, and forward traffic on a simulated network.

3.6.1.3.4 Switch

The Switch subclass models a network switch. It inherits from Node and acts at layer 2 of the OSI model to forward frames based on MAC addresses.

3.6.1.3.4.1 Inherits Node Capabilities

Since Switch subclasses Node, it inherits all capabilities from Node like:

- **Managing NICs**
- **Running network services like ARP, ICMP**
- **Sending and receiving frames**
- **Maintaining system logs**

3.6.1.3.4.2 Ports

A Switch has multiple ports implemented using SwitchPort instances:

- **switch_ports** - A dictionary mapping port numbers to SwitchPort instances.
- **num_ports** - The number of ports the Switch has.

3.6.1.3.4.3 Forwarding

A Switch forwards frames between ports based on the destination MAC:

- **dst_mac_table** - MAC address table that maps MACs to SwitchPorts.
- **forward_frame()** - Forwards a frame out the port associated with the destination MAC.

When a frame is received on a SwitchPort:

1. The source MAC address is extracted from the frame.
2. An entry is added to dst_mac_table that maps this source MAC to the SwitchPort it was received on.
3. When a frame with that destination MAC is received in the future, it will be forwarded out this SwitchPort.

This allows the Switch to dynamically build up a mapping table between MAC addresses and SwitchPorts based on traffic received. If no entry exists for a destination MAC, it floods the frame out all ports.

3.6.1.3.5 Link

The Link class represents a physical link or connection between two network endpoints like NICs or SwitchPorts.

3.6.1.3.5.1 Endpoints

A Link connects two endpoints:

- **endpoint_a** - The first endpoint, a NIC or SwitchPort.
- **endpoint_b** - The second endpoint, a NIC or SwitchPort.

3.6.1.3.5.2 Transmission

Links transmit Frames between the endpoints:

- **transmit_frame()** - Sends a Frame from one endpoint to the other.

Uses bandwidth/load properties to determine if transmission is possible.

3.6.1.3.5.3 Bandwidth & Load

- **bandwidth** - The total capacity of the Link in Mbps.
- **current_load** - The current bandwidth utilization of the Link in Mbps.

As Frames are sent over the Link, the load increases. The Link tracks if there is enough unused capacity to transmit a Frame based on its size and the current load.

3.6.1.3.5.4 Status

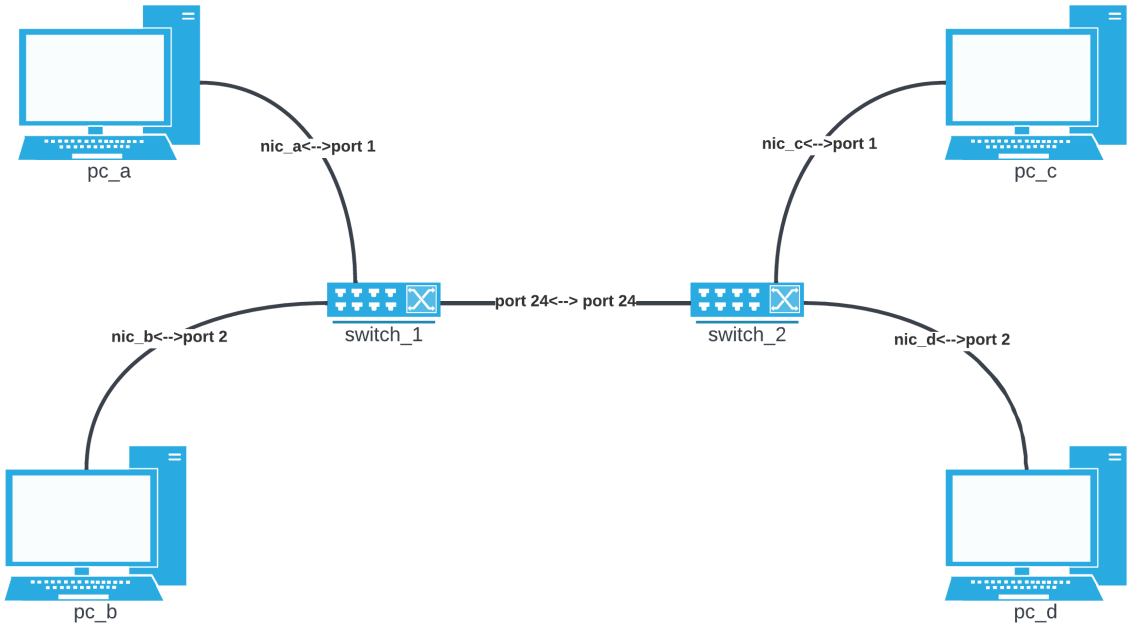
- **up** - Boolean indicating if the Link is currently up/active based on the endpoint status.
- **endpoint_up()/down()** - Notifies the Link when an endpoint goes up or down.

This allows the Link to realistically model the connection and transmission characteristics between two endpoints.

3.6.1.3.6 Putting it all Together

We'll now demonstrate how the nodes, NICs, switches, and links connect in a network, including full code examples and syslog extracts to illustrate the step-by-step process.

To demonstrate successful network communication between nodes and switches, we'll model a standard network with four PC's and two switches.



3.6.1.3.6.1 Create Nodes & NICs

First, we'll create the four nodes, each with a single NIC.

```
pc_a = Node(hostname="pc_a")
nic_a = NIC(ip_address="192.168.0.10", subnet_mask="255.255.255.0", gateway="192.168.0.1
↪")
pc_a.connect_nic(nic_a)
pc_a.power_on()

pc_b = Node(hostname="pc_b")
nic_b = NIC(ip_address="192.168.0.11", subnet_mask="255.255.255.0", gateway="192.168.0.1
↪")
pc_b.connect_nic(nic_b)
```

(continues on next page)

(continued from previous page)

```
pc_b.power_on()

pc_c = Node(hostname="pc_c")
nic_c = NIC(ip_address="192.168.0.12", subnet_mask="255.255.255.0", gateway="192.168.0.1
↪")
pc_c.connect_nic(nic_c)
pc_c.power_on()

pc_d = Node(hostname="pc_d")
nic_d = NIC(ip_address="192.168.0.13", subnet_mask="255.255.255.0", gateway="192.168.0.1
↪")
pc_d.connect_nic(nic_d)
pc_d.power_on()
```

This produces:

node_a NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
80:af:f2:f6:58:b7	102.169.0.10	255.255.255.0	192.168.0.1	100	Disabled

node_a sys log

```
2023-08-08 15:50:08,355 INFO: Connected NIC 80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,355 INFO: Turned on
```

node_b NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
98:ad:eb:7c:dc:cb	102.169.0.11	255.255.255.0	192.168.0.1	100	Disabled

node_b sys log

```
2023-08-08 15:50:08,357 INFO: Connected NIC 98:ad:eb:7c:dc:cb/192.168.0.11
2023-08-08 15:50:08,357 INFO: Turned on
```

node_c NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
bc:72:82:5d:82:a4	102.169.0.12	255.255.255.0	192.168.0.1	100	Disabled

node_c sys log

```
2023-08-08 15:50:08,358 INFO: Connected NIC bc:72:82:5d:82:a4/192.168.0.12
2023-08-08 15:50:08,358 INFO: Turned on
```

node_d NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
84:20:7c:ec:a5:c6	102.169.0.13	255.255.255.0	192.168.0.1	100	Disabled

node_d sys log

```
2023-08-08 15:50:08,359 INFO: Connected NIC 84:20:7c:ec:a5:c6/192.168.0.13
2023-08-08 15:50:08,360 INFO: Turned on
```

3.6.1.3.6.2 Create Switches

Next, we'll create two six-port switches:

```
switch_1 = Switch(hostname="switch_1", num_ports=6)
switch_1.power_on()

switch_2 = Switch(hostname="switch_2", num_ports=6)
switch_2.power_on()
```

This produces:

switch_1 MAC table

Port	MAC Address	Speed (Mbps)	Status
1	9d:ac:59:a0:05:13	100	Disabled
2	45:f5:8e:b6:f5:d3	100	Disabled
3	ef:f5:b9:28:cb:ae	100	Disabled
4	88:76:0a:72:fc:14	100	Disabled
5	79:de:da:bd:e2:ba	100	Disabled
6	91:d5:83:a0:02:f2	100	Disabled

switch_1 sys log

```
2023-08-08 15:50:08,373 INFO: Turned on
```

switch_2 MAC table

Port	MAC Address	Speed (Mbps)	Status
1	aa:58:fa:66:d7:be	100	Disabled
2	72:d2:1e:88:e9:45	100	Disabled
3	8a:fc:2a:56:d5:c5	100	Disabled
4	fb:b5:9a:04:4a:49	100	Disabled
5	88:aa:48:d0:21:9e	100	Disabled
6	96:77:39:d1:de:44	100	Disabled

switch_2 sys log

```
2023-08-08 15:50:08,374 INFO: Turned on
```

3.6.1.3.6.3 Create Links

Finally, we'll create the five links that connect the nodes and the switches:

```
link_nic_a_switch_1 = Link(endpoint_a=nic_a, endpoint_b=switch_1.switch_ports[1])
link_nic_b_switch_1 = Link(endpoint_a=nic_b, endpoint_b=switch_1.switch_ports[2])
link_nic_c_switch_2 = Link(endpoint_a=nic_c, endpoint_b=switch_2.switch_ports[1])
link_nic_d_switch_2 = Link(endpoint_a=nic_d, endpoint_b=switch_2.switch_ports[2])
link_switch_1_switch_2 = Link(
    endpoint_a=switch_1.switch_ports[6], endpoint_b=switch_2.switch_ports[6]
)
```

This produces:

node_a NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
80:af:f2:f6:58:b7	102.169.0.10	255.255.255.0	192.168.0.1	100	Enabled

node_a sys log

```
2023-08-08 15:50:08,355 INFO: Connected NIC 80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,355 INFO: Turned on
2023-08-08 15:50:08,355 INFO: NIC 80:af:f2:f6:58:b7/192.168.0.10 enabled
```

node_b NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
98:ad:eb:7c:dc:cb	102.169.0.11	255.255.255.0	192.168.0.1	100	Enabled

node_b sys log

```
2023-08-08 15:50:08,357 INFO: Connected NIC 98:ad:eb:7c:dc:cb/192.168.0.11
2023-08-08 15:50:08,357 INFO: Turned on
2023-08-08 15:50:08,357 INFO: NIC 98:ad:eb:7c:dc:cb/192.168.0.11 enabled
```

node_c NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
bc:72:82:5d:82:a4	102.169.0.12	255.255.255.0	192.168.0.1	100	Enabled

node_c sys log

```
2023-08-08 15:50:08,358 INFO: Connected NIC bc:72:82:5d:82:a4/192.168.0.12
2023-08-08 15:50:08,358 INFO: Turned on
2023-08-08 15:50:08,358 INFO: NIC bc:72:82:5d:82:a4/192.168.0.12 enabled
```

node_d NIC table

MAC Address	IP Address	Subnet Mask	Default Gateway	Speed (Mbps)	Status
84:20:7c:ec:a5:c6	102.169.0.13	255.255.255.0	192.168.0.1	100	Enabled

node_d sys log

```
2023-08-08 15:50:08,359 INFO: Connected NIC 84:20:7c:ec:a5:c6/192.168.0.13
2023-08-08 15:50:08,360 INFO: Turned on
2023-08-08 15:50:08,360 INFO: NIC 84:20:7c:ec:a5:c6/192.168.0.13 enabled
```

switch_1 MAC table

Port	MAC Address	Speed (Mbps)	Status
1	9d:ac:59:a0:05:13	100	Enabled
2	45:f5:8e:b6:f5:d3	100	Enabled
3	ef:f5:b9:28:cb:ae	100	Disabled
4	88:76:0a:72:fc:14	100	Disabled
5	79:de:da:bd:e2:ba	100	Disabled
6	91:d5:83:a0:02:f2	100	Enabled

switch_1 sys log

```
2023-08-08 15:50:08,373 INFO: Turned on
2023-08-08 15:50:08,378 INFO: SwitchPort 9d:ac:59:a0:05:13 enabled
2023-08-08 15:50:08,380 INFO: SwitchPort 45:f5:8e:b6:f5:d3 enabled
2023-08-08 15:50:08,384 INFO: SwitchPort 91:d5:83:a0:02:f2 enabled
```

switch_2 MAC table

Port	MAC Address	Speed (Mbps)	Status
1	aa:58:fa:66:d7:be	100	Enabled
2	72:d2:1e:88:e9:45	100	Enabled
3	8a:fc:2a:56:d5:c5	100	Disabled
4	fb:b5:9a:04:4a:49	100	Disabled
5	88:aa:48:d0:21:9e	100	Disabled
6	96:77:39:d1:de:44	100	Enabled

switch_2 sys log

```
2023-08-08 15:50:08,374 INFO: Turned on
2023-08-08 15:50:08,381 INFO: SwitchPort aa:58:fa:66:d7:be enabled
2023-08-08 15:50:08,383 INFO: SwitchPort 72:d2:1e:88:e9:45 enabled
2023-08-08 15:50:08,384 INFO: SwitchPort 96:77:39:d1:de:44 enabled
```

3.6.1.3.6.4 Perform Ping

Now with the network setup and operational, we can perform a ping to confirm that communication between nodes over a switched network is possible. In the below example, we ping 192.168.0.13 (node_d) from node_a:

```
pc_a.ping("192.168.0.13")
```

This produces:

node_a sys log

```
2023-08-08 15:50:08,355 INFO: Connected NIC 80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,355 INFO: Turned on
2023-08-08 15:50:08,355 INFO: NIC 80:af:f2:f6:58:b7/192.168.0.10 enabled
2023-08-08 15:50:08,406 INFO: Attempting to ping 192.168.0.13
2023-08-08 15:50:08,406 INFO: No entry in ARP cache for 192.168.0.13
2023-08-08 15:50:08,406 INFO: Sending ARP request from NIC 80:af:f2:f6:58:b7/192.168.0.
↪10 for ip 192.168.0.13
2023-08-08 15:50:08,413 INFO: Received ARP response for 192.168.0.13 from␣
↪84:20:7c:ec:a5:c6 via NIC 80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,413 INFO: Adding ARP cache entry for 84:20:7c:ec:a5:c6/192.168.0.13␣
↪via NIC 80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,415 INFO: Sending echo request to 192.168.0.13
2023-08-08 15:50:08,417 INFO: Received echo reply from 192.168.0.13
2023-08-08 15:50:08,419 INFO: Sending echo request to 192.168.0.13
2023-08-08 15:50:08,421 INFO: Received echo reply from 192.168.0.13
2023-08-08 15:50:08,422 INFO: Sending echo request to 192.168.0.13
2023-08-08 15:50:08,424 INFO: Received echo reply from 192.168.0.13
2023-08-08 15:50:08,425 INFO: Sending echo request to 192.168.0.13
2023-08-08 15:50:08,427 INFO: Received echo reply from 192.168.0.13
```

node_b sys log

```
2023-08-08 15:50:08,357 INFO: Connected NIC 98:ad:eb:7c:dc:cb/192.168.0.11
2023-08-08 15:50:08,357 INFO: Turned on
2023-08-08 15:50:08,357 INFO: NIC 98:ad:eb:7c:dc:cb/192.168.0.11 enabled
2023-08-08 15:50:08,410 INFO: Received ARP request for 192.168.0.13 from␣
↪80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,410 INFO: Ignoring ARP request for 192.168.0.13
```

node_c sys log

```
2023-08-08 15:50:08,358 INFO: Connected NIC bc:72:82:5d:82:a4/192.168.0.12
2023-08-08 15:50:08,358 INFO: Turned on
2023-08-08 15:50:08,358 INFO: NIC bc:72:82:5d:82:a4/192.168.0.12 enabled
2023-08-08 15:50:08,411 INFO: Received ARP request for 192.168.0.13 from␣
↪80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,411 INFO: Ignoring ARP request for 192.168.0.13
```

node_d sys log

```
2023-08-08 15:50:08,359 INFO: Connected NIC 84:20:7c:ec:a5:c6/192.168.0.13
2023-08-08 15:50:08,360 INFO: Turned on
2023-08-08 15:50:08,360 INFO: NIC 84:20:7c:ec:a5:c6/192.168.0.13 enabled
```

(continues on next page)

(continued from previous page)

```
2023-08-08 15:50:08,412 INFO: Received ARP request for 192.168.0.13 from␣
↪80:af:f2:f6:58:b7/192.168.0.10
2023-08-08 15:50:08,412 INFO: Adding ARP cache entry for 80:af:f2:f6:58:b7/192.168.0.10␣
↪via NIC 84:20:7c:ec:a5:c6/192.168.0.13
2023-08-08 15:50:08,412 INFO: Sending ARP reply from 84:20:7c:ec:a5:c6/192.168.0.13 to␣
↪192.168.0.10/80:af:f2:f6:58:b7
2023-08-08 15:50:08,416 INFO: Received echo request from 192.168.0.10
2023-08-08 15:50:08,417 INFO: Sending echo reply to 192.168.0.10
2023-08-08 15:50:08,420 INFO: Received echo request from 192.168.0.10
2023-08-08 15:50:08,420 INFO: Sending echo reply to 192.168.0.10
2023-08-08 15:50:08,423 INFO: Received echo request from 192.168.0.10
2023-08-08 15:50:08,423 INFO: Sending echo reply to 192.168.0.10
2023-08-08 15:50:08,426 INFO: Received echo request from 192.168.0.10
2023-08-08 15:50:08,426 INFO: Sending echo reply to 192.168.0.10
```

switch_1 sys log

```
2023-08-08 15:50:08,373 INFO: Turned on
2023-08-08 15:50:08,378 INFO: SwitchPort 9d:ac:59:a0:05:13 enabled
2023-08-08 15:50:08,380 INFO: SwitchPort 45:f5:8e:b6:f5:d3 enabled
2023-08-08 15:50:08,384 INFO: SwitchPort 91:d5:83:a0:02:f2 enabled
2023-08-08 15:50:08,409 INFO: Added MAC table entry: Port 1 -> 80:af:f2:f6:58:b7
2023-08-08 15:50:08,413 INFO: Added MAC table entry: Port 6 -> 84:20:7c:ec:a5:c6
```

switch_2 sys log

```
2023-08-08 15:50:08,374 INFO: Turned on
2023-08-08 15:50:08,381 INFO: SwitchPort aa:58:fa:66:d7:be enabled
2023-08-08 15:50:08,383 INFO: SwitchPort 72:d2:1e:88:e9:45 enabled
2023-08-08 15:50:08,384 INFO: SwitchPort 96:77:39:d1:de:44 enabled
2023-08-08 15:50:08,411 INFO: Added MAC table entry: Port 6 -> 80:af:f2:f6:58:b7
2023-08-08 15:50:08,412 INFO: Added MAC table entry: Port 2 -> 84:20:7c:ec:a5:c6
```

3.6.1.4 Transport Layer to Data Link Layer

From the OSI Model, the transport layer (layer 4) through to the data link layer (layer 2) have been loosely modelled to provide somewhat realistic network Frame generation.

3.6.1.4.1 Transport Layer (Layer 4)

UDPHeader: Represents a UDP header for the transport layer of a Network Frame. It includes source and destination ports. UDP (User Datagram Protocol) is a connectionless and unreliable transport protocol used for data transmission.

TCPFlags: Enum representing TCP control flags used in a TCP connection, such as SYN, ACK, FIN, and RST. TCP (Transmission Control Protocol) is a connection-oriented and reliable transport protocol used for establishing and maintaining data streams.

TCPHeader: Represents a TCP header for the transport layer of a Network Frame. It includes source and destination ports and TCP flags. This header is used for establishing and managing TCP connections.

3.6.1.4.2 Network Layer (Layer 3)

IPProtocol: Enum representing transport layer protocols in the IP header, such as TCP, UDP, and ICMP. It is used to indicate the type of transport layer protocol being used in the IP header.

Precedence: Enum representing the Precedence levels in Quality of Service (QoS) for IP packets. It is used to specify the priority of IP packets for Quality of Service handling.

ICMPType: Enumeration of common ICMP (Internet Control Message Protocol) types. It defines various types of ICMP messages used for network troubleshooting and error reporting.

ICMPPacket: Models an ICMP header and includes ICMP type, code, identifier, and sequence number. It is used to create ICMP packets for network control and error reporting.

IPPacket: Represents the IP layer of a network frame. It includes source and destination IP addresses, protocol (TCP/UDP/ICMP), Time to Live (TTL), and Precedence for QoS. This header is used to route data packets across the network based on IP addresses.

3.6.1.4.3 PrimAITE Layer (Custom Layer)

The PrimAITE layer has a custom header represented by the `PrimaiteHeader` class. It is designed to carry PrimAITE-specific metadata required for reinforcement learning (RL) purposes.

PrimaiteHeader: This is a custom header for carrying PrimAITE-specific metadata. It contains the following fields:

- **agent_source:** Enum representing the agent source of the transmission, such as RED, GREEN, or BLUE. This field helps identify the source or category of the data transmission.
- **data_status:** Enum representing the status of the data in transmission, such as GOOD, COMPROMISED, or CORRUPT. This field indicates the integrity of the data being transmitted.

3.6.1.4.4 Data Link Layer (Layer 2)

ARPEntry: Represents an entry in the ARP cache. It consists of the following fields:

- **mac_address:** The MAC address associated with the IP address.
- **nic_uuid:** The NIC (Network Interface Card) UUID through which the NIC with the IP address is reachable.

ARPPacket: Represents the ARP layer of a network frame, and it includes the following fields:

- **request:** ARP operation. Set to True for a request and False for a reply.
- **sender_mac_addr:** Sender's MAC address.
- **sender_ip:** Sender's IP address (IPv4 format).
- **target_mac_addr:** Target's MAC address.
- **target_ip:** Target's IP address (IPv4 format).

EthernetHeader: Represents the Ethernet layer of a network frame. It includes source and destination MAC addresses. This header is used to identify the physical hardware addresses of devices on a local network.

Frame: Represents a complete network frame with all layers. It includes an `EthernetHeader`, an `IPPacket`, an optional `TCPHeader`, `UDPHeader`, or `ICMPPacket`, a `PrimaiteHeader` and an optional payload. This class combines all the headers and data to create a complete network frame that can be sent over the network and used in the PrimAITE simulation.

3.6.1.4.5 Basic Usage

3.6.1.4.5.1 TCP SYN Frame

Here we will model a TCP synchronize request from a port 80 on the host 192.168.0.100 which has a NIC with a MAC address of 'aa:bb:cc:dd:ee:ff' to port 8080 on the host 10.0.0.10 which has a NIC with a MAC address of '11:22:33:44:55:66'.

```
from primaite.simulator.network.transmission.data_link_layer import EthernetHeader, Frame
from primaite.simulator.network.transmission.network_layer import IPPacket, IPProtocol
from primaite.simulator.network.transmission.transport_layer import TCPFlags, TCPHeader

# Transport Layer
tcp_header = TCPHeader(
    src_port=80,
    dst_port=8080,
    flags=[TCPFlags.SYN]
)

# Network Layer
ip_packet = IPPacket(
    src_ip="192.168.0.100",
    dst_ip="10.0.0.10",
    protocol=IPProtocol.TCP
)
# Data Link Layer
ethernet_header = EthernetHeader(
    src_mac_addr="aa:bb:cc:dd:ee:ff",
    dst_mac_addr="11:22:33:44:55:66"
)

frame = Frame(
    ethernet=ethernet_header,
    ip=ip_packet,
    tcp=tcp_header,
)
```

This produces the following `Frame` (displayed in json format)

```
{
    "ethernet": {
        "src_mac_addr": "aa:bb:cc:dd:ee:ff",
        "dst_mac_addr": "11:22:33:44:55:66"
    },
    "ip": {
        "src_ip": "192.168.0.100",
        "dst_ip": "10.0.0.10",
        "protocol": "tcp",
        "ttl": 64,
        "precedence": 0
    },
    "tcp": {
        "src_port": 80,
```

(continues on next page)

(continued from previous page)

```
        "dst_port": 8080,
        "flags": [
            1
        ]
    },
    "primaite": {
        "agent_source": 2,
        "data_status": 1
    }
}
```

3.7 primaite

Functions

getLogger	Get a PrimAITE logger.

3.7.1 primaite.getLogger

primaite.**getLogger**(*name: str*) → Logger

Get a PrimAITE logger.

Parameters

name – The logger name. Use `__name__`.

Returns

An instance of `logging.Logger` with the PrimAITE logging config.

primaite.acl	Access Control List.
primaite.agents	Common interface between RL agents from different libraries and PrimAITE.
primaite.cli	Provides a CLI using Typer as an entry point.
primaite.common	Objects which are shared between many PrimAITE modules.
primaite.config	Configuration parameters for running experiments.
primaite.data_viz	Utility to generate plots of sessions metrics after PrimAITE.
primaite.environment	Gym/Gymnasium environment for RL agents consisting of a simulated computer network.
primaite.exceptions	
primaite.links	Network connections between nodes in the simulation.
primaite.main	The main PrimAITE session runner module.
primaite.nodes	Nodes represent network hosts in the simulation.
primaite.notebooks	Contains default jupyter notebooks which demonstrate PrimAITE functionality.
primaite.pol	Pattern of Life- Represents the actions of users on the network.
primaite.primaite_session	Main entry point to PrimAITE.
primaite.setup	Utilities to prepare the user's data folders.
primaite.simulator	
primaite.transactions	Record data of the system's state and agent's observations and actions.
primaite.utils	Utilities for PrimAITE.

3.7.2 primaite.acl

Access Control List. Models firewall functionality.

primaite.acl.access_control_list	A class that implements the access control list implementation for the network.
primaite.acl.acl_rule	A class that implements an access control list rule.

3.7.2.1 primaite.acl.access_control_list

A class that implements the access control list implementation for the network.

Classes

AccessControlList	Access Control List class.

3.7.2.1.1 primaite.acl.access_control_list.AccessControlList

class primaite.acl.access_control_list.**AccessControlList**(*implicit_permission:* RulePermissionType, *max_acl_rules: int*)

Bases: `object`

Access Control List class.

Methods

add_rule	Adds a new rule.
check_address_match	Checks for IP address matches.
get_dictionary_hash	Produces a hash value for a rule.
get_relevant_rules	Get all ACL rules that relate to the given arguments.
is_blocked	Checks for rules that block a protocol / port.
remove_all_rules	Removes all rules.
remove_rule	Removes a rule.

Attributes

acl	Public access method for private _acl.

__init__(*implicit_permission:* RulePermissionType, *max_acl_rules: int*) → None

Init.

property acl: List[*ACLRule* | None]

Public access method for private _acl.

add_rule(*_permission:* RulePermissionType, *_source_ip: str*, *_dest_ip: str*, *_protocol: str*, *_port: str*, *_position: str*) → None

Adds a new rule.

Args:

_permission: the permission value (e.g. “ALLOW” or “DENY”) _source_ip: the source IP address _dest_ip: the destination IP address _protocol: the protocol _port: the port _position: position to insert ACL rule into ACL list (starting from index 1 and NOT 0)

check_address_match(*_rule:* ACLRule, *_source_ip_address: str*, *_dest_ip_address: str*) → bool

Checks for IP address matches.

Parameters

- **_rule** (ACLRule) – The rule object to check
- **_source_ip_address** (*str*) – Source IP address to compare
- **_dest_ip_address** (*str*) – Destination IP address to compare

Returns
True if there is a match, otherwise False.

Return type
bool

get_dictionary_hash(*_permission:* RulePermissionType, *_source_ip: str*, *_dest_ip: str*, *_protocol: str*, *_port: str*) → int

Produces a hash value for a rule.

Args:
_permission: the permission value (e.g. "ALLOW" or "DENY") _source_ip: the source IP address _dest_ip: the destination IP address _protocol: the protocol _port: the port

Returns:
Hash value based on rule parameters.

get_relevant_rules(*_source_ip_address: str*, *_dest_ip_address: str*, *_protocol: str*, *_port: str*) → Dict[int, *ACLRule*]

Get all ACL rules that relate to the given arguments.

Parameters

- **_source_ip_address** – the source IP address to check
- **_dest_ip_address** – the destination IP address to check
- **_protocol** – the protocol to check
- **_port** – the port to check

Returns
Dictionary of all ACL rules that relate to the given arguments

Return type
Dict[int, *ACLRule*]

is_blocked(*_source_ip_address: str*, *_dest_ip_address: str*, *_protocol: str*, *_port: str*) → bool

Checks for rules that block a protocol / port.

Args:
_source_ip_address: the source IP address to check _dest_ip_address: the destination IP address to check _protocol: the protocol to check _port: the port to check

Returns:
Indicates block if all conditions are satisfied.

remove_all_rules() → None

Removes all rules.

remove_rule(*_permission: RulePermissionType*, *_source_ip: str*, *_dest_ip: str*, *_protocol: str*, *_port: str*) → None

Removes a rule.

Args:
_permission: the permission value (e.g. "ALLOW" or "DENY") _source_ip: the source IP address _dest_ip: the destination IP address _protocol: the protocol _port: the port

3.7.2.2 primaite.acl.acl_rule

A class that implements an access control list rule.

Classes

ACLRule	Access Control List Rule class.

3.7.2.2.1 primaite.acl.acl_rule.ACLRule

class primaite.acl.acl_rule.**ACLRule**(*_permission: RulePermissionType*, *_source_ip: str*, *_dest_ip: str*, *_protocol: str*, *_port: str*)

Bases: `object`

Access Control List Rule class.

Methods

get_dest_ip	Gets the desintation IP address attribute.
get_permission	Gets the permission attribute.
get_port	Gets the port attribute.
get_protocol	Gets the protocol attribute.
get_source_ip	Gets the source IP address attribute.

__init__(*_permission: RulePermissionType*, *_source_ip: str*, *_dest_ip: str*, *_protocol: str*, *_port: str*) → None

Initialise an ACL Rule.

Parameters

- **_permission** – The permission (ALLOW or DENY)
- **_source_ip** – The source IP address
- **_dest_ip** – The destination IP address
- **_protocol** – The rule protocol
- **_port** – The rule port

`get_dest_ip()` → str

Gets the desintation IP address attribute.

Returns:
Returns destination IP address attribute

`get_permission()` → str

Gets the permission attribute.

Returns:
Returns permission attribute

`get_port()` → str

Gets the port attribute.

Returns:
Returns port attribute

`get_protocol()` → str

Gets the protocol attribute.

Returns:
Returns protocol attribute

`get_source_ip()` → str

Gets the source IP address attribute.

Returns:
Returns source IP address attribute

3.7.3 primaite.agents

Common interface between RL agents from different libraries and PrimAITE.

primaite.agents.agent_abc
primaite.agents.hardcoded_abc
primaite.agents.hardcoded_acl
primaite.agents.hardcoded_node
primaite.agents.rllib
primaite.agents.sb3
primaite.agents.simple
primaite.agents.utils

3.7.3.1 primaite.agents.agent_abc

Functions

get_session_path	Get the directory path the session will output to.

3.7.3.1.1 primaite.agents.agent_abc.get_session_path

primaite.agents.agent_abc.**get_session_path**(*session_timestamp: datetime*) → Path

Get the directory path the session will output to.

This is set in the format of:
~/primaite/2.0.0/sessions/<yyyy-mm-dd>/<yyyy-mm-dd>_<hh-mm-ss>.

Parameters
session_timestamp – This is the datetime that the session started.

Returns
The session directory path.

Classes

AgentSessionABC	An ABC that manages training and/or evaluation of agents in PrimAITE.

3.7.3.1.2 primaite.agents.agent_abc.AgentSessionABC

class primaite.agents.agent_abc.**AgentSessionABC**(*training_config_path: str | Path | None = None*, *lay_down_config_path: str | Path | None = None*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*)

Bases: ABC

An ABC that manages training and/or evaluation of agents in PrimAITE.

This class cannot be directly instantiated and must be inherited from with all implemented abstract methods implemented.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.
session_path	The Session path

abstract __init__(*training_config_path: str | Path | None = None*, *lay_down_config_path: str | Path | None = None*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*) → None

Initialise an agent session from config files, or load a previous session.

If training configuration and laydown configuration are provided with a session path, the session path will be used.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.
- **legacy_training_config** – True if the training config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **legacy_lay_down_config** – True if the lay_down config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **session_path** – directory path of the session to load

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

abstract evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters

kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

abstract export() → None

Export the agent to transportable file format.

abstract learn(***kwargs: Any*) → None

Train the agent.

Parameters

kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

load(*path: str | Path*) → None

Load an agent from file.

abstract save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.2 primaite.agents.hardcoded_abc

Classes

HardCodedAgentSessionABC	An Agent Session ABC for evaluation deterministic agents.

3.7.3.2.1 primaite.agents.hardcoded_abc.HardCodedAgentSessionABC

class primaite.agents.hardcoded_abc.**HardCodedAgentSessionABC**(*training_config_path: str | Path | None = '', lay_down_config_path: str | Path | None = '', session_path: str | Path | None = None*)

Bases: *AgentSessionABC*

An Agent Session ABC for evaluation deterministic agents.

This class cannot be directly instantiated and must be inherited from with all implemented abstract methods implemented.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.3 primaite.agents.hardcoded_acl

Classes

HardCodedACLAgent	An Agent Session class that implements a deterministic ACL agent.

3.7.3.3.1 primaite.agents.hardcoded_acl.HardCodedACLAgent

class primaite.agents.hardcoded_acl.**HardCodedACLAgent**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*)

Bases: *HardCodedAgentSessionABC*

An Agent Session class that implements a deterministic ACL agent.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
get_allow_acl_rules	List ALLOW rules relating to specified nodes.
get_allow_acl_rules_for_ier	Get all allowing ACL rules for an IER.
get_blocked_green_iers	Get blocked green IERs.
get_blocking_acl_rules_for_ier	Get blocking ACL rules for an IER.
get_deny_acl_rules	List DENY rules relating to specified nodes.
get_matching_acl_rules	Filter ACL rules to only those which are relevant to the specified nodes.
get_matching_acl_rules_for_ier	Get list of ACL rules which are relevant to an IER.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters

kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

get_allow_acl_rules(*source_node_id: int*, *dest_node_id: str*, *protocol: int*, *port: str*, *acl:* AccessControlList, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *services_list: List[str]*) → Dict[int, *ACLRule*]

List ALLOW rules relating to specified nodes.

Parameters

- **source_node_id** (*int*) – Source node id
- **dest_node_id** (*str*) – Destination node
- **protocol** (*int*) – Network protocol
- **port** (*str*) – Port
- **acl** (AccessControlList) – Firewall ruleset which is applied to the network
- **nodes** (*Dict[str, NodeUnion]*) – The simulation's node store
- **services_list** (*List[str]*) – Services list

Returns
Filtered ACL Rule directory which includes only those rules which affect the specified source and desination nodes

Return type
Dict[str, *ACLRule*]

`get_allow_acl_rules_for_ier`(*ier:* IER, *acl:* AccessControlList, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*) → Dict[int, *ACLRule*]

Get all allowing ACL rules for an IER.

Parameters

- **ier** (IER) – Information Exchange Request to query against the ACL list
- **acl** (AccessControlList) – Firewall rules
- **nodes** (*Dict[str,NodeUnion]*) – Nodes in the network

Returns
description

Return type
type

`get_blocked_green_iers`(*green_iers: Dict[str,* IER*]*, *acl:* AccessControlList, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*) → Dict[str, *IER*]

Get blocked green IERs.

Parameters

- **green_iers** (*Dict[str, IER]*) – Green IERs to check for being
- **acl** (AccessControlList) – Firewall rules
- **nodes** (*Dict[str,NodeUnion]*) – Nodes in the network

Returns
Same as *green_iers* input dict, but filtered to only contain the blocked ones.

Return type
Dict[str, *IER*]

`get_blocking_acl_rules_for_ier`(*ier:* IER, *acl:* AccessControlList, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*) → Dict[int, *ACLRule*]

Get blocking ACL rules for an IER.

> **Warning:** Can return empty dict but IER can still be blocked by default (No ALLOW rule, therefore blocked).

Parameters

- **ier** (IER) – Information Exchange Request to query against the ACL list
- **acl** (AccessControlList) – Firewall rules
- **nodes** (*Dict[str,NodeUnion]*) – Nodes in the network

Returns
description

Return type
type

get_deny_acl_rules(*source_node_id: int*, *dest_node_id: str*, *protocol: int*, *port: str*, *acl:* AccessControlList, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *services_list: List[str]*) → Dict[int, *ACLRule*]

List DENY rules relating to specified nodes.

Parameters

- **source_node_id** (*int*) – Source node id
- **dest_node_id** (*str*) – Destination node
- **protocol** (*int*) – Network protocol
- **port** (*str*) – Port
- **acl** (AccessControlList) – Firewall ruleset which is applied to the network
- **nodes** (*Dict[str, NodeUnion]*) – The simulation's node store
- **services_list** (*List[str]*) – Services list

Returns
Filtered ACL Rule directory which includes only those rules which affect the specified source and desination nodes

Return type
Dict[str, *ACLRule*]

get_matching_acl_rules(*source_node_id: str*, *dest_node_id: str*, *protocol: str*, *port: str*, *acl:* AccessControlList, *nodes: Dict[str,* ServiceNode | ActiveNode*]*, *services_list: List[str]*) → Dict[int, *ACLRule*]

Filter ACL rules to only those which are relevant to the specified nodes.

Parameters

- **source_node_id** (*str*) – Source node
- **dest_node_id** (*str*) – Destination nodes
- **protocol** (*str*) – Network protocol
- **port** (*str*) – Network port
- **acl** (AccessControlList) – Access Control list which will be filtered
- **nodes** (*Dict[str, Union[*ServiceNode, ActiveNode*]]*) – The environment's node directory.
- **services_list** (*List[str]*) – List of services registered for the environment.

Returns
Filtered version of 'acl'

Return type
Dict[str, *ACLRule*]

get_matching_acl_rules_for_ier(*ier:* IER, *acl:* AccessControlList, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*) → Dict[int, *ACLRule*]

Get list of ACL rules which are relevant to an IER.

Parameters

- **ier** (IER) – Information Exchange Request to query against the ACL list
- **acl** (AccessControlList) – Firewall rules
- **nodes** (*Dict[str,NodeUnion]*) – Nodes in the network

Returns
description

Return type
type

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.4 primaite.agents.hardcoded_node

Classes

HardCodedNodeAgent	An Agent Session class that implements a deterministic Node agent.

3.7.3.4.1 primaite.agents.hardcoded_node.HardCodedNodeAgent

class primaite.agents.hardcoded_node.**HardCodedNodeAgent**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*)

Bases: *HardCodedAgentSessionABC*

An Agent Session class that implements a deterministic Node agent.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.5 primaite.agents.rllib

3.7.3.6 primaite.agents.sb3

Classes

SB3Agent	An AgentSession class that implements a Stable Baselines3 agent.

3.7.3.6.1 primaite.agents.sb3.SB3Agent

class primaite.agents.sb3.**SB3Agent**(*training_config_path: str | Path | None = None*, *lay_down_config_path: str | Path | None = None*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*)

Bases: *AgentSessionABC*

An AgentSession class that implements a Stable Baselines3 agent.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = None*, *lay_down_config_path: str | Path | None = None*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*) → None

Initialise the SB3 Agent training session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.
- **legacy_training_config** – True if the training config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **legacy_lay_down_config** – True if the lay_down config file is a legacy file from PrimAITE < 2.0, otherwise False.

Raises

- **ValueError** – If the training config contains an unexpected value for agent_framework (should be “SB3”)
- **ValueError** – If the training config contains an unexpected value for agent_identifies (should be *PPO* or *A2C*)

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters

kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters

kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

load(*path: str | Path*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.7 primaite.agents.simple

Classes

DoNothingACLAgent	A do nothing ACL agent.
DoNothingNodeAgent	A do nothing Node agent.
DummyAgent	A Dummy Agent.
RandomAgent	A Random Agent.

3.7.3.7.1 primaite.agents.simple.DoNothingACLAgent

class primaite.agents.simple.**DoNothingACLAgent**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*)

Bases: *HardCodedAgentSessionABC*

A do nothing ACL agent.

A valid ACL action that has no effect; does nothing.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.7.2 primaite.agents.simple.DoNothingNodeAgent

class primaite.agents.simple.**DoNothingNodeAgent**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*)

Bases: *HardCodedAgentSessionABC*

A do nothing Node agent.

A valid Node action that has no effect; does nothing.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.7.3 primaite.agents.simple.DummyAgent

class primaite.agents.simple.**DummyAgent**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*)

Bases: *HardCodedAgentSessionABC*

A Dummy Agent.

All action spaces setup so dummy action is always 0 regardless of action type used.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.7.4 primaite.agents.simple.RandomAgent

class primaite.agents.simple.**RandomAgent**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*)

Bases: *HardCodedAgentSessionABC*

A Random Agent.

Get a completely random action from the action space.

Methods

close	Closes the agent.
evaluate	Evaluate the agent.
export	Export the agent to transportable file format.
learn	Train the agent.
load	Load an agent from file.
save	Save the agent.

Attributes

checkpoints_path	The Session checkpoints path.
evaluation_path	The evaluation outputs path.
learning_path	The learning outputs path.
timestamp_str	The session timestamp as a string.
uuid	The Agent Session UUID.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*) → None

Initialise a hardcoded agent session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.

property checkpoints_path: Path

The Session checkpoints path.

close() → None

Closes the agent.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property evaluation_path: Path

The evaluation outputs path.

export() → None

Export the agent to transportable file format.

learn(***kwargs: Any*) → None

Train the agent.

Parameters
kwargs – Any agent-specific key-word args to be passed.

property learning_path: Path

The learning outputs path.

classmethod load(*path: str | Path | None = None*) → None

Load an agent from file.

save() → None

Save the agent.

session_path

The Session path

property timestamp_str: str

The session timestamp as a string.

property uuid: str

The Agent Session UUID.

3.7.3.8 primaite.agents.utils

Functions

convert_to_new_obs	Convert original gym Box observation space to new multiDiscrete observation space.
convert_to_old_obs	Convert to old observation.
describe_obs_change	Build a string describing the difference between two observations.
get_new_action	Get new action (e.g.
get_node_of_ip	Get the node ID of an IP address.
is_valid_acl_action	Is the ACL action an actual valid action.
is_valid_acl_action_extra	Harsher version of valid acl actions, does not allow action.
is_valid_node_action	Is the node action an actual valid action.
transform_action_acl_enum	Convert acl action from readable str format, to enumerated format.
transform_action_acl_readable	Transform an ACL action to a more readable format.
transform_action_node_enum	Convert a node action from readable string format, to enumerated format.
transform_action_node_readable	Convert a node action from enumerated format to readable format.
transform_change_obs_readable	Transform list of transactions to readable list of each observation property.
transform_obs_readable	Transform observation to readable format.

3.7.3.8.1 primaite.agents.utils.convert_to_new_obs

primaite.agents.utils.**convert_to_new_obs**(*obs: ndarray*, *num_nodes: int = 10*) → ndarray

Convert original gym Box observation space to new multiDiscrete observation space.

Parameters

- **obs** (*np.ndarray*) – observation in the 'old' (NodeLinkTable) format
- **num_nodes** (*int, optional*) – number of nodes in the network, defaults to 10

Returns

reformatted observation

Return type

np.ndarray

3.7.3.8.2 primaite.agents.utils.convert_to_old_obs

primaite.agents.utils.**convert_to_old_obs**(*obs: ndarray*, *num_nodes: int = 10*, *num_links: int = 10*, *num_services: int = 1*) → ndarray

Convert to old observation.

Links filled with 0's as no information is included in new observation space.

example: obs = array([1, 1, 1, 1, 1, 1, 1, 1, 1, …, 1, 1, 1])

new_obs = array([[1, 1, 1, 1],

[2, 1, 1, 1], [3, 1, 1, 1], …

[20, 0, 0, 0]])

Parameters

- **obs** (*np.ndarray*) – observation in the 'new' (MultiDiscrete) format
- **num_nodes** (*int, optional*) – number of nodes in the network, defaults to 10
- **num_links** (*int, optional*) – number of links in the network, defaults to 10
- **num_services** (*int, optional*) – number of services on the network, defaults to 1

Returns
2-d BOX observation space, in the same format as NodeLinkTable

Return type
np.ndarray

3.7.3.8.3 primaite.agents.utils.describe_obs_change

primaite.agents.utils.**describe_obs_change**(*obs1: ndarray*, *obs2: ndarray*, *num_nodes: int = 10*, *num_links: int = 10*, *num_services: int = 1*) → str

Build a string describing the difference between two observations.

example: obs_1 = array([[1, 1, 1, 1, 3], [2, 1, 1, 1, 1]]) obs_2 = array([[1, 1, 1, 1, 1], [2, 1, 1, 1, 1]]) output = 'ID 1: SERVICE 2 set to GOOD'

Parameters

- **obs1** (*np.ndarray*) – First observation
- **obs2** (*np.ndarray*) – Second observation
- **num_nodes** (*int, optional*) – How many nodes are in the network laydown, defaults to 10
- **num_links** (*int, optional*) – How many links are in the network laydown, defaults to 10
- **num_services** (*int, optional*) – How many services are configured for this scenario, defaults to 1

Returns
A multi-line string with a human-readable description of the difference.

Return type
str

3.7.3.8.4 primaite.agents.utils.get_new_action

primaite.agents.utils.**get_new_action**(*old_action: ndarray*, *action_dict: Dict[int, List]*) → int

Get new action (e.g. 32) from old action e.g. [1,1,1,0].

Old_action can be either node or acl action type

Parameters

- **old_action** (*np.ndarray*) – Action expressed as a list of choices, eg. [1,1,1,0]
- **action_dict** (*Dict[int,List]*) – Dictionary for translating the multidiscrete actions into the list-based actions.

Returns
Action key correspoinding to the input *old_action*

Return type
int

3.7.3.8.5 primaite.agents.utils.get_node_of_ip

primaite.agents.utils.**get_node_of_ip**(*ip: str*, *node_dict: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*) → str

Get the node ID of an IP address.

node_dict: dictionary of nodes where key is ID, and value is the node (can be ontained from env.nodes)

Parameters

- **ip** (*str*) – The IP address of the node whose ID is required
- **node_dict** (*Dict[str,NodeUnion]*) – The environment's node registry dictionary

Returns
The key from the registry dict that corresponds to the node with the IP adress provided by *ip*

Return type
str

3.7.3.8.6 primaite.agents.utils.is_valid_acl_action

primaite.agents.utils.**is_valid_acl_action**(*action: List[int]*) → bool

Is the ACL action an actual valid action.

Only uses information about the action to determine if the action has an effect.

Does NOT consider:

- Trying to create identical rules
- Trying to create a rule which is a subset of another rule (caused by "ANY")

Parameters
action (*List[int]*) – Agent action, formatted as a list of ints, for more information check out *primaite.environment.primaite_env.Primaite*

Returns
Whether the action is valid

Return type
bool

3.7.3.8.7 primaite.agents.utils.is_valid_acl_action_extra

primaite.agents.utils.**is_valid_acl_action_extra**(*action: List[int]*) → bool

Harsher version of valid acl actions, does not allow action.

Parameters
action (*List[int]*) – Agent action, formatted as a list of ints, for more information check out *primaite.environment.primaite_env.Primaite*

Returns
Whether the action is valid

Return type
bool

3.7.3.8.8 primaite.agents.utils.is_valid_node_action

primaite.agents.utils.**is_valid_node_action**(*action: List[int]*) → bool

Is the node action an actual valid action.

Only uses information about the action to determine if the action has an effect

Does NOT consider: - Node ID not valid to perform an operation - e.g. selected node has no service so cannot patch - Node already being in that state (turning an ON node ON)

Parameters
action (*List[int]*) – Agent action, formatted as a list of ints, for more information check out *primaite.environment.primaite_env.Primaite*

Returns
Whether the action is valid

Return type
bool

3.7.3.8.9 primaite.agents.utils.transform_action_acl_enum

primaite.agents.utils.**transform_action_acl_enum**(*action: List[int | str]*) → ndarray

Convert acl action from readable str format, to enumerated format.

Parameters
action (*List[Union[int,str]]*) – ACL-based action expressed as a list of human-readable ints and strings

Returns
The same action but encoded to contain only integers.

Return type
np.ndarray

3.7.3.8.10 primaite.agents.utils.transform_action_acl_readable

primaite.agents.utils.**transform_action_acl_readable**(*action: List[int]*) → List[int | str]

Transform an ACL action to a more readable format.

example: [0, 1, 2, 5, 0, 1] -> ['NONE', 'ALLOW', 2, 5, 'ANY', 1]

Parameters
action (*List[int]*) – Agent action, formatted as a list of ints, for more information check out *primaite.environment.primaite_env.Primaite*

Returns
The same action list, but with the encodings translated back into meaningful labels

Return type
List[Union[int,str]]

3.7.3.8.11 primaite.agents.utils.transform_action_node_enum

primaite.agents.utils.**transform_action_node_enum**(*action: List[int | str]*) → List[int]

Convert a node action from readable string format, to enumerated format.

example: [1, 'SERVICE', 'PATCHING', 0] -> [1, 3, 1, 0] :param action: Action in 'readable' format :type action: List[Union[str,int]] :return: Action with verbs encoded as ints :rtype: List[int]

3.7.3.8.12 primaite.agents.utils.transform_action_node_readable

primaite.agents.utils.**transform_action_node_readable**(*action: List[int]*) → List[int | str]

Convert a node action from enumerated format to readable format.

example: [1, 3, 1, 0] -> [1, 'SERVICE', 'PATCHING', 0]

Parameters
action (*List[int]*) – Agent action, formatted as a list of ints, for more information check out *primaite.environment.primaite_env.Primaite*

Returns
The same action list, but with the encodings translated back into meaningful labels

Return type
List[Union[int,str]]

3.7.3.8.13 primaite.agents.utils.transform_change_obs_readable

primaite.agents.utils.**transform_change_obs_readable**(*obs: ndarray*) → List[List[int | str]]

Transform list of transactions to readable list of each observation property.

example: np.array([[1,2,1,3],[2,1,1,1]]) -> [[1, 2], ['OFF', 'ON'], ['GOOD', 'GOOD'], ['COMPROMISED', 'GOOD']]

Parameters
obs (*np.ndarray*) – Raw observation from the environment.

Returns
The same observation, but the encoded integer values are replaced with readable names.

Return type
List[List[Union[str, int]]]

3.7.3.8.14 primaite.agents.utils.transform_obs_readable

primaite.agents.utils.**transform_obs_readable**(*obs: ndarray*) → List[List[int | str]]

Transform observation to readable format.

example np.array([[1,2,1,3],[2,1,1,1]]) -> [[1, 'OFF', 'GOOD', 'COMPROMISED'], [2, 'ON', 'GOOD', 'GOOD']]

Parameters
obs (*np.ndarray*) – Raw observation from the environment.

Returns
The same observation, but the encoded integer values are replaced with readable names.

Return type
List[List[Union[str, int]]]

3.7.4 primaite.cli

Provides a CLI using Typer as an entry point.

Functions

build_dirs	Build the PrimAITE app directories.
clean_up	Cleans up left over files from previous version installations.
log_level	View or set the PrimAITE Log Level.
logs	Print the PrimAITE log file.
notebooks	Start Jupyter Lab in the users PrimAITE notebooks directory.
plotly_template	View or set the plotly template for Session plots.
reset_notebooks	Force a reset of the demo notebooks in the users notebooks directory.
session	Run a PrimAITE session.
setup	Perform the PrimAITE first-time setup.
version	Get the installed PrimAITE version number.

3.7.4.1 primaite.cli.build_dirs

primaite.cli.**build_dirs**() → None

Build the PrimAITE app directories.

3.7.4.2 primaite.cli.clean_up

primaite.cli.**clean_up**() → None

Cleans up left over files from previous version installations.

3.7.4.3 primaite.cli.log_level

primaite.cli.**log_level**(*level: LogLevel | None = None*) → None

View or set the PrimAITE Log Level.

To View, simply call: primaite log-level

To set, call: primaite log-level <desired log level>

For example, to set the to debug, call: primaite log-level DEBUG

3.7.4.4 primaite.cli.logs

primaite.cli.**logs**(*last_n: int*) → None

Print the PrimAITE log file.

Parameters

last_n – The number of lines to print. Default value is 10.

3.7.4.5 primaite.cli.notebooks

primaite.cli.**notebooks**() → None

Start Jupyter Lab in the users PrimAITE notebooks directory.

3.7.4.6 primaite.cli.plotly_template

primaite.cli.**plotly_template**(*template: PlotlyTemplate | None = None*) → None

View or set the plotly template for Session plots.

To View, simply call: primaite plotly-template

To set, call: primaite plotly-template <desired template>

For example, to set as plotly_dark, call: primaite plotly-template PLOTLY_DARK

3.7.4.7 primaite.cli.reset_notebooks

primaite.cli.**reset_notebooks**(*overwrite: bool = True*) → None

Force a reset of the demo notebooks in the users notebooks directory.

Parameters

overwrite – If True, will overwrite existing demo notebooks.

3.7.4.8 primaite.cli.session

primaite.cli.**session**(*tc: str | None = None*, *ldc: str | None = None*, *load: str | None = None*, *legacy_tc: bool = False*, *legacy_ldc: bool = False*) → None

Run a PrimAITE session.

tc: The training config filepath. Optional. If no value is passed then example default training config is used from: ~/primaite/2.0.0/config/example_config/training/training_config_main.yaml.

ldc: The lay down config file path. Optional. If no value is passed then example default lay down config is used from: ~/primaite/2.0.0/config/example_config/lay_down/lay_down_config_3_doc_very_basic.yaml.

load: The directory of a previous session. Optional. If no value is passed, then the session will use the default training config and laydown config. Inversely, if a training config and laydown config is passed while a session directory is passed, PrimAITE will load the session and ignore the training config and laydown config.

legacy_tc: If the training config file is a legacy file from PrimAITE < 2.0.

legacy_ldf: If the lay down config file is a legacy file from PrimAITE < 2.0.

3.7.4.9 primaite.cli.setup

primaite.cli.**setup**(*overwrite_existing: bool = True*) → None

Perform the PrimAITE first-time setup.

WARNING: All user-data will be lost.

3.7.4.10 primaite.cli.version

primaite.cli.**version**() → None

Get the installed PrimAITE version number.

3.7.5 primaite.common

Objects which are shared between many PrimAITE modules.

primaite.common.custom_typing	
primaite.common.enums	Enumerations for APE.
primaite.common.protocol	The protocol class.
primaite.common.service	The Service class.

3.7.5.1 primaite.common.custom_typing

Module attributes

NodeUnion	A Union of ActiveNode, PassiveNode, and ServiceNode.

3.7.5.1.1 primaite.common.custom_typing.NodeUnion

primaite.common.custom_typing.**NodeUnion**

A Union of ActiveNode, PassiveNode, and ServiceNode.

alias of Union[*ActiveNode*, *PassiveNode*, *ServiceNode*]

3.7.5.2 primaite.common.enums

Enumerations for APE.

Classes

ActionType	Action type enumeration.
AgentFramework	The agent algorithm framework/package.
AgentIdentifier	The Red Agent algo/class.
DeepLearningFramework	The deep learning framework.
FileSystemState	File System State.
HardCodedAgentView	The view the deterministic hard-coded agent has of the environment.
HardwareState	Node hardware state enumeration.
LinkStatus	Link traffic status.
NodeHardwareAction	Node hardware action.
NodePOLInitiator	Node Pattern of Life initiator enumeration.
NodePOLType	Node Pattern of Life type enumeration.
NodeSoftwareAction	Node software action.
NodeType	Node type enumeration.
ObservationType	Observation type enumeration.
Priority	Node priority enumeration.
Protocol	Service protocol enumeration.
RulePermissionType	Any firewall rule type.
SB3OutputVerboseLevel	The Stable Baselines3 learn/eval output verbosity level.
SessionType	The type of PrimAITE Session to be run.
SoftwareState	Software or Service state enumeration.

3.7.5.2.1 primaite.common.enums.ActionType

class primaite.common.enums.**ActionType**(*value*)

Bases: Enum

Action type enumeration.

Attributes

NODE	
ACL	
ANY	

3.7.5.2.2 primaite.common.enums.AgentFramework

class primaite.common.enums.**AgentFramework**(*value*)

Bases: Enum

The agent algorithm framework/package.

Attributes

CUSTOM	Custom Agent
SB3	Stable Baselines3

CUSTOM = 0

Custom Agent

SB3 = 1

Stable Baselines3

3.7.5.2.3 primaite.common.enums.AgentIdentifier

class primaite.common.enums.**AgentIdentifier**(*value*)

Bases: Enum

The Red Agent algo/class.

Attributes

A2C	Advantage Actor Critic
PPO	Proximal Policy Optimization
HARDCODED	The Hardcoded agents
DO_NOTHING	The DoNothing agents
RANDOM	The RandomAgent
DUMMY	The DummyAgent

A2C = 1

Advantage Actor Critic

DO_NOTHING = 4

The DoNothing agents

DUMMY = 6

The DummyAgent

HARDCODED = 3

The Hardcoded agents

PPO = 2

Proximal Policy Optimization

RANDOM = 5

The RandomAgent

3.7.5.2.4 primaite.common.enums.DeepLearningFramework

class primaite.common.enums.**DeepLearningFramework**(*value*)

Bases: Enum

The deep learning framework.

Attributes

TF	Tensorflow
TF2	Tensorflow 2.x
TORCH	PyTorch

TF = 'tf'

Tensorflow

TF2 = 'tf2'

Tensorflow 2.x

TORCH = 'torch'

PyTorch

3.7.5.2.5 primaite.common.enums.FileSystemState

class primaite.common.enums.**FileSystemState**(*value*)

Bases: Enum

File System State.

Attributes

GOOD	
CORRUPT	
DESTROYED	
REPAIRING	
RESTORING	

3.7.5.2.6 primaite.common.enums.HardCodedAgentView

class primaite.common.enums.**HardCodedAgentView**(*value*)

Bases: Enum

The view the deterministic hard-coded agent has of the environment.

Attributes

BASIC	The current observation space only
FULL	Full environment view with actions taken and reward feedback

BASIC = 1

The current observation space only

FULL = 2

Full environment view with actions taken and reward feedback

3.7.5.2.7 primaite.common.enums.HardwareState

class primaite.common.enums.**HardwareState**(*value*)

Bases: Enum

Node hardware state enumeration.

Attributes

NONE
ON
OFF
RESETTING
SHUTTING_DOWN
BOOTING

3.7.5.2.8 primaite.common.enums.LinkStatus

class primaite.common.enums.**LinkStatus**(*value*)

Bases: Enum

Link traffic status.

Attributes

NONE
LOW
MEDIUM
HIGH
OVERLOAD

3.7.5.2.9 primaite.common.enums.NodeHardwareAction

class primaite.common.enums.**NodeHardwareAction**(*value*)

Bases: Enum

Node hardware action.

Attributes

NONE
ON
OFF
RESET

3.7.5.2.10 primaite.common.enums.NodePOLInitiator

class primaite.common.enums.**NodePOLInitiator**(*value*)

Bases: Enum

Node Pattern of Life initiator enumeration.

Attributes

DIRECT
IER
SERVICE

3.7.5.2.11 primaite.common.enums.NodePOLType

class primaite.common.enums.**NodePOLType**(*value*)

Bases: Enum

Node Pattern of Life type enumeration.

Attributes

NONE
OPERATING
OS
SERVICE
FILE

3.7.5.2.12 primaite.common.enums.NodeSoftwareAction

class primaite.common.enums.**NodeSoftwareAction**(*value*)

Bases: Enum

Node software action.

Attributes

NONE
PATCHING

3.7.5.2.13 primaite.common.enums.NodeType

class primaite.common.enums.**NodeType**(*value*)

Bases: Enum

Node type enumeration.

Attributes

CCTV
SWITCH
COMPUTER
LINK
MONITOR
PRINTER
LOP
RTU
ACTUATOR
SERVER

3.7.5.2.14 primaite.common.enums.ObservationType

class primaite.common.enums.**ObservationType**(*value*)

Bases: Enum

Observation type enumeration.

Attributes

BOX
MULTIDISCRETE

3.7.5.2.15 primaite.common.enums.Priority

class primaite.common.enums.**Priority**(*value*)

Bases: Enum

Node priority enumeration.

Attributes

P1
P2
P3
P4
P5

3.7.5.2.16 primaite.common.enums.Protocol

class primaite.common.enums.**Protocol**(*value*)

Bases: Enum

Service protocol enumeration.

Attributes

LDAP
FTP
HTTPS
SMTP
RTP
IPP
TCP
NONE

3.7.5.2.17 primaite.common.enums.RulePermissionType

class primaite.common.enums.**RulePermissionType**(*value*)

Bases: Enum

Any firewall rule type.

Attributes

NONE
DENY
ALLOW

3.7.5.2.18 primaite.common.enums.SB3OutputVerboseLevel

class primaite.common.enums.**SB3OutputVerboseLevel**(*value*)

Bases: IntEnum

The Stable Baselines3 learn/eval output verbosity level.

Attributes

NONE
INFO
DEBUG

3.7.5.2.19 primaite.common.enums.SessionType

class primaite.common.enums.**SessionType**(*value*)

Bases: Enum

The type of PrimAITE Session to be run.

Attributes

TRAIN	Train an agent
EVAL	Evaluate an agent
TRAIN_EVAL	Train then evaluate an agent

EVAL = 2

Evaluate an agent

TRAIN = 1

Train an agent

TRAIN_EVAL = 3

Train then evaluate an agent

3.7.5.2.20 primaite.common.enums.SoftwareState

class primaite.common.enums.**SoftwareState**(*value*)

Bases: Enum

Software or Service state enumeration.

Attributes

NONE
GOOD
PATCHING
COMPROMISED
OVERWHELMED

3.7.5.3 primaite.common.protocol

The protocol class.

Classes

Protocol	Protocol class.

3.7.5.3.1 primaite.common.protocol.Protocol

class primaite.common.protocol.**Protocol**(*_name: str*)

Bases: object

Protocol class.

Methods

add_load	Adds load to the protocol.
clear_load	Clears the load on this protocol.
get_load	Gets the protocol load.
get_name	Gets the protocol name.

__init__(*_name: str*) → None

Initialise a protocol.

Parameters

_name (*str*) – The name of the protocol

add_load(*_load: int*) → None

Adds load to the protocol.

Args:

_load: The load to add

clear_load() → None

Clears the load on this protocol.

get_load() → int

Gets the protocol load.

Returns:
The protocol load (bps)

get_name() → str

Gets the protocol name.

Returns:
The protocol name

3.7.5.4 primaite.common.service

The Service class.

Classes

Service	Service class.

3.7.5.4.1 primaite.common.service.Service

class primaite.common.service.**Service**(*name: str*, *port: str*, *software_state:* SoftwareState)

Bases: object

Service class.

Methods

reduce_patching_count	Reduces the patching count for the service.

__init__(*name: str*, *port: str*, *software_state:* SoftwareState) → None

Initialise a service.

Parameters

- **name** – The service name.
- **port** – The service port.
- **software_state** – The service SoftwareState.

reduce_patching_count() → None

Reduces the patching count for the service.

3.7.6 primaite.config

Configuration parameters for running experiments.

primaite.config.lay_down_config	
primaite.config.training_config	

3.7.6.1 primaite.config.lay_down_config

Functions

convert_legacy_lay_down_config	Convert a legacy lay down config to the new format.
data_manipulation_config_path	The path to the example lay_down_config_5_data_manipulation.yaml file.
ddos_basic_one_config_path	The path to the example lay_down_config_1_DDOS_basic.yaml file.
ddos_basic_two_config_path	The path to the example lay_down_config_2_DDOS_basic.yaml file.
dos_very_basic_config_path	The path to the example lay_down_config_3_DOS_very_basic.yaml file.
load	Read in a lay down config yaml file.

3.7.6.1.1 primaite.config.lay_down_config.convert_legacy_lay_down_config

primaite.config.lay_down_config.**convert_legacy_lay_down_config**(*legacy_config: List[Dict[str, Any]]*) → List[Dict[str, Any]]

Convert a legacy lay down config to the new format.

Parameters
legacy_config – A legacy lay down config.

3.7.6.1.2 primaite.config.lay_down_config.data_manipulation_config_path

primaite.config.lay_down_config.**data_manipulation_config_path**() → Path

The path to the example lay_down_config_5_data_manipulation.yaml file.

Returns
The file path.

3.7.6.1.3 primaite.config.lay_down_config.ddos_basic_one_config_path

primaite.config.lay_down_config.**ddos_basic_one_config_path**() → Path

The path to the example lay_down_config_1_DDOS_basic.yaml file.

Returns
The file path.

3.7.6.1.4 primaite.config.lay_down_config.ddos_basic_two_config_path

primaite.config.lay_down_config.**ddos_basic_two_config_path**() → Path

The path to the example lay_down_config_2_DDOS_basic.yaml file.

Returns
The file path.

3.7.6.1.5 primaite.config.lay_down_config.dos_very_basic_config_path

primaite.config.lay_down_config.**dos_very_basic_config_path**() → Path

The path to the example lay_down_config_3_DOS_very_basic.yaml file.

Returns
The file path.

3.7.6.1.6 primaite.config.lay_down_config.load

primaite.config.lay_down_config.**load**(*file_path: str | Path*, *legacy_file: bool = False*) → Dict

Read in a lay down config yaml file.

Parameters

- **file_path** – The config file path.
- **legacy_file** – True if the config file is legacy format, otherwise False.

Returns
The lay down config as a dict.

Raises
ValueError – If the file_path does not exist.

3.7.6.2 primaite.config.training_config

Functions

convert_legacy_training_config_dict	Convert a legacy training config dict to the new format.
load	Read in a training config yaml file.
main_training_config_path	The path to the example training_config_main.yaml file.

3.7.6.2.1 primaite.config.training_config.convert_legacy_training_config_dict

primaite.config.training_config.**convert_legacy_training_config_dict**(*legacy_config_dict: Dict[str, Any]*, *agent_framework:* AgentFramework = *AgentFramework.SB3*, *agent_identifier:* AgentIdentifier = *AgentIdentifier.PPO*, *action_type:* ActionType = *ActionType.ANY*, *num_train_steps: int = 256*, *num_eval_steps: int = 256*, *num_train_episodes: int = 10*, *num_eval_episodes: int = 1*) → Dict[str, Any]

Convert a legacy training config dict to the new format.

Parameters

- **legacy_config_dict** – A legacy training config dict.
- **agent_framework** – The agent framework to use as legacy training configs don't have agent_framework values.
- **agent_identifier** – The red agent identifier to use as legacy training configs don't have agent_identifier values.
- **action_type** – The action space type to set as legacy training configs don't have action_type values.
- **num_train_steps** – The number of train steps to set as legacy training configs don't have num_train_steps values.
- **num_eval_steps** – The number of eval steps to set as legacy training configs don't have num_eval_steps values.
- **num_train_episodes** – The number of train episodes to set as legacy training configs don't have num_train_episodes values.
- **num_eval_episodes** – The number of eval episodes to set as legacy training configs don't have num_eval_episodes values.

Returns

The converted training config dict.

3.7.6.2.2 primaite.config.training_config.load

primaite.config.training_config.**load**(*file_path: str | Path*, *legacy_file: bool = False*) → *TrainingConfig*

Read in a training config yaml file.

Parameters

- **file_path** – The config file path.
- **legacy_file** – True if the config file is legacy format, otherwise False.

Returns
An instance of *TrainingConfig*.

Raises

- **ValueError** – If the file_path does not exist.
- **TypeError** – When the TrainingConfig object cannot be created using the values from the config file read from `file_path`.

3.7.6.2.3 primaite.config.training_config.main_training_config_path

primaite.config.training_config.**main_training_config_path**() → Path

The path to the example training_config_main.yaml file.

Returns
The file path.

Classes

TrainingConfig	The Training Config class.

3.7.6.2.4 primaite.config.training_config.TrainingConfig

class primaite.config.training_config.**TrainingConfig**(*agent_framework: ~primaite.common.enums.AgentFramework = AgentFramework.SB3*, *deep_learning_framework: ~primaite.common.enums.DeepLearningFramework = DeepLearningFramework.TF*, *agent_identifier: ~primaite.common.enums.AgentIdentifier = AgentIdentifier.PPO*, *hard_coded_agent_view: ~primaite.common.enums.HardCodedAgentView = HardCodedAgentView.FULL*, *random_red_agent: bool = False*, *action_type: ~primaite.common.enums.ActionType = ActionType.ANY*, *num_train_episodes: int = 10*, *num_train_steps: int = 256*, *num_eval_episodes: int = 1*, *num_eval_steps: int = 256*, *checkpoint_every_n_episodes: int = 5*, *observation_space: dict = <factory>*, *time_delay: int = 10*, *session_type: ~primaite.common.enums.SessionType = SessionType.TRAIN*, *load_agent: bool = False*, *agent_load_file: str | None = None*, *observation_space_high_value: int = 1000000000*, *sb3_output_verbose_level: ~primaite.common.enums.SB3OutputVerboseLevel = SB3OutputVerboseLevel.NONE*, *implicit_acl_rule: ~primaite.common.enums.RulePermissionType = RulePermissionType.DENY*, *max_number_acl_rules: int = 30*, *all_ok: float = 0*, *off_should_be_on: float = -0.001*, *off_should_be_resetting: float = -0.0005*, *on_should_be_off: float = -0.0002*, *on_should_be_resetting: float = -0.0005*, *resetting_should_be_on: float = -0.0005*, *resetting_should_be_off: float = -0.0002*, *resetting: float = -0.0003*, *good_should_be_patching: float = 0.0002*, *good_should_be_compromised: float = 0.0005*, *good_should_be_overwhelmed: float = 0.0005*, *patching_should_be_good: float = -0.0005*, *patching_should_be_compromised: float = 0.0002*, *patching_should_be_overwhelmed: float = 0.0002*, *patching: float = -0.0003*, *compromised_should_be_good: float = -0.002*, *compromised_should_be_patching: float = -0.002*, *compromised_should_be_overwhelmed: float = -0.002*, *compromised: float = -0.002*, *overwhelmed_should_be_good: float = -0.002*, *overwhelmed_should_be_patching: float = -0.002*, *overwhelmed_should_be_compromised: float = -0.002*, *overwhelmed: float = -0.002*, *good_should_be_repairing: float = 0.0002*, *good_should_be_restoring: float = 0.0002*, *good_should_be_corrupt: float = 0.0005*, *good_should_be_destroyed: float = 0.001*, *repairing_should_be_good: float = -0.0005*,

Bases: `object`

The Training Config class.

Methods

from_dict	Create an instance of TrainingConfig from a dict.
to_dict	Serialise the `TrainingConfig` as dict.

Attributes

action_type	The ActionType to use
agent_framework	The AgentFramework
agent_identifier	The AgentIdentifier
agent_load_file	File path and file name of agent if you're loading one in
all_ok	
checkpoint_every_n_episodes	The agent will save a checkpoint every n episodes
compromised	
compromised_should_be_good	
compromised_should_be_overwhelmed	
compromised_should_be_patching	
corrupt	
corrupt_should_be_destroyed	
corrupt_should_be_good	
corrupt_should_be_repairing	
corrupt_should_be_restoring	
deep_learning_framework	The DeepLearningFramework
destroyed	
destroyed_should_be_corrupt	
destroyed_should_be_good	
destroyed_should_be_repairing	
destroyed_should_be_restoring	
deterministic	If true, the training will be deterministic

continues on next page

Table 3 – continued from previous page

file_system_repairing_limit	The time take to repair the file system
file_system_restoring_limit	The time take to restore the file system
file_system_scanning_limit	The time taken to scan the file system
good_should_be_compromised	
good_should_be_corrupt	
good_should_be_destroyed	
good_should_be_overwhelmed	
good_should_be_patching	
good_should_be_repairing	
good_should_be_restoring	
green_ier_blocked	
hard_coded_agent_view	The view the deterministic hard-coded agent has of the environment
implicit_acl_rule	ALLOW or DENY implicit firewall rule to go at the end of list of ACL list.
load_agent	Determine whether to load an agent from file
max_number_acl_rules	Sets a limit for number of acl rules allowed in the list and environment.
node_booting_duration	The Time taken to turn on the node
node_reset_duration	The time taken to reset a node (hardware)
node_shutdown_duration	The time taken to turn off the node
num_eval_episodes	The number of episodes to train over during an evaluation session
num_eval_steps	The number of steps in an episode during an evaluation session
num_train_episodes	The number of episodes to train over during an training session
num_train_steps	The number of steps in an episode during an training session
observation_space_high_value	The high value for the observation space
off_should_be_on	
off_should_be_resetting	
on_should_be_off	
on_should_be_resetting	
os_patching_duration	The time taken to patch the OS
overwhelmed	
overwhelmed_should_be_compromised	

continues on next page

Table 3 – continued from previous page

overwhelmed_should_be_good	
overwhelmed_should_be_patching	
patching	
patching_should_be_compromised	
patching_should_be_good	
patching_should_be_overwhelmed	
random_red_agent	Creates Random Red Agent Attacks
red_ier_running	
repairing	
repairing_should_be_corrupt	
repairing_should_be_destroyed	
repairing_should_be_good	
repairing_should_be_restoring	
resetting	
resetting_should_be_off	
resetting_should_be_on	
restoring	
restoring_should_be_corrupt	
restoring_should_be_destroyed	
restoring_should_be_good	
restoring_should_be_repairing	
sb3_output_verbose_level	Stable Baselines3 learn/eval output verbosity level
scanning	
seed	The random number generator seed to be used while training the agent
service_patching_duration	The time taken to patch a service
session_type	The type of PrimAITE session to run
time_delay	The delay between steps (ms).
observation_space	The observation space config dict

__init__(*agent_framework: ~primaite.common.enums.AgentFramework = AgentFramework.SB3*, *deep_learning_framework: ~primaite.common.enums.DeepLearningFramework = DeepLearningFramework.TF*, *agent_identifier: ~primaite.common.enums.AgentIdentifier = AgentIdentifier.PPO*, *hard_coded_agent_view: ~primaite.common.enums.HardCodedAgentView = HardCodedAgentView.FULL*, *random_red_agent: bool = False*, *action_type: ~primaite.common.enums.ActionType = ActionType.ANY*, *num_train_episodes: int = 10*, *num_train_steps: int = 256*, *num_eval_episodes: int = 1*, *num_eval_steps: int = 256*, *checkpoint_every_n_episodes: int = 5*, *observation_space: dict = <factory>*, *time_delay: int = 10*, *session_type: ~primaite.common.enums.SessionType = SessionType.TRAIN*, *load_agent: bool = False*, *agent_load_file: str | None = None*, *observation_space_high_value: int = 1000000000*, *sb3_output_verbose_level: ~primaite.common.enums.SB3OutputVerboseLevel = SB3OutputVerboseLevel.NONE*, *implicit_acl_rule: ~primaite.common.enums.RulePermissionType = RulePermissionType.DENY*, *max_number_acl_rules: int = 30*, *all_ok: float = 0*, *off_should_be_on: float = -0.001*, *off_should_be_resetting: float = -0.0005*, *on_should_be_off: float = -0.0002*, *on_should_be_resetting: float = -0.0005*, *resetting_should_be_on: float = -0.0005*, *resetting_should_be_off: float = -0.0002*, *resetting: float = -0.0003*, *good_should_be_patching: float = 0.0002*, *good_should_be_compromised: float = 0.0005*, *good_should_be_overwhelmed: float = 0.0005*, *patching_should_be_good: float = -0.0005*, *patching_should_be_compromised: float = 0.0002*, *patching_should_be_overwhelmed: float = 0.0002*, *patching: float = -0.0003*, *compromised_should_be_good: float = -0.002*, *compromised_should_be_patching: float = -0.002*, *compromised_should_be_overwhelmed: float = -0.002*, *compromised: float = -0.002*, *overwhelmed_should_be_good: float = -0.002*, *overwhelmed_should_be_patching: float = -0.002*, *overwhelmed_should_be_compromised: float = -0.002*, *overwhelmed: float = -0.002*, *good_should_be_repairing: float = 0.0002*, *good_should_be_restoring: float = 0.0002*, *good_should_be_corrupt: float = 0.0005*, *good_should_be_destroyed: float = 0.001*, *repairing_should_be_good: float = -0.0005*, *repairing_should_be_restoring: float = 0.0002*, *repairing_should_be_corrupt: float = 0.0002*, *repairing_should_be_destroyed: float = 0.0*, *repairing: float = -0.0003*, *restoring_should_be_good: float = -0.001*, *restoring_should_be_repairing: float = -0.0002*, *restoring_should_be_corrupt: float = 0.0001*, *restoring_should_be_destroyed: float = 0.0002*, *restoring: float = -0.0006*, *corrupt_should_be_good: float = -0.001*, *corrupt_should_be_repairing: float = -0.001*, *corrupt_should_be_restoring: float = -0.001*, *corrupt_should_be_destroyed: float = 0.0002*, *corrupt: float = -0.001*, *destroyed_should_be_good: float = -0.002*, *destroyed_should_be_repairing: float = -0.002*, *destroyed_should_be_restoring: float = -0.002*, *destroyed_should_be_corrupt: float = -0.002*, *destroyed: float = -0.002*, *scanning: float = -0.0002*, *red_ier_running: float = -0.0005*, *green_ier_blocked: float = -0.001*, *os_patching_duration: int = 5*, *node_reset_duration: int = 5*, *node_booting_duration: int = 3*, *node_shutdown_duration: int = 2*, *service_patching_duration: int = 5*, *file_system_repairing_limit: int = 5*, *file_system_restoring_limit: int = 5*, *file_system_scanning_limit: int = 5*, *deterministic: bool = False*, *seed: int | None = None*) → None

action_type: *ActionType* = 2

The ActionType to use

agent_framework: *AgentFramework* = 1

The AgentFramework

agent_identifier: *AgentIdentifier* = 2

The AgentIdentifier

agent_load_file: str | None = None

File path and file name of agent if you're loading one in

checkpoint_every_n_episodes: int = 5

The agent will save a checkpoint every n episodes

deep_learning_framework: *DeepLearningFramework* = 'tf'

The DeepLearningFramework

deterministic: bool = False

If true, the training will be deterministic

file_system_repairing_limit: int = 5

The time take to repair the file system

file_system_restoring_limit: int = 5

The time take to restore the file system

file_system_scanning_limit: int = 5

The time taken to scan the file system

classmethod from_dict(*config_dict: Dict[str, Any]*) → *TrainingConfig*

Create an instance of TrainingConfig from a dict.

Parameters

config_dict – The training config dict.

Returns

The instance of TrainingConfig.

hard_coded_agent_view: *HardCodedAgentView* = 2

The view the deterministic hard-coded agent has of the environment

implicit_acl_rule: *RulePermissionType* = 1

ALLOW or DENY implicit firewall rule to go at the end of list of ACL list.

load_agent: bool = False

Determine whether to load an agent from file

max_number_acl_rules: int = 30

Sets a limit for number of acl rules allowed in the list and environment.

node_booting_duration: int = 3

The Time taken to turn on the node

node_reset_duration: int = 5

The time taken to reset a node (hardware)

node_shutdown_duration: int = 2

The time taken to turn off the node

num_eval_episodes: int = 1

The number of episodes to train over during an evaluation session

num_eval_steps: int = 256

The number of steps in an episode during an evaluation session

num_train_episodes: int = 10

The number of episodes to train over during an training session

num_train_steps: int = 256

The number of steps in an episode during an training session

observation_space: dict

The observation space config dict

observation_space_high_value: int = 1000000000

The high value for the observation space

os_patching_duration: int = 5

The time taken to patch the OS

random_red_agent: bool = False

Creates Random Red Agent Attacks

sb3_output_verbose_level: *SB3OutputVerboseLevel* = 0

Stable Baselines3 learn/eval output verbosity level

seed: int | None = None

The random number generator seed to be used while training the agent

service_patching_duration: int = 5

The time taken to patch a service

session_type: *SessionType* = 1

The type of PrimAITE session to run

time_delay: int = 10

The delay between steps (ms). Applies to generic agents only

to_dict(*json_serializable: bool = True*) → Dict

Serialise the `TrainingConfig` as dict.

Parameters

json_serializable – If True, Enums are converted to their string name.

Returns

The `TrainingConfig` as a dict.

3.7.7 primaite.data_viz

Utility to generate plots of sessions metrics after PrimAITE.

Classes

PlotlyTemplate	The built-in plotly templates.

3.7.7.1 primaite.data_viz.PlotlyTemplate

class primaite.data_viz.**PlotlyTemplate**(*value*)

Bases: `Enum`

The built-in plotly templates.

Attributes

PLOTLY
PLOTLY_WHITE
PLOTLY_DARK
GGPLOT2
SEABORN
SIMPLE_WHITE
NONE

primaite.data_viz.session_plots

3.7.7.2 primaite.data_viz.session_plots

Functions

get_plotly_config	Get the plotly config from primaite_config.yaml.
plot_av_reward_per_episode	Plot the average reward per episode from a csv session output.

3.7.7.2.1 primaite.data_viz.session_plots.get_plotly_config

primaite.data_viz.session_plots.**get_plotly_config**() → Dict

Get the plotly config from primaite_config.yaml.

3.7.7.2.2 primaite.data_viz.session_plots.plot_av_reward_per_episode

primaite.data_viz.session_plots.**plot_av_reward_per_episode**(*av_reward_per_episode_csv: str | Path*, *title: str | None = None*, *subtitle: str | None = None*) → Figure

Plot the average reward per episode from a csv session output.

Parameters

- **av_reward_per_episode_csv** – The average reward per episode csv file path.
- **title** – The plot title. This is optional.
- **subtitle** – The plot subtitle. This is optional.

Returns
The plot as an instance of `plotly.graph_objs._figure.Figure`.

3.7.8 primaite.environment

Gym/Gymnasium environment for RL agents consisting of a simulated computer network.

primaite.environment.observations	Module for handling configurable observation spaces in PrimAITE.
primaite.environment.primaite_env	Main environment module containing the PRIMmary AI Training Evironment (Primaite) class.
primaite.environment.reward	Implements reward function.

3.7.8.1 primaite.environment.observations

Module for handling configurable observation spaces in PrimAITE.

Classes

AbstractObservationComponent	Represents a part of the PrimAITE observation space.
AccessControlList	Flat list of all the Access Control Rules in the Access Control List.
LinkTrafficLevels	Flat list of traffic levels encoded into banded categories.
NodeLinkTable	Table with nodes and links as rows and hardware/software status as cols.
NodeStatuses	Flat list of nodes' hardware, OS, file system, and service states.
ObservationsHandler	Component-based observation space handler.

3.7.8.1.1 primaite.environment.observations.AbstractObservationComponent

class primaite.environment.observations.**AbstractObservationComponent**(*env:* Primaite)

Bases: ABC

Represents a part of the PrimAITE observation space.

Methods

generate_structure	Return a list of labels for the components of the flattened observation space.
update	Update the observation based on the current state of the environment.

abstract __init__(*env:* Primaite) → None

Initialise observation component.

Parameters
env (Primaite) – Primaite training environment.

abstract generate_structure() → List[str]

Return a list of labels for the components of the flattened observation space.

abstract update() → None

Update the observation based on the current state of the environment.

3.7.8.1.2 primaite.environment.observations.AccessControlList

class primaite.environment.observations.**AccessControlList**(*env:* Primaite)

Bases: *AbstractObservationComponent*

Flat list of all the Access Control Rules in the Access Control List.

The MultiDiscrete observation space can be though of as a one-dimensional vector of discrete states, represented by integers.

Each ACL Rule has 6 elements. It will have the following structure: .. code-block:

```
[
    acl_rule1 permission,
    acl_rule1 source_ip,
    acl_rule1 dest_ip,
    acl_rule1 protocol,
    acl_rule1 port,
    acl_rule1 position,
    acl_rule2 permission,
    acl_rule2 source_ip,
    acl_rule2 dest_ip,
    acl_rule2 protocol,
    acl_rule2 port,
    acl_rule2 position,
    ...
]
```

Terms (for ACL Observation Space):
[0, 1, 2] - Permission (0 = NA, 1 = DENY, 2 = ALLOW) [0, num nodes] - Source IP (0 = NA, 1 = any, then 2 -> x resolving to Node IDs) [0, num nodes] - Dest IP (0 = NA, 1 = any, then 2 -> x resolving to Node IDs) [0, num services] - Protocol (0 = NA, 1 = any, then 2 -> x resolving to protocol) [0, num ports] - Port (0 = NA, 1 = any, then 2 -> x resolving to port) [0, max acl rules - 1] - Position (0 = NA, 1 = first index, then 2 -> x index resolving to acl rule in acl list)

NOTE: NA is Non-Applicable - this means the ACL Rule in the list is a NoneType and NOT an ACLRule object.

Methods

generate_structure	Return a list of labels for the components of the flattened observation space.
update	Update the observation based on current environment state.

__init__(*env:* Primaite) → None

Initialise an AccessControlList observation component.

Parameters
env (Primaite) – The environment that forms the basis of the observations

generate_structure() → List[str]

Return a list of labels for the components of the flattened observation space.

update() → None

Update the observation based on current environment state.

The structure of the observation space is described in *AccessControlList*

3.7.8.1.3 primaite.environment.observations.LinkTrafficLevels

class primaite.environment.observations.**LinkTrafficLevels**(*env:* Primaite, *combine_service_traffic: bool = False*, *quantisation_levels: int = 5*)

Bases: *AbstractObservationComponent*

Flat list of traffic levels encoded into banded categories.

For each link, total traffic or traffic per service is encoded into a categorical value. For example, if quantisation_levels=5, the traffic levels represent these values:

- 0 = No traffic (0% of bandwidth)
- 1 = No traffic (0%-33% of bandwidth)
- 2 = No traffic (33%-66% of bandwidth)
- 3 = No traffic (66%-100% of bandwidth)
- 4 = No traffic (100% of bandwidth)

Note: The lowest category always corresponds to no traffic and the highest category to the link being at max capacity. Any amount of traffic between 0% and 100% (exclusive) is divided evenly into the remaining categories.

Methods

generate_structure	Return a list of labels for the components of the flattened observation space.
update	Update the observation based on current environment state.

__init__(*env:* Primaite, *combine_service_traffic: bool = False*, *quantisation_levels: int = 5*) → None

Initialise a LinkTrafficLevels observation component.

Parameters

- **env** (Primaite) – The environment that forms the basis of the observations
- **combine_service_traffic** (*bool, optional*) – Whether to consider total traffic on the link, or each protocol individually, defaults to False
- **quantisation_levels** (*int, optional*) – How many bands to consider when converting the traffic amount to a categorical value, defaults to 5

generate_structure() → List[str]

Return a list of labels for the components of the flattened observation space.

update() → None

Update the observation based on current environment state.

The structure of the observation space is described in *LinkTrafficLevels*

3.7.8.1.4 primaite.environment.observations.NodeLinkTable

class primaite.environment.observations.**NodeLinkTable**(*env:* Primaite)

Bases: *AbstractObservationComponent*

Table with nodes and links as rows and hardware/software status as cols.

This will create the observation space formatted as a table of integers. There is one row per node, followed by one row per link. The number of columns is 4 plus one per service. They are:

- node/link ID
- node hardware status / 0 for links
- node operating system status (if active/service) / 0 for links
- node file system status (active/service only) / 0 for links
- node service1 status / traffic load from that service for links
- node service2 status / traffic load from that service for links
- ...
- node serviceN status / traffic load from that service for links

For example if the environment has 5 nodes, 7 links, and 3 services, the observation space shape will be `(12, 7)`

Methods

generate_structure	Return a list of labels for the components of the flattened observation space.
update	Update the observation based on current environment state.

__init__(*env:* Primaite) → None

Initialise a NodeLinkTable observation space component.

Parameters
env (Primaite) – Training environment.

generate_structure() → List[str]

Return a list of labels for the components of the flattened observation space.

update() → None

Update the observation based on current environment state.

The structure of the observation space is described in *NodeLinkTable*

3.7.8.1.5 primaite.environment.observations.NodeStatuses

class primaite.environment.observations.**NodeStatuses**(*env:* Primaite)

Bases: *AbstractObservationComponent*

Flat list of nodes' hardware, OS, file system, and service states.

The MultiDiscrete observation space can be though of as a one-dimensional vector of discrete states, represented by integers. Each node has 3 elements plus 1 per service. It will have the following structure: .. code-block:

```
[
    node1 hardware state,
    node1 OS state,
    node1 file system state,
    node1 service1 state,
    node1 service2 state,
    node1 serviceN state (one for each service),
    node2 hardware state,
    node2 OS state,
    node2 file system state,
    node2 service1 state,
    node2 service2 state,
    node2 serviceN state (one for each service),
    ...
]
```

Methods

generate_structure	Return a list of labels for the components of the flattened observation space.
update	Update the observation based on current environment state.

__init__(*env:* Primaite) → None

Initialise a NodeStatuses observation component.

Parameters
env (Primaite) – Training environment.

generate_structure() → List[str]

Return a list of labels for the components of the flattened observation space.

update() → None

Update the observation based on current environment state.

The structure of the observation space is described in *NodeStatuses*

3.7.8.1.6 primaite.environment.observations.ObservationsHandler

class primaite.environment.observations.**ObservationsHandler**

Bases: object

Component-based observation space handler.

This allows users to configure observation spaces by mixing and matching components. Each component can also define further parameters to make them more flexible.

Methods

deregister	Remove a component from this handler.
describe_structure	Create a list of names for the features of the obs space.
from_config	Parse a config dictinary, return a new observation handler populated with new observation component objects.
register	Add a component for this handler to track.
update_obs	Fetch fresh information about the environment.
update_space	Rebuild the handler's composite observation space from its components.

Attributes

current_observation	Current observation, return the flattened version if flatten is True.
space	Observation space, return the flattened version if flatten is True.

__init__() → None

Initialise the observation handler.

property current_observation: ndarray | Tuple[ndarray]

Current observation, return the flattened version if flatten is True.

deregister(*obs_component:* AbstractObservationComponent) → None

Remove a component from this handler.

Parameters

obs_component (AbstractObservationComponent) – Which component to remove. It must exist within this object's `registered_obs_components` attribute.

describe_structure() → List[str]

Create a list of names for the features of the obs space.

The order of labels follows the flattened version of the space.

classmethod from_config(*env:* Primaite, *obs_space_config: dict*) → *ObservationsHandler*

Parse a config dictinary, return a new observation handler populated with new observation component objects.

The expected format for the config dictionary is:

```
config = {
    components: [
        {
            "name": "<COMPONENT1_NAME>"
        },
        {
            "name": "<COMPONENT2_NAME>"
            "options": {"opt1": val1, "opt2": val2}
        },
        {
            ...
        },
    ]
}
```

Returns

Observation handler

Return type

primaite.environment.observations.ObservationsHandler

register(*obs_component:* AbstractObservationComponent) → None

Add a component for this handler to track.

Parameters
obs_component (AbstractObservationComponent) – The component to add.

property space: Space

Observation space, return the flattened version if flatten is True.

update_obs() → None

Fetch fresh information about the environment.

update_space() → None

Rebuild the handler's composite observation space from its components.

3.7.8.2 primaite.environment.primaite_env

Main environment module containing the PRIMmary AI Training Evironment (Primaite) class.

Classes

Primaite	PRIMmary AI Training Evironment (Primaite) class.

3.7.8.2.1 primaite.environment.primaite_env.Primaite

class primaite.environment.primaite_env.**Primaite**(*training_config_path: str | Path*, *lay_down_config_path: str | Path*, *session_path: Path*, *timestamp_str: str*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*)

Bases: Env

PRIMmary AI Training Evironment (Primaite) class.

Methods

apply_actions_to_acl	Applies agent actions to the Access Control List [TODO].
apply_actions_to_nodes	Applies agent actions to the nodes.
apply_time_based_updates	Updates anything that needs to count down and then change state.
close	Override parent close and close writers.
create_acl_action_dict	Creates a dictionary mapping each possible discrete action to more readable multidiscrete action.
create_acl_rule	Creates an ACL rule from config data.
create_green_ier	Creates a green IER from config data.
create_green_pol	Creates a green PoL object from config data.
create_link	Creates a link from config data.
create_node	Creates a node from config data.
create_node_action_dict	Creates a dictionary mapping each possible discrete action to more readable multidiscrete action.

continues on next page

Table 4 – continued from previous page

create_node_and_acl_action_dict	Create a dictionary mapping each possible discrete action to a more readable mutlidiscrete action.
create_ports_list	Creates a list of ports from config data.
create_red_ier	Creates a red IER from config data.
create_red_pol	Creates a red PoL object from config data.
create_services_list	Creates a list of services (enum) from config data.
get_action_info	Extracts action_info.
get_observation_info	Extracts observation_info.
init_acl	Initialise the Access Control List.
init_observations	Create the environment's observation handler.
interpret_action_and_apply	Applies agent actions to the nodes and Access Control List.
load_lay_down_config	Loads config data in order to build the environment configuration.
output_link_status	Output the link status of all links to the console.
render	Renders the environment.
reset	AI Gym Reset function.
reset_environment	Resets environment.
reset_node	Resets the statuses of a node.
save_obs_config	Cache the config for the observation space.
seed	Sets the seed for this env's random number generator(s).
set_as_eval	Set the writers to write to eval directories.
step	AI Gym Step function.
update_environent_obs	Updates the observation space based on the node and link status.

Attributes

ACTION_SPACE_ACL_ACTION_VALUES	
ACTION_SPACE_ACL_PERMISSION_VALUES	
ACTION_SPACE_NODE_ACTION_VALUES	
ACTION_SPACE_NODE_PROPERTY_VALUES	
action_space	
actual_episode_count	Shifts the episode_count by -1 for RLlib learning session.
metadata	
observation_space	
reward_range	
spec	
unwrapped	Completely unwrap this env.
total_step_count	The total number of time steps completed.

__init__(*training_config_path: str | Path*, *lay_down_config_path: str | Path*, *session_path: Path*, *timestamp_str: str*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*) → None

The Primaite constructor.

Parameters

- **training_config_path** – The training config filepath.
- **lay_down_config_path** – The lay down config filepath.
- **session_path** – The directory path the session is writing to.
- **timestamp_str** – The session timestamp in the format: <yyyy-mm-dd>_<hh-mm- ss>.
- **legacy_training_config** – True if the training config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **legacy_lay_down_config** – True if the lay_down config file is a legacy file from PrimAITE < 2.0, otherwise False.

property actual_episode_count: int

Shifts the episode_count by -1 for RLlib learning session.

apply_actions_to_acl(*_action: int*) → None

Applies agent actions to the Access Control List [TO DO].

Args:
_action: The action space from the agent

apply_actions_to_nodes(*_action: int*) → None

Applies agent actions to the nodes.

Args:
_action: The action space from the agent

apply_time_based_updates() → None
Updates anything that needs to count down and then change state.

e.g. reset / patching status

close() → None
Override parent close and close writers.

create_acl_action_dict() → Dict[int, List[int]]
Creates a dictionary mapping each possible discrete action to more readable multidiscrete action.

create_acl_rule(*item: Dict*) → None
Creates an ACL rule from config data.

Args:
item: A config data item

create_green_ier(*item: Dict*) → None
Creates a green IER from config data.

Args:
item: A config data item

create_green_pol(*item: Dict*) → None
Creates a green PoL object from config data.

Args:
item: A config data item

create_link(*item: Dict*) → None
Creates a link from config data.

Args:
item: A config data item

create_node(*item: Dict*) → None
Creates a node from config data.

Args:
item: A config data item

create_node_action_dict() → Dict[int, List[int]]
Creates a dictionary mapping each possible discrete action to more readable multidiscrete action.

Note: Only actions that have the potential to change the state exist in the mapping (except for key 0)

example return: {0: [1, 0, 0, 0], 1: [1, 1, 1, 0], 2: [1, 1, 2, 0], 3: [1, 1, 3, 0], 4: [1, 2, 1, 0], 5: [1, 3, 1, 0], … }

create_node_and_acl_action_dict() → Dict[int, List[int]]
Create a dictionary mapping each possible discrete action to a more readable mutlidiscrete action.

The dictionary contains actions of both Node and ACL action types.

create_ports_list(*ports: Dict*) → None
Creates a list of ports from config data.

Args:
item: A config data item representing the ports

create_red_ier(*item: Dict*) → None

Creates a red IER from config data.

Args:
item: A config data item

create_red_pol(*item: Dict*) → None

Creates a red PoL object from config data.

Args:
item: A config data item

create_services_list(*services: Dict*) → None

Creates a list of services (enum) from config data.

Args:
item: A config data item representing the services

get_action_info(*action_info: Dict*) → None

Extracts action_info.

Args:
item: A config data item representing action info

get_observation_info(*observation_info: Dict*) → None

Extracts observation_info.

Parameters
observation_info (*str*) – Config item that defines which type of observation space to use

init_acl() → None

Initialise the Access Control List.

init_observations() → Tuple[Space, ndarray]

Create the environment's observation handler.

Returns
The observation space, initial observation (zeroed out array with the correct shape)

Return type
Tuple[spaces.Space, np.ndarray]

interpret_action_and_apply(*_action: int*) → None

Applies agent actions to the nodes and Access Control List.

Args:
_action: The action space from the agent

load_lay_down_config() → None

Loads config data in order to build the environment configuration.

output_link_status() → None

Output the link status of all links to the console.

abstract render(*mode='human'*)

Renders the environment.

The set of supported modes varies per environment. (And some environments do not support rendering at all.) By convention, if mode is:

- human: render to the current display or terminal and return nothing. Usually for human consumption.

- rgb_array: Return an numpy.ndarray with shape (x, y, 3), representing RGB values for an x-by-y pixel image, suitable for turning into a video.
- ansi: Return a string (str) or StringIO.StringIO containing a terminal-style text representation. The text can include newlines and ANSI escape sequences (e.g. for colors).

Note:

Make sure that your class's metadata 'render.modes' key includes
the list of supported modes. It's recommended to call super() in implementations to use the functionality of this method.

Args:
mode (str): the mode to render with

Example:

class MyEnv(Env):
metadata = {'render.modes': ['human', 'rgb_array']}

def render(self, mode='human'):

if mode == 'rgb_array':
return np.array(...) # return RGB frame suitable for video

elif mode == 'human':
... # pop up a window and render

else:
super(MyEnv, self).render(mode=mode) # just raise an exception

`reset()` → ndarray

AI Gym Reset function.

Returns:
Environment observation space (reset)

`reset_environment()` → None

Resets environment.

Uses config data config data in order to build the environment configuration.

`reset_node(item: Dict)` → None

Resets the statuses of a node.

Args:
item: A config data item

`save_obs_config(obs_config: dict)` → None

Cache the config for the observation space.

This is necessary as the observation space can't be built while reading the config, it must be done after all the nodes, links, and services have been initialised.

Parameters
obs_config – Parsed config relating to the observation space. The format is described in

primaite.environment.observations.ObservationsHandler.from_config() :type obs_config: dict

seed(*seed=None*)

Sets the seed for this env's random number generator(s).

Note:
Some environments use multiple pseudorandom number generators. We want to capture all such seeds used in order to ensure that there aren't accidental correlations between multiple generators.

Returns:

list<bigint>: Returns the list of seeds used in this env's random
number generators. The first value in the list should be the "main" seed, or the value which a reproducer should pass to 'seed'. Often, the main seed equals the provided 'seed', but this won't be true if seed=None, for example.

set_as_eval() → None

Set the writers to write to eval directories.

step(*action: int*) → Tuple[ndarray, float, bool, Dict]

AI Gym Step function.

Args:
action: Action space from agent

Returns:
env_obs: Observation space reward: Reward value for this step done: Indicates episode is complete if True step_info: Additional information relating to this step

total_step_count: int

The total number of time steps completed.

property unwrapped

Completely unwrap this env.

Returns:
gym.Env: The base non-wrapped gym.Env instance

update_environent_obs() → None

Updates the observation space based on the node and link status.

3.7.8.3 primaite.environment.reward

Implements reward function.

Functions

calculate_reward_function	Compares the states of the initial and final nodes/links to get a reward.
score_node_file_system	Calculates score relating to the file system state of a node.
score_node_operating_state	Calculates score relating to the hardware state of a node.
score_node_os_state	Calculates score relating to the Software State of a node.
score_node_service_state	Calculates score relating to the service state(s) of a node.

3.7.8.3.1 primaite.environment.reward.calculate_reward_function

primaite.environment.reward.**calculate_reward_function**(*initial_nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *final_nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *reference_nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *green_iers: Dict[str,* IER*]*, *green_iers_reference: Dict[str,* IER*]*, *red_iers: Dict[str,* IER*]*, *step_count: int*, *config_values:* TrainingConfig) → float

Compares the states of the initial and final nodes/links to get a reward.

Args:

initial_nodes: The nodes before red and blue agents take effect final_nodes: The nodes after red and blue agents take effect reference_nodes: The nodes if there had been no red or blue effect green_iers: The green IERs (should be running) red_iers: Should be stopeed (ideally) by the blue agent step_count: current step config_values: Config values

3.7.8.3.2 primaite.environment.reward.score_node_file_system

primaite.environment.reward.**score_node_file_system**(*final_node:* ActiveNode | ServiceNode, *initial_node:* ActiveNode | ServiceNode, *reference_node:* ActiveNode | ServiceNode, *config_values:* TrainingConfig) → float

Calculates score relating to the file system state of a node.

Args:

final_node: The node after red and blue agents take effect initial_node: The node before red and blue agents take effect reference_node: The node if there had been no red or blue effect

3.7.8.3.3 primaite.environment.reward.score_node_operating_state

primaite.environment.reward.**score_node_operating_state**(*final_node:* ActiveNode | PassiveNode | ServiceNode, *initial_node:* ActiveNode | PassiveNode | ServiceNode, *reference_node:* ActiveNode | PassiveNode | ServiceNode, *config_values:* TrainingConfig) → float

Calculates score relating to the hardware state of a node.

Args:

final_node: The node after red and blue agents take effect initial_node: The node before red and blue agents take effect reference_node: The node if there had been no red or blue effect config_values: Config values

3.7.8.3.4 primaite.environment.reward.score_node_os_state

primaite.environment.reward.**score_node_os_state**(*final_node:* ActiveNode | ServiceNode, *initial_node:* ActiveNode | ServiceNode, *reference_node:* ActiveNode | ServiceNode, *config_values:* TrainingConfig) → float

Calculates score relating to the Software State of a node.

Args:
final_node: The node after red and blue agents take effect initial_node: The node before red and blue agents take effect reference_node: The node if there had been no red or blue effect config_values: Config values

3.7.8.3.5 primaite.environment.reward.score_node_service_state

primaite.environment.reward.**score_node_service_state**(*final_node:* ServiceNode, *initial_node:* ServiceNode, *reference_node:* ServiceNode, *config_values:* TrainingConfig) → float

Calculates score relating to the service state(s) of a node.

Args:
final_node: The node after red and blue agents take effect initial_node: The node before red and blue agents take effect reference_node: The node if there had been no red or blue effect config_values: Config values

3.7.9 primaite.exceptions

Exceptions

NetworkError	Raised when an error occurs at the network level.
PrimaiteError	The root PrimAITe Error.
RLlibAgentError	Raised when there is a generic error with a RLlib agent that is specific to PRimAITE.

3.7.9.1 primaite.exceptions.NetworkError

exception primaite.exceptions.**NetworkError**

Raised when an error occurs at the network level.

3.7.9.2 primaite.exceptions.PrimaiteError

exception primaite.exceptions.**PrimaiteError**

The root PrimAITe Error.

3.7.9.3 primaite.exceptions.RLlibAgentError

exception primaite.exceptions.**RLlibAgentError**

Raised when there is a generic error with a RLlib agent that is specific to PRimAITE.

3.7.10 primaite.links

Network connections between nodes in the simulation.

primaite.links.link	The link class.

3.7.10.1 primaite.links.link

The link class.

Classes

Link	Link class.

3.7.10.1.1 primaite.links.link.Link

class primaite.links.link.**Link**(*_id: str*, *_bandwidth: int*, *_source_node_name: str*, *_dest_node_name: str*, *_services: str*)

Bases: `object`

Link class.

Methods

add_protocol	Adds a new protocol to the list of protocols on this link.
add_protocol_load	Adds a loading to a protocol on this link.
clear_traffic	Clears all traffic on this link.
get_bandwidth	Gets bandwidth of link.
get_current_load	Gets current total load on this link.
get_dest_node_name	Gets destination node name.
get_id	Gets link ID.
get_protocol_list	Gets list of protocols on this link.
get_source_node_name	Gets source node name.

__init__(*_id: str*, *_bandwidth: int*, *_source_node_name: str*, *_dest_node_name: str*, *_services: str*) → None

Initialise a Link within the simulated network.

Parameters

- **_id** – The IER id

- **_bandwidth** – The bandwidth of the link (bps)
- **_source_node_name** – The name of the source node
- **_dest_node_name** – The name of the destination node
- **_protocols** – The protocols to add to the link

add_protocol(*_protocol: str*) → None

Adds a new protocol to the list of protocols on this link.

Args:
_protocol: The protocol to be added (enum)

add_protocol_load(*_protocol: str*, *_load: int*) → None

Adds a loading to a protocol on this link.

Args:
_protocol: The protocol to load _load: The amount to load (bps)

clear_traffic() → None

Clears all traffic on this link.

get_bandwidth() → int

Gets bandwidth of link.

Returns:
Link bandwidth (bps)

get_current_load() → int

Gets current total load on this link.

Returns:
Total load on this link (bps)

get_dest_node_name() → str

Gets destination node name.

Returns:
Destination node name

get_id() → str

Gets link ID.

Returns:
Link ID

get_protocol_list() → List[*Protocol*]

Gets list of protocols on this link.

Returns:
List of protocols on this link

get_source_node_name() → str

Gets source node name.

Returns:
Source node name

3.7.11 primaite.main

The main PrimAITE session runner module.

Functions

run	Run the PrimAITE Session.

3.7.11.1 primaite.main.run

primaite.main.**run**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*) → None

Run the PrimAITE Session.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.
- **session_path** – directory path of the session to load
- **legacy_training_config** – True if the training config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **legacy_lay_down_config** – True if the lay_down config file is a legacy file from PrimAITE < 2.0, otherwise False.

3.7.12 primaite.nodes

Nodes represent network hosts in the simulation.

primaite.nodes.active_node	An Active Node (i.e.
primaite.nodes.node	The base Node class.
primaite.nodes.node_state_instruction_green	Defines node behaviour for Green PoL.
primaite.nodes.node_state_instruction_red	Defines node behaviour for Green PoL.
primaite.nodes.passive_node	The Passive Node class (i.e.
primaite.nodes.service_node	A Service Node (i.e.

3.7.12.1 primaite.nodes.active_node

An Active Node (i.e. not an actuator).

Classes

ActiveNode	Active Node class.

3.7.12.1.1 primaite.nodes.active_node.ActiveNode

class primaite.nodes.active_node.**ActiveNode**(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *ip_address: str*, *software_state:* SoftwareState, *file_system_state:* FileSystemState, *config_values:* TrainingConfig)

Bases: *Node*

Active Node class.

Methods

reset	Sets the node state to Resetting and starts the reset count.
set_file_system_state	Sets the file system state (actual and observed).
set_file_system_state_if_not_compromised	Sets the file system state (actual and observed) if not in a compromised state.
set_software_state_if_not_compromised	Sets Software State if the node is not compromised.
start_file_system_scan	Starts a file system scan.
turn_off	Sets the node state to OFF.
turn_on	Sets the node state to ON.
update_booting_status	Updates the booting software and file state to GOOD.
update_file_system_state	Updates file system status based on scanning/restore/repair cycle.
update_os_patching_status	Updates operating system status based on patching cycle.
update_resetting_status	Updates the reset count & makes software and file state to GOOD.
update_shutdown_status	Updates the shutdown count.

Attributes

software_state	Get the software_state.

__init__(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *ip_address: str*, *software_state:* SoftwareState, *file_system_state:* FileSystemState, *config_values:* TrainingConfig) → None

Initialise an active node.

Parameters

- **node_id** – The node ID
- **name** – The node name
- **node_type** – The node type (enum)
- **priority** – The node priority (enum)
- **hardware_state** – The node Hardware State
- **ip_address** – The node IP address
- **software_state** – The node Software State
- **file_system_state** – The node file system state
- **config_values** – The config values

reset() → None

Sets the node state to Resetting and starts the reset count.

set_file_system_state(*file_system_state:* FileSystemState) → None

Sets the file system state (actual and observed).

Args:
file_system_state: File system state

set_file_system_state_if_not_compromised(*file_system_state:* FileSystemState) → None

Sets the file system state (actual and observed) if not in a compromised state.

Use for green PoL to prevent it overturning a compromised state

Args:
file_system_state: File system state

set_software_state_if_not_compromised(*software_state:* SoftwareState) → None

Sets Software State if the node is not compromised.

Args:
software_state: Software State

property software_state: *SoftwareState*

Get the software_state.

Returns
The software_state.

start_file_system_scan() → None

Starts a file system scan.

turn_off() → None

Sets the node state to OFF.

turn_on() → None

Sets the node state to ON.

update_booting_status() → None

Updates the booting software and file state to GOOD.

update_file_system_state() → None

Updates file system status based on scanning/restore/repair cycle.

update_os_patching_status() → None

Updates operating system status based on patching cycle.

update_resetting_status() → None

Updates the reset count & makes software and file state to GOOD.

update_shutdown_status() → None

Updates the shutdown count.

3.7.12.2 primaite.nodes.node

The base Node class.

Classes

Node	Node class.

3.7.12.2.1 primaite.nodes.node.Node

class primaite.nodes.node.**Node**(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *config_values:* TrainingConfig)

Bases: object

Node class.

Methods

reset	Sets the node state to Resetting and starts the reset count.
turn_off	Sets the node state to OFF.
turn_on	Sets the node state to ON.
update_booting_status	Updates the booting count.
update_resetting_status	Updates the resetting count.
update_shutdown_status	Updates the shutdown count.

__init__(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *config_values:* TrainingConfig) → None

Initialise a node.

Parameters

- **node_id** – The node id.
- **name** – The name of the node.
- **node_type** – The type of the node.
- **priority** – The priority of the node.
- **hardware_state** – The state of the node.
- **config_values** – Config values.

reset() → None
Sets the node state to Resetting and starts the reset count.

turn_off() → None
Sets the node state to OFF.

turn_on() → None
Sets the node state to ON.

update_booting_status() → None
Updates the booting count.

update_resetting_status() → None
Updates the resetting count.

update_shutdown_status() → None
Updates the shutdown count.

3.7.12.3 primaite.nodes.node_state_instruction_green

Defines node behaviour for Green PoL.

Classes

NodeStateInstructionGreen	The Node State Instruction class.

3.7.12.3.1 primaite.nodes.node_state_instruction_green.NodeStateInstructionGreen

class primaite.nodes.node_state_instruction_green.**NodeStateInstructionGreen**(*_id: str*, *_start_step: int*, *_end_step: int*, *_node_id: str*, *_node_pol_type:* NodePOLType, *_service_name: str*, *_state:* HardwareState | SoftwareState | FileSystemState)

Bases: object

The Node State Instruction class.

Methods

get_end_step	Gets the end step.
get_node_id	Gets the node ID.
get_node_pol_type	Gets the node pattern of life type (enum).
get_service_name	Gets the service name.
get_start_step	Gets the start step.
get_state	Gets the state (node or service).

__init__(*_id: str*, *_start_step: int*, *_end_step: int*, *_node_id: str*, *_node_pol_type:* NodePOLType, *_service_name: str*, *_state:* HardwareState | SoftwareState | FileSystemState) → None

Initialise the Node State Instruction.

Parameters

- **_id** – The node state instruction id
- **_start_step** – The start step of the instruction
- **_end_step** – The end step of the instruction
- **_node_id** – The id of the associated node
- **_node_pol_type** – The pattern of life type
- **_service_name** – The service name
- **_state** – The state (node or service)

get_end_step() → int

Gets the end step.

Returns:
The end step

get_node_id() → str

Gets the node ID.

Returns:
The node ID

get_node_pol_type() → *NodePOLType*

Gets the node pattern of life type (enum).

Returns:
The node pattern of life type (enum)

get_service_name() → str

Gets the service name.

Returns:
The service name

get_start_step() → int

Gets the start step.

Returns:
The start step

get_state() → *HardwareState* | *SoftwareState* | *FileSystemState*

Gets the state (node or service).

Returns:
The state (node or service)

3.7.12.4 primaite.nodes.node_state_instruction_red

Defines node behaviour for Green PoL.

Classes

NodeStateInstructionRed	The Node State Instruction class.

3.7.12.4.1 primaite.nodes.node_state_instruction_red.NodeStateInstructionRed

class primaite.nodes.node_state_instruction_red.**NodeStateInstructionRed**(*_id: str*, *_start_step: int*, *_end_step: int*, *_target_node_id: str*, *_pol_initiator: NodePOLInitiator*, *_pol_type: NodePOLType*, *pol_protocol: str*, *_pol_state: HardwareState | SoftwareState | FileSystemState*, *_pol_source_node_id: str*, *_pol_source_node_service: str*, *_pol_source_node_service_state: str*)

Bases: object

The Node State Instruction class.

Methods

get_end_step	Gets the end step.
get_initiator	Gets the initiator.
get_pol_type	Gets the node pattern of life type (enum).
get_service_name	Gets the service name.
get_source_node_id	Gets the source node id (used for initiator type SERVICE).
get_source_node_service	Gets the source node service (used for initiator type SERVICE).
get_source_node_service_state	Gets the source node service state (used for initiator type SERVICE).
get_start_step	Gets the start step.
get_state	Gets the state (node or service).
get_target_node_id	Gets the node ID.

__init__(*_id: str*, *_start_step: int*, *_end_step: int*, *_target_node_id: str*, *_pol_initiator:* NodePOLInitiator, *_pol_type:* NodePOLType, *pol_protocol: str*, *_pol_state:* HardwareState | SoftwareState | FileSystemState, *_pol_source_node_id: str*, *_pol_source_node_service: str*, *_pol_source_node_service_state: str*) → None

Initialise the Node State Instruction for the red agent.

Parameters

- **_id** – The node state instruction id
- **_start_step** – The start step of the instruction
- **_end_step** – The end step of the instruction
- **_target_node_id** – The id of the associated node
- **-pol_initiator** – The way the PoL is applied (DIRECT, IER or SERVICE)
- **_pol_type** – The pattern of life type
- **pol_protocol** – The pattern of life protocol/service affected
- **_pol_state** – The state (node or service)
- **_pol_source_node_id** – The source node Id (used for initiator type SERVICE)
- **_pol_source_node_service** – The source node service (used for initiator type SERVICE)
- **_pol_source_node_service_state** – The source node service state (used for initiator type SERVICE)

get_end_step() → int

Gets the end step.

Returns:
The end step

get_initiator() → *NodePOLInitiator*

Gets the initiator.

Returns:
The initiator

`get_pol_type()` → *NodePOLType*

Gets the node pattern of life type (enum).

Returns:
The node pattern of life type (enum)

`get_service_name()` → str

Gets the service name.

Returns:
The service name

`get_source_node_id()` → str

Gets the source node id (used for initiator type SERVICE).

Returns:
The source node id

`get_source_node_service()` → str

Gets the source node service (used for initiator type SERVICE).

Returns:
The source node service

`get_source_node_service_state()` → str

Gets the source node service state (used for initiator type SERVICE).

Returns:
The source node service state

`get_start_step()` → int

Gets the start step.

Returns:
The start step

`get_state()` → *HardwareState* | *SoftwareState* | *FileSystemState*

Gets the state (node or service).

Returns:
The state (node or service)

`get_target_node_id()` → str

Gets the node ID.

Returns:
The node ID

3.7.12.5 primaite.nodes.passive_node

The Passive Node class (i.e. an actuator).

Classes

PassiveNode	The Passive Node class.

3.7.12.5.1 primaite.nodes.passive_node.PassiveNode

class primaite.nodes.passive_node.**PassiveNode**(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *config_values:* TrainingConfig)

Bases: *Node*

The Passive Node class.

Methods

reset	Sets the node state to Resetting and starts the reset count.
turn_off	Sets the node state to OFF.
turn_on	Sets the node state to ON.
update_booting_status	Updates the booting count.
update_resetting_status	Updates the resetting count.
update_shutdown_status	Updates the shutdown count.

Attributes

ip_address	Gets the node IP address as an empty string.

__init__(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *config_values:* TrainingConfig) → None

Initialise a passive node.

Parameters

- **node_id** – The node id.
- **name** – The name of the node.
- **node_type** – The type of the node.
- **priority** – The priority of the node.
- **hardware_state** – The state of the node.
- **config_values** – Config values.

property ip_address: str

Gets the node IP address as an empty string.

No concept of IP address for passive nodes for now.

Returns

The node IP address.

reset() → None
Sets the node state to Resetting and starts the reset count.

turn_off() → None
Sets the node state to OFF.

turn_on() → None
Sets the node state to ON.

update_booting_status() → None
Updates the booting count.

update_resetting_status() → None
Updates the resetting count.

update_shutdown_status() → None
Updates the shutdown count.

3.7.12.6 primaite.nodes.service_node

A Service Node (i.e. not an actuator).

Classes

ServiceNode	ServiceNode class.

3.7.12.6.1 primaite.nodes.service_node.ServiceNode

class primaite.nodes.service_node.**ServiceNode**(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *ip_address: str*, *software_state:* SoftwareState, *file_system_state:* FileSystemState, *config_values:* TrainingConfig)

Bases: *ActiveNode*

ServiceNode class.

Methods

add_service	Adds a service to the node.
get_service_state	Gets the state of a service.
has_service	Indicates whether a service is on a node.
reset	Sets the node state to Resetting and starts the reset count.
service_is_overwhelmed	Indicates whether a service is in an overwhelmed state on the node.
service_running	Indicates whether a service is in a running state on the node.
set_file_system_state	Sets the file system state (actual and observed).
set_file_system_state_if_not_compromised	Sets the file system state (actual and observed) if not in a compromised state.
set_service_state	Sets the software_state of a service (protocol) on the node.
set_service_state_if_not_compromised	Sets the software_state of a service (protocol) on the node.
set_software_state_if_not_compromised	Sets Software State if the node is not compromised.
start_file_system_scan	Starts a file system scan.
turn_off	Sets the node state to OFF.
turn_on	Sets the node state to ON.
update_booting_status	Update booting counter and set software to good if it reached 0.
update_file_system_state	Updates file system status based on scanning/restore/repair cycle.
update_os_patching_status	Updates operating system status based on patching cycle.
update_resetting_status	Update resetting counter and set software state if it reached 0.
update_services_patching_status	Updates the patching counter for any service that are patching.
update_shutdown_status	Updates the shutdown count.

Attributes

software_state	Get the software_state.

__init__(*node_id: str*, *name: str*, *node_type:* NodeType, *priority:* Priority, *hardware_state:* HardwareState, *ip_address: str*, *software_state:* SoftwareState, *file_system_state:* FileSystemState, *config_values:* TrainingConfig) → None

Initialise a Service Node.

Parameters

- **node_id** – The node ID
- **name** – The node name
- **node_type** – The node type (enum)
- **priority** – The node priority (enum)

- **hardware_state** – The node Hardware State
- **ip_address** – The node IP address
- **software_state** – The node Software State
- **file_system_state** – The node file system state
- **config_values** – The config values

add_service(*service:* Service) → None

Adds a service to the node.

Parameters
service – The service to add

get_service_state(*protocol_name: str*) → *SoftwareState*

Gets the state of a service.

Returns
The software_state of the service.

has_service(*protocol_name: str*) → bool

Indicates whether a service is on a node.

Parameters
protocol_name – The service (protocol)e.

Returns
True if service (protocol) is on the node, otherwise False.

reset() → None

Sets the node state to Resetting and starts the reset count.

service_is_overwhelmed(*protocol_name: str*) → bool

Indicates whether a service is in an overwhelmed state on the node.

Parameters
protocol_name – The service (protocol)

Returns
True if service (protocol) is in an overwhelmed state on the node, otherwise False.

service_running(*protocol_name: str*) → bool

Indicates whether a service is in a running state on the node.

Parameters
protocol_name – The service (protocol)

Returns
True if service (protocol) is in a running state on the node, otherwise False.

set_file_system_state(*file_system_state:* FileSystemState) → None

Sets the file system state (actual and observed).

Args:
file_system_state: File system state

set_file_system_state_if_not_compromised(*file_system_state:* FileSystemState) → None

Sets the file system state (actual and observed) if not in a compromised state.

Use for green PoL to prevent it overturning a compromised state

Args:
file_system_state: File system state

set_service_state(*protocol_name: str*, *software_state:* SoftwareState) → None
Sets the software_state of a service (protocol) on the node.

Parameters

- **protocol_name** – The service (protocol).
- **software_state** – The software_state.

set_service_state_if_not_compromised(*protocol_name: str*, *software_state:* SoftwareState) → None
Sets the software_state of a service (protocol) on the node.

Done if the software_state is not "compromised".

Parameters

- **protocol_name** – The service (protocol).
- **software_state** – The software_state.

set_software_state_if_not_compromised(*software_state:* SoftwareState) → None
Sets Software State if the node is not compromised.

Args:
software_state: Software State

property software_state: *SoftwareState*
Get the software_state.

Returns
The software_state.

start_file_system_scan() → None
Starts a file system scan.

turn_off() → None
Sets the node state to OFF.

turn_on() → None
Sets the node state to ON.

update_booting_status() → None
Update booting counter and set software to good if it reached 0.

update_file_system_state() → None
Updates file system status based on scanning/restore/repair cycle.

update_os_patching_status() → None
Updates operating system status based on patching cycle.

update_resetting_status() → None
Update resetting counter and set software state if it reached 0.

update_services_patching_status() → None
Updates the patching counter for any service that are patching.

update_shutdown_status() → None
Updates the shutdown count.

3.7.13 primaite.notebooks

Contains default jupyter notebooks which demonstrate PrimAITE functionality.

Functions

start_jupyter_session	Starts a new Jupyter notebook session in the app notebooks directory.

3.7.13.1 primaite.notebooks.start_jupyter_session

primaite.notebooks.**start_jupyter_session**() → None

Starts a new Jupyter notebook session in the app notebooks directory.

Currently only works on Windows OS.

3.7.14 primaite.pol

Pattern of Life- Represents the actions of users on the network.

primaite.pol.green_pol	Implements Pattern of Life on the network (nodes and links).
primaite.pol.ier	Information Exchange Requirements for APE.
primaite.pol.red_agent_pol	Implements POL on the network (nodes and links) resulting from the red agent attack.

3.7.14.1 primaite.pol.green_pol

Implements Pattern of Life on the network (nodes and links).

Functions

apply_iers	Applies IERs to the links (link pattern of life).
apply_node_pol	Applies node pattern of life.

3.7.14.1.1 primaite.pol.green_pol.apply_iers

primaite.pol.green_pol.**apply_iers**(*network: MultiGraph*, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *links: Dict[str,* Link*]*, *iers: Dict[str,* IER*]*, *acl:* AccessControlList, *step: int*) → None

Applies IERs to the links (link pattern of life).

Args:

network: The network modelled in the environment nodes: The nodes within the environment links: The links within the environment iers: The IERs to apply to the links acl: The Access Control List step: The step number.

3.7.14.1.2 primaite.pol.green_pol.apply_node_pol

primaite.pol.green_pol.**apply_node_pol**(*nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *node_pol: Dict[str,* NodeStateInstructionGreen*]*, *step: int*) → None

Applies node pattern of life.

Args:
nodes: The nodes within the environment node_pol: The node pattern of life to apply step: The step number.

3.7.14.2 primaite.pol.ier

Information Exchange Requirements for APE.

Used to represent an information flow from source to destination.

Classes

IER	Information Exchange Requirement class.

3.7.14.2.1 primaite.pol.ier.IER

class primaite.pol.ier.**IER**(*_id: str*, *_start_step: int*, *_end_step: int*, *_load: int*, *_protocol: str*, *_port: str*, *_source_node_id: str*, *_dest_node_id: str*, *_mission_criticality: int*, *_running: bool = False*)

Bases: `object`

Information Exchange Requirement class.

Methods

get_dest_node_id	Gets IER destination node ID.
get_end_step	Gets IER end step.
get_id	Gets IER ID.
get_is_running	Informs whether the IER is currently running.
get_load	Gets IER load.
get_mission_criticality	Gets the IER mission criticality (used in the reward function).
get_port	Gets IER port.
get_protocol	Gets IER protocol.
get_source_node_id	Gets IER source node ID.
get_start_step	Gets IER start step.
set_is_running	Sets the running state of the IER.

__init__(*_id: str*, *_start_step: int*, *_end_step: int*, *_load: int*, *_protocol: str*, *_port: str*, *_source_node_id: str*, *_dest_node_id: str*, *_mission_criticality: int*, *_running: bool = False*) → None

Initialise an Information Exchange Request.

Parameters

- **_id** – The IER id
- **_start_step** – The step when this IER should start
- **_end_step** – The step when this IER should end
- **_load** – The load this IER should put on a link (bps)
- **_protocol** – The protocol of this IER
- **_port** – The port this IER runs on
- **_source_node_id** – The source node ID
- **_dest_node_id** – The destination node ID
- **_mission_criticality** – Criticality of this IER to the mission (0 none, 5 mission critical)
- **_running** – Indicates whether the IER is currently running

get_dest_node_id() → str

Gets IER destination node ID.

Returns:
IER destination node ID

get_end_step() → int

Gets IER end step.

Returns:
IER end step

get_id() → str

Gets IER ID.

Returns:
IER ID

get_is_running() → bool

Informs whether the IER is currently running.

Returns:
True if running

get_load() → int

Gets IER load.

Returns:
IER load

get_mission_criticality() → int

Gets the IER mission criticality (used in the reward function).

Returns:
Mission criticality value (0 lowest to 5 highest)

get_port() → str

Gets IER port.

Returns:
IER port

get_protocol() → str

Gets IER protocol.

Returns:
IER protocol

get_source_node_id() → str

Gets IER source node ID.

Returns:
IER source node ID

get_start_step() → int

Gets IER start step.

Returns:
IER start step

set_is_running(*_value: bool*) → None

Sets the running state of the IER.

Args:
_value: running status

3.7.14.3 primaite.pol.red_agent_pol

Implements POL on the network (nodes and links) resulting from the red agent attack.

Functions

apply_red_agent_iers	Applies IERs to the links (link POL) resulting from red agent attack.
apply_red_agent_node_pol	Applies node pattern of life.
is_red_ier_incoming	Checks if the RED IER is incoming.

3.7.14.3.1 primaite.pol.red_agent_pol.apply_red_agent_iers

primaite.pol.red_agent_pol.**apply_red_agent_iers**(*network: MultiGraph*, *nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *links: Dict[str,* Link*]*, *iers: Dict[str,* IER*]*, *acl:* AccessControlList, *step: int*) → None

Applies IERs to the links (link POL) resulting from red agent attack.

Args:
network: The network modelled in the environment nodes: The nodes within the environment links: The links within the environment iers: The red agent IERs to apply to the links acl: The Access Control List step: The step number.

3.7.14.3.2 primaite.pol.red_agent_pol.apply_red_agent_node_pol

primaite.pol.red_agent_pol.**apply_red_agent_node_pol**(*nodes: Dict[str,* ActiveNode | PassiveNode | ServiceNode*]*, *iers: Dict[str,* IER*]*, *node_pol: Dict[str,* NodeStateInstructionRed*]*, *step: int*) → None

Applies node pattern of life.

Args:
nodes: The nodes within the environment iers: The red agent IERs node_pol: The red agent node pattern of life to apply step: The step number.

3.7.14.3.3 primaite.pol.red_agent_pol.is_red_ier_incoming

primaite.pol.red_agent_pol.**is_red_ier_incoming**(*node:* ActiveNode | PassiveNode | ServiceNode, *iers: Dict[str,* IER*]*, *node_pol_type:* NodePOLType) → bool

Checks if the RED IER is incoming.

Parameters

- **node** (*NodeUnion*) – Destination node of the IER
- **iers** (*Dict[str,IER]*) – Directory of IERs
- **node_pol_type** (NodePOLType) – Type of Pattern-Of-Life

Returns
Whether the RED IER is incoming.

Return type
bool

3.7.15 primaite.primaite_session

Main entry point to PrimAITE. Configure training/evaluation experiments and input/output.

Classes

PrimaiteSession	The PrimaiteSession class.

3.7.15.1 primaite.primaite_session.PrimaiteSession

class primaite.primaite_session.**PrimaiteSession**(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*)

Bases: `object`

The PrimaiteSession class.

Provides a single learning and evaluation entry point for all training and lay down configurations.

Methods

close	Closes the agent.
eval_all_transactions_dict	Get the eval all transactions from file.
eval_av_reward_per_episode_dict	Get the eval av reward per episode from file.
evaluate	Evaluate the agent.
learn	Train the agent.
learn_all_transactions_dict	Get the learn all transactions from file.
learn_av_reward_per_episode_dict	Get the learn av reward per episode from file.
metadata_file_as_dict	Read the session_metadata.json file and return as a dict.
setup	Performs the session setup.

__init__(*training_config_path: str | Path | None = ''*, *lay_down_config_path: str | Path | None = ''*, *session_path: str | Path | None = None*, *legacy_training_config: bool = False*, *legacy_lay_down_config: bool = False*) → None

The PrimaiteSession constructor.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.
- **session_path** – directory path of the session to load
- **legacy_training_config** – True if the training config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **legacy_lay_down_config** – True if the lay_down config file is a legacy file from PrimAITE < 2.0, otherwise False.

close() → None

Closes the agent.

eval_all_transactions_dict() → Dict[Tuple[int, int], Dict[str, Any]]

Get the eval all transactions from file.

eval_av_reward_per_episode_dict() → Dict[int, float]

Get the eval av reward per episode from file.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters

kwargs – Any agent-framework specific key word args.

learn(***kwargs: Any*) → None

Train the agent.

Parameters

kwargs – Any agent-framework specific key word args.

learn_all_transactions_dict() → Dict[Tuple[int, int], Dict[str, Any]]

Get the learn all transactions from file.

learn_av_reward_per_episode_dict() → Dict[int, float]

Get the learn av reward per episode from file.

metadata_file_as_dict() → Dict[str, Any]

Read the session_metadata.json file and return as a dict.

setup() → None

Performs the session setup.

3.7.16 primaite.setup

Utilities to prepare the user's data folders.

primaite.setup.old_installation_clean_up
primaite.setup.reset_demo_notebooks
primaite.setup.reset_example_configs

3.7.16.1 primaite.setup.old_installation_clean_up

Functions

run	Perform the full clean-up.

3.7.16.1.1 primaite.setup.old_installation_clean_up.run

primaite.setup.old_installation_clean_up.**run**() → None

Perform the full clean-up.

3.7.16.2 primaite.setup.reset_demo_notebooks

Functions

run	Resets the demo jupyter notebooks in the users app notebooks directory.

3.7.16.2.1 primaite.setup.reset_demo_notebooks.run

primaite.setup.reset_demo_notebooks.**run**(*overwrite_existing: bool = True*) → None

Resets the demo jupyter notebooks in the users app notebooks directory.

Parameters

overwrite_existing – A bool to toggle replacing existing edited notebooks on or off.

3.7.16.3 primaite.setup.reset_example_configs

Functions

run	Resets the example config files in the users app config directory.

3.7.16.3.1 primaite.setup.reset_example_configs.run

primaite.setup.reset_example_configs.**run**(*overwrite_existing: bool = True*) → None

Resets the example config files in the users app config directory.

Parameters

overwrite_existing – A bool to toggle replacing existing edited config on or off.

3.7.17 primaite.simulator

Module attributes

TEMP_SIM_OUTPUT	A path at the repo root dir to use temporarily for sim output testing while in dev.

3.7.17.1 primaite.simulator.TEMP_SIM_OUTPUT

primaite.simulator.**TEMP_SIM_OUTPUT = WindowsPath('C:/Users/ChristopherMcCarthy/source/azure_devops/ma-dev-uk/PrimAITE/simulation_output')**

A path at the repo root dir to use temporarily for sim output testing while in dev.

primaite.simulator.core	Core of the PrimAITE Simulator.
primaite.simulator.domain	
primaite.simulator.file_system	
primaite.simulator.network	
primaite.simulator.system	

3.7.17.2 primaite.simulator.core

Core of the PrimAITE Simulator.

Classes

Action	This object stores data related to a single action.
ActionManager	ActionManager is used by *SimComponent* instances to keep track of actions.
ActionPermissionValidator	Base class for action validators.
AllowAllValidator	Always allows the action.
SimComponent	Extension of pydantic BaseModel with additional methods that must be defined by all classes in the simulator.

3.7.17.2.1 primaite.simulator.core.Action

class primaite.simulator.core.**Action**(*func: Callable[[List[str], Dict], None]*, *validator:* ActionPermissionValidator)

Bases: `object`

This object stores data related to a single action.

This includes the callable that can execute the action request, and the validator that will decide whether the action can be performed or not.

Methods

__init__(*func: Callable[[List[str], Dict], None]*, *validator:* ActionPermissionValidator) → None

Save the functions that are for this action.

Here's a description for the intended use of both of these.

`func` is a function that accepts a request and a context dict. Typically this would be a lambda function that invokes a class method of your SimComponent. For example if the component is a node and the action is for turning it off, then the SimComponent should have a turn_off(self) method that does not need to accept any args. Then, this Action will be given something like `func = lambda request, context: self.turn_off()`.

`validator` is an instance of a subclass of *ActionPermissionValidator*. This is essentially a callable that accepts *request* and *context* and returns a boolean to represent whether the permission is granted to perform the action.

Parameters

- **func** (*Callable[[List[str], Dict], None]*) – Function that performs the request.
- **validator** (ActionPermissionValidator) – Function that checks if the request is authenticated given the context.

3.7.17.2.2 primaite.simulator.core.ActionManager

class primaite.simulator.core.**ActionManager**

Bases: object

ActionManager is used by *SimComponent* instances to keep track of actions.

Its main purpose is to be a lookup from action name to action function and corresponding validation function. This class is responsible for providing a consistent API for processing actions as well as helpful error messages.

Methods

add_action	Add an action to this action manager.
process_request	Process an action request.

__init__() → None

Initialise ActionManager with an empty action lookup.

add_action(*name: str*, *action:* Action) → None

Add an action to this action manager.

Parameters

- **name** (*str*) – The string associated to this action.
- **action** (Action) – Action object.

process_request(*request: List[str]*, *context: Dict*) → None

Process an action request.

Parameters

- **request** (*List[str]*) – A list of strings which specify what action to take. The first string must be one of the allowed actions, i.e. it must be a key of self.actions. The subsequent strings in the list are passed as parameters to the action function.
- **context** (*Dict*) – Dictionary of additional information necessary to process or validate the request.

Raises

RuntimeError – If the request parameter does not have a valid action identifier as the first item.

3.7.17.2.3 primaite.simulator.core.ActionPermissionValidator

class primaite.simulator.core.**ActionPermissionValidator**

Bases: ABC

Base class for action validators.

The permissions manager is designed to be generic. So, although in the first instance the permissions are evaluated purely on membership to AccountGroup, this class can support validating permissions based on any arbitrary criteria.

Methods

abstract __call__(*request: List[str]*, *context: Dict*) → bool

Use the request and context paramters to decide whether the action should be permitted.

3.7.17.2.4 primaite.simulator.core.AllowAllValidator

class primaite.simulator.core.**AllowAllValidator**

Bases: *ActionPermissionValidator*

Always allows the action.

Methods

__call__(*request: List[str]*, *context: Dict*) → bool

Always allow the action.

3.7.17.2.5 primaite.simulator.core.SimComponent

class primaite.simulator.core.**SimComponent**(*, *uuid: str*, ***kwargs*)

Bases: BaseModel

Extension of pydantic BaseModel with additional methods that must be defined by all classes in the simulator.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Return a dictionary describing the state of this object and any objects managed by it.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
uuid	The component UUID.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters
action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

abstract describe_state() → Dict

Return a dictionary describing the state of this object and any objects managed by it.

This is similar to pydantic `model_dump()`, but it only outputs information about the objects owned by this object. If there are objects referenced by this object that are owned by something else, it is not included in this output.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

`model_dump_json`(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

`property model_extra: dict[str, Any] | None`

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

`model_fields: ClassVar[dict[str, FieldInfo]] = {'uuid': FieldInfo(annotation=str, required=True)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

`property model_fields_set: set[str]`

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

uuid: str

The component UUID.

3.7.17.3 primaite.simulator.domain

primaite.simulator.domain.account	User account simulation.
primaite.simulator.domain.controller	

3.7.17.3.1 primaite.simulator.domain.account

User account simulation.

Classes

Account	User accounts.
AccountType	Whether the account is intended for a user to log in or for a service to use.
PasswordPolicyLevel	Complexity requirements for account passwords.

3.7.17.3.1.1 primaite.simulator.domain.account.Account

class primaite.simulator.domain.account.**Account**(*, *uuid: str*, *num_logons: int = 0*, *num_logoffs: int = 0*, *num_group_changes: int = 0*, *username: str*, *password: str*, *account_type:* AccountType, *enabled: bool = True*, ***kwargs*)

Bases: *SimComponent*

User accounts.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describe state for agent observations.
dict	
disable	Set the status to disabled.
enable	Set the status to enabled.
from_orm	
json	
log_off	TODO.
log_on	TODO.
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
num_logons	The number of times this account was logged into since last reset.
num_logoffs	The number of times this account was logged out of since last reset.
num_group_changes	The number of times this account was moved in or out of an AccountGroup.
username	Account username.
password	Account password.
account_type	Account Type, currently this can be service account (used by apps) or user account.
enabled	

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

account_type: *AccountType*

Account Type, currently this can be service account (used by apps) or user account.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

describe_state() → Dict

Describe state for agent observations.

disable()

Set the status to disabled.

enable()

Set the status to enabled.

log_off() → None

TODO. Once the accounts are integrated with nodes, populate this accordingly.

log_on() → None

TODO. Once the accounts are integrated with nodes, populate this accordingly.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

`property model_extra: dict[str, Any] | None`

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

`model_fields: ClassVar[dict[str, FieldInfo]] = {'account_type': FieldInfo(annotation=AccountType, required=True), 'enabled': FieldInfo(annotation=bool, required=False, default=True), 'num_group_changes': FieldInfo(annotation=int, required=False, default=0), 'num_logoffs': FieldInfo(annotation=int, required=False, default=0), 'num_logons': FieldInfo(annotation=int, required=False, default=0), 'password': FieldInfo(annotation=str, required=True), 'username': FieldInfo(annotation=str, required=True), 'uuid': FieldInfo(annotation=str, required=True)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

`property model_fields_set: set[str]`

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

`classmethod model_json_schema`(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

`classmethod model_parametrized_name`(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

num_group_changes: int

The number of times this account was moved in or out of an AccountGroup.

num_logoffs: int

The number of times this account was logged out of since last reset.

num_logons: int

The number of times this account was logged into since last reset.

password: str

Account password.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

username: str

Account username.

uuid: str

The component UUID.

3.7.17.3.1.2 primaite.simulator.domain.account.AccountType

class primaite.simulator.domain.account.**AccountType**(*value*)

Bases: Enum

Whether the account is intended for a user to log in or for a service to use.

Attributes

SERVICE	Service accounts are used to grant permissions to software on nodes to perform actions
USER	User accounts are used to allow agents to log in and perform actions

SERVICE = 1

Service accounts are used to grant permissions to software on nodes to perform actions

USER = 2

User accounts are used to allow agents to log in and perform actions

3.7.17.3.1.3 primaite.simulator.domain.account.PasswordPolicyLevel

class primaite.simulator.domain.account.**PasswordPolicyLevel**(*value*)

Bases: Enum

Complexity requirements for account passwords.

Attributes

LOW
MEDIUM
HIGH

3.7.17.3.2 primaite.simulator.domain.controller

Classes

AccountGroup	Permissions are set at group-level and accounts can belong to these groups.
DomainController	Main object for controlling the domain.
GroupMembershipValidator	Permit actions based on group membership.
temp_application	Placeholder for application class for type hinting purposes.
temp_file	Placeholder for file class for type hinting purposes.
temp_folder	Placeholder for folder class for type hinting purposes.
temp_node	Placeholder for node class for type hinting purposes.

3.7.17.3.2.1 primaite.simulator.domain.controller.AccountGroup

class primaite.simulator.domain.controller.**AccountGroup**(*value*)

Bases: Enum

Permissions are set at group-level and accounts can belong to these groups.

Attributes

LOCAL_USER	For performing basic actions on a node
DOMAIN_USER	For performing basic actions to the domain
LOCAL_ADMIN	For full access to actions on a node
DOMAIN_ADMIN	For full access

DOMAIN_ADMIN = 4

For full access

DOMAIN_USER = 2

For performing basic actions to the domain

LOCAL_ADMIN = 3

For full access to actions on a node

LOCAL_USER = 1

For performing basic actions on a node

3.7.17.3.2.2 primaite.simulator.domain.controller.DomainController

class primaite.simulator.domain.controller.**DomainController**(**, uuid: str, accounts: ~typing.Dict[str, ~primaite.simulator.domain.account.Account] = {}, domain_group_membership: ~typing.Dict[~typing.Literal[<AccountGroup.DOMAIN_ADMIN: 4>, <AccountGroup.DOMAIN_USER: 2>], ~typing.List[~primaite.simulator.domain.account.Account]] = {}, local_group_membership: ~typing.Dict[~typing.Tuple[~primaite.simulator.domain.controller.AccountGroup, ~typing.Literal[<AccountGroup.LOCAL_ADMIN: 3>, <AccountGroup.LOCAL_USER: 1>]], ~typing.List[~primaite.simulator.domain.account.Account]] = {}, nodes: ~typing.Dict[str, ~primaite.simulator.domain.controller.temp_node] = {}, applications: ~typing.Dict[str, ~primaite.simulator.domain.controller.temp_application] = {}, folders: ~typing.List[~primaite.simulator.domain.controller.temp_folder] = {}, files: ~typing.List[~primaite.simulator.domain.controller.temp_file] = {}, **kwargs*)

Bases: *SimComponent*

Main object for controlling the domain.

Methods

add_account_to_group	TODO.
apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
check_account_permissions	Return a list of permission groups that this account has on this node.
construct	
copy	Returns a copy of the model.
create_account	TODO.
delete_account	TODO.
deregister_node	TODO.
describe_state	Return a dictionary describing the state of this object and any objects managed by it.
dict	

continues on next page

Table 5 – continued from previous page

from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
register_node	TODO.
remove_account_from_group	TODO.
reset_component_for_episode	Reset this component to its original state for a new episode.
rotate_account_credentials	TODO.
rotate_all_credentials	TODO.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

groups	
model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
accounts	
domain_group_membership	
local_group_membership	
nodes	
applications	
folders	
files	

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

add_account_to_group(*account:* Account, *group:* AccountGroup) → None

TODO.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

check_account_permissions(*account:* Account, *node:* temp_node) → List[*AccountGroup*]

Return a list of permission groups that this account has on this node.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

create_account(*username: str*, *password: str*, *account_type:* AccountType) → *Account*

TODO.

delete_account(*account:* Account) → None

TODO.

deregister_node(*node:* temp_node) → None

TODO.

abstract describe_state() → Dict

Return a dictionary describing the state of this object and any objects managed by it.

This is similar to pydantic `model_dump()`, but it only outputs information about the objects owned by this object. If there are objects referenced by this object that are owned by something else, it is not included in this output.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'accounts': FieldInfo(annotation=Dict[str, Account], required=False, default={}), 'applications': FieldInfo(annotation=Dict[str, temp_application], required=False, default={}), 'domain_group_membership': FieldInfo(annotation=Dict[Literal[AccountGroup, AccountGroup], List[Account]], required=False, default={}), 'files': FieldInfo(annotation=List[temp_file], required=False, default={}), 'folders': FieldInfo(annotation=List[temp_folder], required=False, default={}), 'local_group_membership': FieldInfo(annotation=Dict[Tuple[temp_node, Literal[AccountGroup, AccountGroup]], List[Account]], required=False, default={}), 'nodes': FieldInfo(annotation=Dict[str, temp_node], required=False, default={}), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

register_node(*node:* temp_node) → None

TODO.

remove_account_from_group(*account:* Account, *group:* AccountGroup) → None

TODO.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

rotate_account_credentials(*account:* Account) → None

TODO.

rotate_all_credentials() → None

TODO.

uuid: str

The component UUID.

3.7.17.3.2.3 primaite.simulator.domain.controller.GroupMembershipValidator

class primaite.simulator.domain.controller.**GroupMembershipValidator**(*allowed_groups: List[*AccountGroup*]*)

Bases: *ActionPermissionValidator*

Permit actions based on group membership.

Methods

__call__(*request: List[str]*, *context: Dict*) → bool

Permit the action if the request comes from an account which belongs to the right group.

__init__(*allowed_groups: List[*AccountGroup*]*) → None

Store a list of groups that should be granted permission.

Parameters
allowed_groups (*List[*AccountGroup*]*) – List of AccountGroups that are permitted to perform some action.

3.7.17.3.2.4 primaite.simulator.domain.controller.temp_application

class primaite.simulator.domain.controller.**temp_application**

Bases: object

Placeholder for application class for type hinting purposes.

Methods

3.7.17.3.2.5 primaite.simulator.domain.controller.temp_file

class primaite.simulator.domain.controller.**temp_file**

Bases: object

Placeholder for file class for type hinting purposes.

Methods

3.7.17.3.2.6 primaite.simulator.domain.controller.temp_folder

class primaite.simulator.domain.controller.**temp_folder**

Bases: object

Placeholder for folder class for type hinting purposes.

Methods

3.7.17.3.2.7 primaite.simulator.domain.controller.temp_node

class primaite.simulator.domain.controller.**temp_node**

Bases: object

Placeholder for node class for type hinting purposes.

Methods

3.7.17.4 primaite.simulator.file_system

primaite.simulator.file_system.file_system
primaite.simulator.file_system.file_system_file
primaite.simulator.file_system.file_system_file_type
primaite.simulator.file_system.file_system_folder
primaite.simulator.file_system.file_system_item_abc

3.7.17.4.1 primaite.simulator.file_system.file_system

Classes

FileSystem	Class that contains all the simulation File System.

3.7.17.4.1.1 primaite.simulator.file_system.file_system.FileSystem

class primaite.simulator.file_system.file_system.**FileSystem**(**, uuid: str, folders: Dict = {}, **kwargs*)

Bases: *SimComponent*

Class that contains all the simulation File System.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
copy_file	Copies a file from one folder to another.
create_file	Creates a FileSystemFile and adds it to the list of files.
create_folder	Creates a FileSystemFolder and adds it to the list of folders.
delete_file	Deletes a file and removes it from the files list.
delete_folder	Deletes a folder, removes it from the folders list and removes any child folders and files.

continues on next page

Table 6 – continued from previous page

describe_state	Get the current state of the FileSystem as a dict.
dict	
from_orm	
get_file_by_id	Checks if the file exists in any file system folders.
get_folder_by_id	Checks if the folder exists.
get_folder_by_name	Returns a the first folder with a matching name.
get_folders	Returns the list of folders.
get_random_file_type	Returns a random FileSystemFileTypeEnum.
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
move_file	Moves a file from one folder to another.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
folders	List containing all the folders in the file system.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping

specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

copy_file(*file:* FileSystemFile, *src_folder:* FileSystemFolder, *target_folder:* FileSystemFolder)

Copies a file from one folder to another.

can provide

Param
file: The file to move

Type
file: FileSystemFile

Param
src_folder: The folder where the file is located

Type
FileSystemFolder

Param
target_folder: The folder where the file should be moved to

Type
FileSystemFolder

create_file(*file_name: str*, *size: float | None = None*, *file_type:* FileSystemFileType *| None = None*, *folder:* FileSystemFolder *| None = None*, *folder_uuid: str | None = None*) → *FileSystemFile*

Creates a FileSystemFile and adds it to the list of files.

If no size or file_type are provided, one will be chosen randomly. If no folder_uuid or folder is provided, a new folder will be created.

Param
file_name: The file name

Type
file_name: str

Param
size: The size the file takes on disk.

Type
size: Optional[float]

Param
file_type: The type of the file

Type
Optional[*FileSystemFileType*]

Param
folder: The folder to add the file to

Type
folder: Optional[*FileSystemFolder*]

Param
folder_uuid: The uuid of the folder to add the file to

Type
folder_uuid: Optional[str]

create_folder(*folder_name: str*) → *FileSystemFolder*

Creates a FileSystemFolder and adds it to the list of folders.

Param
folder_name: The name of the folder

Type
folder_name: str

delete_file(*file:* FileSystemFile | *None* = *None*)

Deletes a file and removes it from the files list.

Parameters
file (*Optional[FileSystemFile]*) – The file to delete

delete_folder(*folder:* FileSystemFolder)

Deletes a folder, removes it from the folders list and removes any child folders and files.

Parameters
folder (FileSystemFolder) – The folder to remove

describe_state() → Dict

Get the current state of the FileSystem as a dict.

Returns
A dict containing the current state of the FileSystemFile.

folders: Dict

List containing all the folders in the file system.

get_file_by_id(*file_id: str*) → *FileSystemFile*

Checks if the file exists in any file system folders.

get_folder_by_id(*folder_id: str*) → *FileSystemFolder*

Checks if the folder exists.

Param
folder_id: The id of the folder to find

Type
folder_id: str

get_folder_by_name(*folder_name: str*) → *FileSystemFolder*

Returns a the first folder with a matching name.

Returns
Returns the first FileSydtemFolder with a matching name

get_folders() → Dict

Returns the list of folders.

get_random_file_type() → *FileSystemFileType*

Returns a random FileSystemFileTypeEnum.

Returns
A random file type Enum

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'folders': FieldInfo(annotation=Dict, required=False, default={}), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

move_file(*file:* FileSystemFile, *src_folder:* FileSystemFolder, *target_folder:* FileSystemFolder)

Moves a file from one folder to another.

can provide

Param
file: The file to move

Type
file: FileSystemFile

Param
src_folder: The folder where the file is located

Type
FileSystemFolder

Param
target_folder: The folder where the file should be moved to

Type
FileSystemFolder

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

uuid: str

The component UUID.

3.7.17.4.2 primaite.simulator.file_system.file_system_file

Classes

FileSystemFile	Class that represents a file in the simulation.

3.7.17.4.2.1 primaite.simulator.file_system.file_system_file.FileSystemFile

class primaite.simulator.file_system.file_system_file.**FileSystemFile**(*, *uuid: str*, *name: str*, *size: float = 0*, *file_type:* FileSystemFileType *= None*, ***kwargs*)

Bases: *FileSystemItem*

Class that represents a file in the simulation.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Get the current state of the FileSystemFile as a dict.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
file_type	The type of the FileSystemFile

__init__(***kwargs*)

Initialise FileSystemFile class.

Parameters

- **name** (*str*) – The name of the file.
- **file_type** (*Optional[FileSystemFileType]*) – The FileSystemFileType of the file
- **size** (*Optional[float]*) – The size of the FileSystemItem

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`describe_state`() → Dict

Get the current state of the FileSystemFile as a dict.

Returns
A dict containing the current state of the FileSystemFile.

`file_type: FileSystemFileType`

The type of the FileSystemFile

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct`(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

`model_copy`(***, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'file_type': FieldInfo(annotation=FileSystemFileType, required=False), 'name': FieldInfo(annotation=str, required=True), 'size': FieldInfo(annotation=float, required=False, default=0), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str

The name of the FileSystemItem.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

size: float

The size the item takes up on disk.

uuid: str

The component UUID.

3.7.17.4.3 primaite.simulator.file_system.file_system_file_type

Classes

FileSystemFileType	An enumeration of common file types.

3.7.17.4.3.1 primaite.simulator.file_system.file_system_file_type.FileSystemFileType

class primaite.simulator.file_system.file_system_file_type.**FileSystemFileType**(*value*)

Bases: `str`, `Enum`

An enumeration of common file types.

Attributes

UNKNOWN	Unknown file type.
TXT	Plain text file.
DOC	Microsoft Word document (.doc)
DOCX	Microsoft Word document (.docx)
PDF	Portable Document Format.
HTML	HyperText Markup Language file.
XML	Extensible Markup Language file.
CSV	Comma-Separated Values file.
XLS	Microsoft Excel file (.xls)
XLSX	Microsoft Excel file (.xlsx)
JPEG	JPEG image file.
PNG	PNG image file.
GIF	GIF image file.
BMP	Bitmap image file.
MP3	MP3 audio file.
WAV	WAV audio file.
MP4	MP4 video file.
AVI	AVI video file.
MKV	MKV video file.
FLV	FLV video file.
PPT	Microsoft PowerPoint file (.ppt)
PPTX	Microsoft PowerPoint file (.pptx)
JS	JavaScript file.
CSS	Cascading Style Sheets file.
PY	Python script file.
C	C source code file.
CPP	C++ source code file.
JAVA	Java source code file.
RAR	RAR archive file.
ZIP	ZIP archive file.
TAR	TAR archive file.
GZ	Gzip compressed file.

AVI = '17'

AVI video file.

BMP = '13'

Bitmap image file.

C = '25'

C source code file.

CPP = '26'

C++ source code file.

CSS = '23'

Cascading Style Sheets file.

CSV = '7'

Comma-Separated Values file.

DOC = '2'

Microsoft Word document (.doc)

DOCX = '3'

Microsoft Word document (.docx)

FLV = '19'

FLV video file.

GIF = '12'

GIF image file.

GZ = '31'

Gzip compressed file.

HTML = '5'

HyperText Markup Language file.

JAVA = '27'

Java source code file.

JPEG = '10'

JPEG image file.

JS = '22'

JavaScript file.

MKV = '18'

MKV video file.

MP3 = '14'

MP3 audio file.

MP4 = '16'

MP4 video file.

PDF = '4'

Portable Document Format.

PNG = '11'

PNG image file.

PPT = '20'

Microsoft PowerPoint file (.ppt)

PPTX = '21'

Microsoft PowerPoint file (.pptx)

PY = '24'
Python script file.

RAR = '28'
RAR archive file.

TAR = '30'
TAR archive file.

TXT = '1'
Plain text file.

UNKNOWN = '0'
Unknown file type.

WAV = '15'
WAV audio file.

XLS = '8'
Microsoft Excel file (.xls)

XLSX = '9'
Microsoft Excel file (.xlsx)

XML = '6'
Extensible Markup Language file.

ZIP = '29'
ZIP archive file.

3.7.17.4.4 primaite.simulator.file_system.file_system_folder

Classes

FileSystemFolder	Simulation FileSystemFolder.

3.7.17.4.4.1 primaite.simulator.file_system.file_system_folder.FileSystemFolder

class primaite.simulator.file_system.file_system_folder.**FileSystemFolder**(*, *uuid: str*, *name: str*, *size: float = 0*, *files: Dict = {}*, *is_quarantined: bool = False*, ***kwargs*)

Bases: *FileSystemItem*

Simulation FileSystemFolder.

Methods

add_file	Adds a file to the folder list.
apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Get the current state of the FileSystemFolder as a dict.
dict	
end_quarantine	Ends the quarantine of the File System Folder.
from_orm	
get_file_by_id	Return a FileSystemFile with the matching id.
get_file_by_name	Return a FileSystemFile with the matching id.
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
quarantine	Quarantines the File System Folder.
quarantine_status	Returns true if the folder is being quarantined.
remove_file	Removes a file from the folder list.
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	

continues on next page

Table 8 – continued from previous page

validate

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
files	List of files stored in the folder.
is_quarantined	Flag that marks the folder as quarantined if true.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

add_file(*file:* FileSystemFile)

Adds a file to the folder list.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`describe_state()` → Dict

Get the current state of the FileSystemFolder as a dict.

Returns
A dict containing the current state of the FileSystemFile.

`end_quarantine()`

Ends the quarantine of the File System Folder.

`files: Dict`

List of files stored in the folder.

`get_file_by_id`(*file_id: str*) → *FileSystemFile*

Return a FileSystemFile with the matching id.

`get_file_by_name`(*file_name: str*) → *FileSystemFile*

Return a FileSystemFile with the matching id.

`is_quarantined: bool`

Flag that marks the folder as quarantined if true.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct`(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

`property model_extra: dict[str, Any] | None`

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

`model_fields: ClassVar[dict[str, FieldInfo]] = {'files': FieldInfo(annotation=Dict, required=False, default={}), 'is_quarantined': FieldInfo(annotation=bool, required=False, default=False), 'name': FieldInfo(annotation=str, required=True), 'size': FieldInfo(annotation=float, required=False, default=0), 'uuid': FieldInfo(annotation=str, required=True)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

`property model_fields_set: set[str]`

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

`classmethod model_json_schema(`*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*`)` → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

`classmethod model_parametrized_name(`*params: tuple[type[Any], ...]*`)` → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

name: str

The name of the FileSystemItem.

quarantine()

Quarantines the File System Folder.

quarantine_status() → bool

Returns true if the folder is being quarantined.

remove_file(*file:* FileSystemFile | *None*)

Removes a file from the folder list.

The method can take a FileSystemFile object or a file id.

Param
file: The file to remove

Type
Optional[*FileSystemFile*]

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

size: float

The size the item takes up on disk.

uuid: str

The component UUID.

3.7.17.4.5 primaite.simulator.file_system.file_system_item_abc

Classes

FileSystemItem	Abstract base class for FileSystemItems used in the file system simulation.

3.7.17.4.5.1 primaite.simulator.file_system.file_system_item_abc.FileSystemItem

class primaite.simulator.file_system.file_system_item_abc.**FileSystemItem**(*, *uuid: str*, *name: str*, *size: float = 0*, ***kwargs*)

Bases: *SimComponent*

Abstract base class for FileSystemItems used in the file system simulation.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Returns the state of the FileSystemItem.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
name	The name of the FileSystemItem.
size	The size the item takes up on disk.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`describe_state`() → Dict

Returns the state of the FileSystemItem.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct`(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

`model_copy`(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

`model_dump`(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

`model_dump_json`(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

`property model_extra: dict[str, Any] | None`

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

`model_fields: ClassVar[dict[str, FieldInfo]] = {'name': FieldInfo(annotation=str, required=True), 'size': FieldInfo(annotation=float, required=False, default=0), 'uuid': FieldInfo(annotation=str, required=True)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

`property model_fields_set: set[str]`

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

`classmethod model_json_schema`(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str

The name of the FileSystemItem.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

size: float

The size the item takes up on disk.

uuid: str

The component UUID.

3.7.17.5 primaite.simulator.network

primaite.simulator.network.hardware
primaite.simulator.network.protocols
primaite.simulator.network.transmission
primaite.simulator.network.utils

3.7.17.5.1 primaite.simulator.network.hardware

primaite.simulator.network.hardware.base
primaite.simulator.network.hardware.nodes

3.7.17.5.1.1 primaite.simulator.network.hardware.base

Functions

generate_mac_address	Generate a random MAC Address.

3.7.17.5.1.2 primaite.simulator.network.hardware.base.generate_mac_address

primaite.simulator.network.hardware.base.**generate_mac_address**(*oui: str | None = None*) → str

Generate a random MAC Address.

Example

```
>>> generate_mac_address()
'ef:7e:97:c8:a8:ce'
```

```
>>> generate_mac_address(oui='aa:bb:cc')
'aa:bb:cc:42:ba:41'
```

Parameters

oui – The Organizationally Unique Identifier (OUI) portion of the MAC address. It should be a string with the first 3 bytes (24 bits) in the format "XX:XX:XX".

Raises

ValueError – If the 'oui' is not in the correct format (hexadecimal and 6 characters).

Classes

ARPCache	The ARPCache (Address Resolution Protocol) class.
ICMP	The ICMP (Internet Control Message Protocol) class.
Link	Represents a network link between NIC<-->NIC, NIC<-->SwitchPort, or SwitchPort<-->SwitchPort.
NIC	Models a Network Interface Card (NIC) in a computer or network device.
Node	A basic Node class that represents a node on the network.
NodeOperatingState	Enumeration of Node Operating States.
Switch	A class representing a Layer 2 network switch.
SwitchPort	Models a switch port in a network switch device.

3.7.17.5.1.3 primaite.simulator.network.hardware.base.ARPCache

class primaite.simulator.network.hardware.base.**ARPCache**(*sys_log:* SysLog)

Bases: `object`

The ARPCache (Address Resolution Protocol) class.

Responsible for maintaining a mapping between IP addresses and MAC addresses (ARP cache) for the network. It provides methods for looking up, adding, and removing entries, and for processing ARPPackets.

Methods

clear_arp_cache	Clear the entire ARP cache, removing all stored entries.
get_arp_cache_mac_address	Get the MAC address associated with an IP address.
get_arp_cache_nic	Get the NIC associated with an IP address.
process_arp_packet	Process a received ARP packet, handling both ARP requests and responses.
send_arp_request	Perform a standard ARP request for a given target IP address.

__init__(*sys_log:* SysLog)

Initialize an ARP (Address Resolution Protocol) cache.

Parameters
sys_log – The nodes sys log.

clear_arp_cache()

Clear the entire ARP cache, removing all stored entries.

get_arp_cache_mac_address(*ip_address: IPv4Address*) → str | None

Get the MAC address associated with an IP address.

Parameters
ip_address – The IP address to look up in the cache.

Returns
The MAC address associated with the IP address, or None if not found.

get_arp_cache_nic(*ip_address: IPv4Address*) → *NIC* | None

Get the NIC associated with an IP address.

Parameters
ip_address – The IP address to look up in the cache.

Returns
The NIC associated with the IP address, or None if not found.

process_arp_packet(*from_nic:* NIC, *arp_packet:* ARPPacket)

Process a received ARP packet, handling both ARP requests and responses.

If an ARP request is received for the local IP, a response is sent back. If an ARP response is received, the ARP cache is updated with the new entry.

Parameters

- **from_nic** – The NIC that received the ARP packet.

- **arp_packet** – The ARP packet to be processed.

send_arp_request(*target_ip_address: IPv4Address | str*)

Perform a standard ARP request for a given target IP address.

Broadcasts the request through all enabled NICs to determine the MAC address corresponding to the target IP address.

Parameters

target_ip_address – The target IP address to send an ARP request for.

3.7.17.5.1.4 primaite.simulator.network.hardware.base.ICMP

class primaite.simulator.network.hardware.base.**ICMP**(*sys_log:* SysLog, *arp_cache:* ARPCache)

Bases: `object`

The ICMP (Internet Control Message Protocol) class.

Provides functionalities for managing and handling ICMP packets, including echo requests and replies.

Methods

ping	Send an ICMP echo request (ping) to a target IP address and manage the sequence and identifier.
process_icmp	Process an ICMP packet, including handling echo requests and replies.

__init__(*sys_log:* SysLog, *arp_cache:* ARPCache)

Initialize the ICMP (Internet Control Message Protocol) service.

Parameters

- **sys_log** – The system log to store system messages and information.
- **arp_cache** – The ARP cache for resolving IP to MAC address mappings.

ping(*target_ip_address: IPv4Address*, *sequence: int = 0*, *identifier: int | None = None*) → Tuple[int, int | None]

Send an ICMP echo request (ping) to a target IP address and manage the sequence and identifier.

Parameters

- **target_ip_address** – The target IP address to send the ping.
- **sequence** – The sequence number of the echo request. Defaults to 0.
- **identifier** – An optional identifier for the ICMP packet. If None, a default will be used.

Returns

A tuple containing the next sequence number and the identifier, or (0, None) if the target IP address was not found in the ARP cache.

process_icmp(*frame:* Frame)

Process an ICMP packet, including handling echo requests and replies.

Parameters

frame – The Frame containing the ICMP packet to process.

3.7.17.5.1.5 primaite.simulator.network.hardware.base.Link

class primaite.simulator.network.hardware.base.**Link**(*, *uuid: str*, *endpoint_a:* NIC | SwitchPort, *endpoint_b:* NIC | SwitchPort, *bandwidth: float = 100.0*, *current_load: float = 0.0*, ***kwargs*)

Bases: *SimComponent*

Represents a network link between NIC<–>NIC, NIC<–>SwitchPort, or SwitchPort<–>SwitchPort.

Parameters

- **endpoint_a** – The first NIC or SwitchPort connected to the Link.
- **endpoint_b** – The second NIC or SwitchPort connected to the Link.
- **bandwidth** – The bandwidth of the Link in Mbps (default is 100 Mbps).

Methods

apply_action	Apply an action to the Link.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Get the current state of the Link as a dict.
dict	
endpoint_down	Let the Link know and endpoint has been brought down.
endpoint_up	Let the Link know and endpoint has been brought up.
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Link reset function.
schema	
schema_json	
transmit_frame	Send a network frame from one NIC or SwitchPort to another connected NIC or SwitchPort.
update_forward_refs	
validate	

Attributes

current_load_percent	Get the current load formatted as a percentage string.
is_up	Informs whether the link is up.
model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
endpoint_a	The first NIC or SwitchPort connected to the Link.
endpoint_b	The second NIC or SwitchPort connected to the Link.
bandwidth	The bandwidth of the Link in Mbps (default is 100 Mbps).
current_load	The current load on the link in Mbps.

__init__(***kwargs*)

Ensure that endpoint_a and endpoint_b are not the same NIC.

Connect the link to the NICs after creation.

Raises
ValueError – If endpoint_a and endpoint_b are the same NIC.

apply_action(*action: str*)

Apply an action to the Link.

Parameters
action (*str*) – The action to be applied.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

bandwidth: float

The bandwidth of the Link in Mbps (default is 100 Mbps).

copy(**, include: AbstractSetIntStr | MappingIntStrAny | None = None, exclude: AbstractSetIntStr | MappingIntStrAny | None = None, update: Dict[str, Any] | None = None, deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`current_load: float`

The current load on the link in Mbps.

`property current_load_percent: str`

Get the current load formatted as a percentage string.

`describe_state()` → Dict

Get the current state of the Link as a dict.

Returns
A dict containing the current state of the Link.

`endpoint_a: NIC | SwitchPort`

The first NIC or SwitchPort connected to the Link.

`endpoint_b: NIC | SwitchPort`

The second NIC or SwitchPort connected to the Link.

`endpoint_down()`

Let the Link know and endpoint has been brought down.

`endpoint_up()`

Let the Link know and endpoint has been brought up.

`property is_up: bool`

Informs whether the link is up.

This is based upon both NIC endpoints being enabled.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct(`*_fields_set: set[str] | None = None*, ***values: Any*`)` → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'bandwidth': FieldInfo(annotation=float, required=False, default=100.0), 'current_load': FieldInfo(annotation=float, required=False, default=0.0), 'endpoint_a': FieldInfo(annotation=Union[NIC, SwitchPort], required=True), 'endpoint_b': FieldInfo(annotation=Union[NIC, SwitchPort], required=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

reset_component_for_episode(*episode: int*)

Link reset function.

Reset:

- returns the link current_load to 0.

transmit_frame(*sender_nic:* NIC | SwitchPort, *frame:* Frame) → bool

Send a network frame from one NIC or SwitchPort to another connected NIC or SwitchPort.

Parameters

- **sender_nic** – The NIC or SwitchPort sending the frame.
- **frame** – The network frame to be sent.

Returns
True if the Frame can be sent, otherwise False.

uuid: str
The component UUID.

3.7.17.5.1.6 primaite.simulator.network.hardware.base.NIC

class primaite.simulator.network.hardware.base.**NIC**(*, *uuid: str*, *ip_address: IPv4Address*, *subnet_mask: str*, *gateway: IPv4Address*, *mac_address: str*, *speed: int = 100*, *mtu: int = 1500*, *wake_on_lan: bool = False*, *dns_servers: List[IPv4Address] = []*, *connected_node:* Node | *None = None*, *connected_link:* Link | *None = None*, *enabled: bool = False*, *pcap:* PacketCapture | *None = None*, ***kwargs*)

Bases: *SimComponent*

Models a Network Interface Card (NIC) in a computer or network device.

Parameters

- **ip_address** – The IPv4 address assigned to the NIC.
- **subnet_mask** – The subnet mask assigned to the NIC.
- **gateway** – The default gateway IP address for forwarding network traffic to other networks.
- **mac_address** – The MAC address of the NIC. Defaults to a randomly set MAC address.
- **speed** – The speed of the NIC in Mbps (default is 100 Mbps).
- **mtu** – The Maximum Transmission Unit (MTU) of the NIC in Bytes, representing the largest data packet size it can handle without fragmentation (default is 1500 B).
- **wake_on_lan** – Indicates if the NIC supports Wake-on-LAN functionality.
- **dns_servers** – List of IP addresses of DNS servers used for name resolution.

Methods

add_dns_server	Add a DNS server IP address.
apply_action	Apply an action to the NIC.
apply_timestep	Apply a timestep evolution to this component.
connect_link	Connect the NIC to a link.
construct	
copy	Returns a copy of the model.
describe_state	Get the current state of the NIC as a dict.
dict	
disable	Disable the NIC.

continues on next page

Table 9 – continued from previous page

disconnect_link	Disconnect the NIC from the connected Link.
enable	Attempt to enable the NIC.
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
receive_frame	Receive a network frame from the connected link if the NIC is enabled.
remove_dns_server	Remove a DNS server IP Address.
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
send_frame	Send a network frame from the NIC to the connected link.
update_forward_refs	
validate	

Attributes

ip_network	Return the IPv4Network of the NIC.
model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
ip_address	The IP address assigned to the NIC for communication on an IP-based network.
subnet_mask	The subnet mask assigned to the NIC.
gateway	The default gateway IP address for forwarding network traffic to other networks.
mac_address	The MAC address of the NIC.
speed	The speed of the NIC in Mbps.
mtu	The Maximum Transmission Unit (MTU) of the NIC in Bytes.
wake_on_lan	Indicates if the NIC supports Wake-on-LAN functionality.
dns_servers	List of IP addresses of DNS servers used for name resolution.
connected_node	The Node to which the NIC is connected.
connected_link	The Link to which the NIC is connected.
enabled	Indicates whether the NIC is enabled.
pcap	

__init__(***kwargs*)

NIC constructor.

Performs some type conversion the calls `super().__init__()`. Then performs some checking on the ip_address and gateway just to check that it's all been configured correctly.

Raises

ValueError – When the ip_address and gateway are the same. And when the ip_address/subnet mask are a network address.

add_dns_server(*ip_address: IPv4Address*)

Add a DNS server IP address.

Parameters

ip_address (*ipaddress.IPv4Address*) – The IP address of the DNS server to be added.

apply_action(*action: str*)

Apply an action to the NIC.

Parameters

action (*str*) – The action to be applied.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

connect_link(*link:* Link)

Connect the NIC to a link.

Parameters
link (Link) – The link to which the NIC is connected.

connected_link: *Link* **| None**

The Link to which the NIC is connected.

connected_node: *Node* **| None**

The Node to which the NIC is connected.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

describe_state() → Dict

Get the current state of the NIC as a dict.

Returns
A dict containing the current state of the NIC.

disable()

Disable the NIC.

disconnect_link()

Disconnect the NIC from the connected Link.

dns_servers: **List[IPv4Address]**

List of IP addresses of DNS servers used for name resolution.

enable()

Attempt to enable the NIC.

enabled: bool

Indicates whether the NIC is enabled.

gateway: IPv4Address

The default gateway IP address for forwarding network traffic to other networks. Randomly generated upon creation.

ip_address: IPv4Address

The IP address assigned to the NIC for communication on an IP-based network.

property ip_network: IPv4Network

Return the IPv4Network of the NIC.

Returns

The IPv4Network from the ip_address/subnet mask.

mac_address: str

The MAC address of the NIC. Defaults to a randomly set MAC address.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'connected_link': FieldInfo(annotation=Union[Link, NoneType], required=False), 'connected_node': FieldInfo(annotation=Union[Node, NoneType], required=False), 'dns_servers': FieldInfo(annotation=List[IPv4Address], required=False, default=[]), 'enabled': FieldInfo(annotation=bool, required=False, default=False), 'gateway': FieldInfo(annotation=IPv4Address, required=True), 'ip_address': FieldInfo(annotation=IPv4Address, required=True), 'mac_address': FieldInfo(annotation=str, required=True), 'mtu': FieldInfo(annotation=int, required=False, default=1500), 'pcap': FieldInfo(annotation=Union[PacketCapture, NoneType], required=False), 'speed': FieldInfo(annotation=int, required=False, default=100), 'subnet_mask': FieldInfo(annotation=str, required=True), 'uuid': FieldInfo(annotation=str, required=True), 'wake_on_lan': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, ***, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

mtu: int

The Maximum Transmission Unit (MTU) of the NIC in Bytes. Default is 1500 B

receive_frame(*frame:* Frame) → bool

Receive a network frame from the connected link if the NIC is enabled.

The Frame is passed to the Node.

Parameters
frame (Frame) – The network frame being received.

remove_dns_server(*ip_address: IPv4Address*)

Remove a DNS server IP Address.

Parameters
ip_address (*ipaddress.IPv4Address*) – The IP address of the DNS server to be removed.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

send_frame(*frame:* Frame) → bool

Send a network frame from the NIC to the connected link.

Parameters
frame (Frame) – The network frame to be sent.

speed: int

The speed of the NIC in Mbps. Default is 100 Mbps.

subnet_mask: str

The subnet mask assigned to the NIC.

uuid: str

The component UUID.

wake_on_lan: bool

Indicates if the NIC supports Wake-on-LAN functionality.

3.7.17.5.1.7 primaite.simulator.network.hardware.base.Node

class primaite.simulator.network.hardware.base.**Node**(*, *uuid: str*, *hostname: str*, *operating_state:* NodeOperatingState = *NodeOperatingState.OFF*, *nics: Dict[str,* NIC*] = {}*, *accounts: Dict = {}*, *applications: Dict = {}*, *services: Dict = {}*, *processes: Dict = {}*, *file_system:* FileSystem, *sys_log:* SysLog, *arp:* ARPCache, *icmp:* ICMP, *session_manager:* SessionManager, *software_manager:* SoftwareManager, *revealed_to_red: bool = False*, ***kwargs*)

Bases: *SimComponent*

A basic Node class that represents a node on the network.

This class manages the state of the node, including the NICs (Network Interface Cards), accounts, applications, services, processes, file system, and various managers like ARP, ICMP, SessionManager, and SoftwareManager.

Parameters

- **hostname** – The node hostname on the network.
- **operating_state** – The node operating state, either ON or OFF.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
connect_nic	Connect a NIC (Network Interface Card) to the node.
construct	
copy	Returns a copy of the model.
describe_state	Describe the state of the Node.
dict	

continues on next page

Table 10 – continued from previous page

disconnect_nic	Disconnect a NIC (Network Interface Card) from the node.
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
ping	Ping an IP address, performing a standard ICMP echo request/response.
power_off	Power off the Node, disabling its NICs if it is in the ON state.
power_on	Power on the Node, enabling its NICs if it is in the OFF state.
receive_frame	Receive a Frame from the connected NIC and process it.
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
send_frame	Send a Frame from the Node to the connected NIC.
show	Prints a table of the NICs on the Node..
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
hostname	The node hostname on the network.
operating_state	The hardware state of the node.
nics	The NICs on the node.
accounts	All accounts on the node.
applications	All applications on the node.
services	All services on the node.
processes	All processes on the node.
file_system	The nodes file system.
sys_log	
arp	
icmp	
session_manager	
software_manager	
revealed_to_red	Informs whether the node has been revealed to a red agent.

__init__(***kwargs*)

Initialize the Node with various components and managers.

This method initializes the ARP cache, ICMP handler, session manager, and software manager if they are not provided.

accounts: Dict

All accounts on the node.

applications: Dict

All applications on the node.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters
action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

connect_nic(*nic:* NIC)

Connect a NIC (Network Interface Card) to the node.

Parameters
nic – The NIC to connect.

Raises
NetworkError – If the NIC is already connected.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

describe_state() → Dict

Describe the state of the Node.

Returns
A dictionary representing the state of the node.

disconnect_nic(*nic:* NIC | *str*)

Disconnect a NIC (Network Interface Card) from the node.

Parameters
nic – The NIC to Disconnect, or its UUID.

Raises
NetworkError – If the NIC is not connected.

file_system: *FileSystem*

The nodes file system.

hostname: str

The node hostname on the network.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(***, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(***, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value

of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'accounts': FieldInfo(annotation=Dict, required=False, default={}), 'applications': FieldInfo(annotation=Dict, required=False, default={}), 'arp': FieldInfo(annotation=ARPCache, required=True), 'file_system': FieldInfo(annotation=FileSystem, required=True), 'hostname': FieldInfo(annotation=str, required=True), 'icmp': FieldInfo(annotation=ICMP, required=True), 'nics': FieldInfo(annotation=Dict[str, NIC], required=False, default={}), 'operating_state': FieldInfo(annotation=NodeOperatingState, required=False, default=<NodeOperatingState.OFF: 0>), 'processes': FieldInfo(annotation=Dict, required=False, default={}), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'services': FieldInfo(annotation=Dict, required=False, default={}), 'session_manager': FieldInfo(annotation=SessionManager, required=True), 'software_manager': FieldInfo(annotation=SoftwareManager, required=True), 'sys_log': FieldInfo(annotation=SysLog, required=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, ***, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

nics: Dict[str, *NIC*]

The NICs on the node.

operating_state: *NodeOperatingState*

The hardware state of the node.

ping(*target_ip_address: IPv4Address | str*, *pings: int = 4*) → bool

Ping an IP address, performing a standard ICMP echo request/response.

Parameters

- **target_ip_address** – The target IP address to ping.
- **pings** – The number of pings to attempt, default is 4.

Returns
True if the ping is successful, otherwise False.

power_off()

Power off the Node, disabling its NICs if it is in the ON state.

power_on()

Power on the Node, enabling its NICs if it is in the OFF state.

processes: Dict

All processes on the node.

receive_frame(*frame:* Frame, *from_nic:* NIC)

Receive a Frame from the connected NIC and process it.

Depending on the protocol, the frame is passed to the appropriate handler such as ARP or ICMP, or up to the SessionManager if no code manager exists.

Parameters

- **frame** – The Frame being received.
- **from_nic** – The NIC that received the frame.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

revealed_to_red: bool

Informs whether the node has been revealed to a red agent.

send_frame(*frame:* Frame)

Send a Frame from the Node to the connected NIC.

Parameters
frame – The Frame to be sent.

services: Dict

All services on the node.

show()

Prints a table of the NICs on the Node..

uuid: str

The component UUID.

3.7.17.5.1.8 primaite.simulator.network.hardware.base.NodeOperatingState

class primaite.simulator.network.hardware.base.**NodeOperatingState**(*value*)

Bases: Enum

Enumeration of Node Operating States.

Attributes

OFF	The node is powered off.
ON	The node is powered on.
SHUTTING_DOWN	The node is in the process of shutting down.
BOOTING	The node is in the process of booting up.

BOOTING = 3

The node is in the process of booting up.

OFF = 0

The node is powered off.

ON = 1

The node is powered on.

SHUTTING_DOWN = 2

The node is in the process of shutting down.

3.7.17.5.1.9 primaite.simulator.network.hardware.base.Switch

class primaite.simulator.network.hardware.base.**Switch**(*, *uuid: str*, *hostname: str*, *operating_state:* NodeOperatingState = *NodeOperatingState.OFF*, *nics: Dict[str,* NIC*] = {}*, *accounts: Dict = {}*, *applications: Dict = {}*, *services: Dict = {}*, *processes: Dict = {}*, *file_system:* FileSystem, *sys_log:* SysLog, *arp:* ARPCache, *icmp:* ICMP, *session_manager:* SessionManager, *software_manager:* SoftwareManager, *revealed_to_red: bool = False*, *num_ports: int = 24*, *switch_ports: Dict[int,* SwitchPort*] = {}*, *dst_mac_table: Dict[str,* SwitchPort*] = {}*, ***kwargs*)

Bases: *Node*

A class representing a Layer 2 network switch.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
connect_nic	Connect a NIC (Network Interface Card) to the node.
construct	
copy	Returns a copy of the model.
describe_state	TODO.
dict	
disconnect_link_from_port	Disconnect a given link from the specified port number on the switch.
disconnect_nic	Disconnect a NIC (Network Interface Card) from the node.
forward_frame	Forward a frame to the appropriate port based on the destination MAC address.
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.

continues on next page

Table 11 – continued from previous page

model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
ping	Ping an IP address, performing a standard ICMP echo request/response.
power_off	Power off the Node, disabling its NICs if it is in the ON state.
power_on	Power on the Node, enabling its NICs if it is in the OFF state.
receive_frame	Receive a Frame from the connected NIC and process it.
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
send_frame	Send a Frame from the Node to the connected NIC.
show	Prints a table of the SwitchPorts on the Switch.
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
num_ports	The number of ports on the switch.
switch_ports	The SwitchPorts on the switch.
dst_mac_table	A MAC address table mapping destination MAC addresses to corresponding SwitchPorts.

__init__(***kwargs*)

Initialize the Node with various components and managers.

This method initializes the ARP cache, ICMP handler, session manager, and software manager if they are not provided.

accounts: Dict

All accounts on the node.

applications: Dict

All applications on the node.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters
action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

connect_nic(*nic:* NIC)

Connect a NIC (Network Interface Card) to the node.

Parameters
nic – The NIC to connect.

Raises
NetworkError – If the NIC is already connected.

copy(**, include: AbstractSetIntStr | MappingIntStrAny | None = None, exclude: AbstractSetIntStr | MappingIntStrAny | None = None, update: Dict[str, Any] | None = None, deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`describe_state`() → Dict

TODO.

`disconnect_link_from_port`(*link:* Link, *port_number: int*)

Disconnect a given link from the specified port number on the switch.

Parameters

- **link** – The Link object to be disconnected.
- **port_number** – The port number on the switch from where the link should be disconnected.

Raises
NetworkError – When an invalid port number is provided or the link does not match the connection.

`disconnect_nic`(*nic:* NIC | *str*)

Disconnect a NIC (Network Interface Card) from the node.

Parameters
nic – The NIC to Disconnect, or its UUID.

Raises
NetworkError – If the NIC is not connected.

`dst_mac_table: Dict[str, SwitchPort]`

A MAC address table mapping destination MAC addresses to corresponding SwitchPorts.

`file_system: FileSystem`

The nodes file system.

`forward_frame`(*frame:* Frame, *incoming_port:* SwitchPort)

Forward a frame to the appropriate port based on the destination MAC address.

Parameters

- **frame** – The Frame to be forwarded.
- **incoming_port** – The port number from which the frame was received.

`hostname: str`

The node hostname on the network.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to

serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'accounts': FieldInfo(annotation=Dict, required=False, default={}), 'applications': FieldInfo(annotation=Dict, required=False, default={}), 'arp': FieldInfo(annotation=ARPCache, required=True), 'dst_mac_table': FieldInfo(annotation=Dict[str, SwitchPort], required=False, default={}), 'file_system': FieldInfo(annotation=FileSystem, required=True), 'hostname': FieldInfo(annotation=str, required=True), 'icmp': FieldInfo(annotation=ICMP, required=True), 'nics': FieldInfo(annotation=Dict[str, NIC], required=False, default={}), 'num_ports': FieldInfo(annotation=int, required=False, default=24), 'operating_state': FieldInfo(annotation=NodeOperatingState, required=False, default=<NodeOperatingState.OFF: 0>), 'processes': FieldInfo(annotation=Dict, required=False, default={}), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'services': FieldInfo(annotation=Dict, required=False, default={}), 'session_manager': FieldInfo(annotation=SessionManager, required=True), 'software_manager': FieldInfo(annotation=SoftwareManager, required=True), 'switch_ports': FieldInfo(annotation=Dict[int, SwitchPort], required=False, default={}), 'sys_log': FieldInfo(annotation=SysLog, required=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

`nics: Dict[str, NIC]`

The NICs on the node.

`num_ports: int`

The number of ports on the switch.

`operating_state: NodeOperatingState`

The hardware state of the node.

`ping`(*target_ip_address: IPv4Address | str*, *pings: int = 4*) → bool

Ping an IP address, performing a standard ICMP echo request/response.

Parameters

- **target_ip_address** – The target IP address to ping.
- **pings** – The number of pings to attempt, default is 4.

Returns
True if the ping is successful, otherwise False.

`power_off`()

Power off the Node, disabling its NICs if it is in the ON state.

`power_on`()

Power on the Node, enabling its NICs if it is in the OFF state.

`processes: Dict`

All processes on the node.

`receive_frame`(*frame:* Frame, *from_nic:* NIC)

Receive a Frame from the connected NIC and process it.

Depending on the protocol, the frame is passed to the appropriate handler such as ARP or ICMP, or up to the SessionManager if no code manager exists.

Parameters

- **frame** – The Frame being received.
- **from_nic** – The NIC that received the frame.

`reset_component_for_episode`(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

`revealed_to_red: bool`

Informs whether the node has been revealed to a red agent.

send_frame(*frame:* Frame)

Send a Frame from the Node to the connected NIC.

Parameters

frame – The Frame to be sent.

services: Dict

All services on the node.

show()

Prints a table of the SwitchPorts on the Switch.

switch_ports: Dict[int, *SwitchPort*]

The SwitchPorts on the switch.

uuid: str

The component UUID.

3.7.17.5.1.10 primaite.simulator.network.hardware.base.SwitchPort

class primaite.simulator.network.hardware.base.**SwitchPort**(*, *uuid: str*, *port_num: int = 1*, *mac_address: str*, *speed: int = 100*, *mtu: int = 1500*, *connected_node:* Switch | *None = None*, *connected_link:* Link | *None = None*, *enabled: bool = False*, *pcap:* PacketCapture | *None = None*, ***kwargs*)

Bases: *SimComponent*

Models a switch port in a network switch device.

Parameters

- **mac_address** – The MAC address of the SwitchPort. Defaults to a randomly set MAC address.
- **speed** – The speed of the SwitchPort in Mbps (default is 100 Mbps).
- **mtu** – The Maximum Transmission Unit (MTU) of the SwitchPort in Bytes, representing the largest data packet size it can handle without fragmentation (default is 1500 B).

Methods

apply_action	Apply an action to the SwitchPort.
apply_timestep	Apply a timestep evolution to this component.
connect_link	Connect the SwitchPort to a link.
construct	
copy	Returns a copy of the model.
describe_state	Get the current state of the SwitchPort as a dict.
dict	
disable	Disable the SwitchPort.
disconnect_link	Disconnect the SwitchPort from the connected Link.

continues on next page

Table 12 – continued from previous page

enable	Attempt to enable the SwitchPort.
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
receive_frame	Receive a network frame from the connected link if the SwitchPort is enabled.
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
send_frame	Send a network frame from the SwitchPort to the connected link.
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
port_num	
mac_address	The MAC address of the SwitchPort.
speed	The speed of the SwitchPort in Mbps.
mtu	The Maximum Transmission Unit (MTU) of the SwitchPort in Bytes.
connected_node	The Node to which the SwitchPort is connected.
connected_link	The Link to which the SwitchPort is connected.
enabled	Indicates whether the SwitchPort is enabled.
pcap	

__init__(***kwargs*)

The SwitchPort constructor.

apply_action(*action: str*)

Apply an action to the SwitchPort.

Parameters
action (*str*) – The action to be applied.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

connect_link(*link:* Link)

Connect the SwitchPort to a link.

Parameters
link – The link to which the SwitchPort is connected.

connected_link: *Link* **| None**

The Link to which the SwitchPort is connected.

connected_node: *Switch* **| None**

The Node to which the SwitchPort is connected.

copy(**, include: AbstractSetIntStr | MappingIntStrAny | None = None, exclude: AbstractSetIntStr | MappingIntStrAny | None = None, update: Dict[str, Any] | None = None, deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`describe_state`() → Dict

Get the current state of the SwitchPort as a dict.

Returns
A dict containing the current state of the SwitchPort.

`disable`()

Disable the SwitchPort.

`disconnect_link`()

Disconnect the SwitchPort from the connected Link.

`enable`()

Attempt to enable the SwitchPort.

`enabled: bool`

Indicates whether the SwitchPort is enabled.

`mac_address: str`

The MAC address of the SwitchPort. Defaults to a randomly set MAC address.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct`(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'connected_link': FieldInfo(annotation=Union[Link, NoneType], required=False), 'connected_node': FieldInfo(annotation=Union[Switch, NoneType], required=False), 'enabled': FieldInfo(annotation=bool, required=False, default=False), 'mac_address': FieldInfo(annotation=str, required=True), 'mtu': FieldInfo(annotation=int, required=False, default=1500), 'pcap': FieldInfo(annotation=Union[PacketCapture, NoneType], required=False), 'port_num': FieldInfo(annotation=int, required=False, default=1), 'speed': FieldInfo(annotation=int, required=False, default=100), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

mtu: int

The Maximum Transmission Unit (MTU) of the SwitchPort in Bytes. Default is 1500 B

receive_frame(*frame:* Frame) → bool

Receive a network frame from the connected link if the SwitchPort is enabled.

The Frame is passed to the Node.

Parameters
frame – The network frame being received.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

send_frame(*frame:* Frame) → bool

Send a network frame from the SwitchPort to the connected link.

Parameters
frame – The network frame to be sent.

speed: int

The speed of the SwitchPort in Mbps. Default is 100 Mbps.

uuid: str

The component UUID.

3.7.17.5.1.11 primaite.simulator.network.hardware.nodes

3.7.17.5.2 primaite.simulator.network.protocols

primaite.simulator.network.protocols.arp

3.7.17.5.2.1 primaite.simulator.network.protocols.arp

Classes

ARPEntry	Represents an entry in the ARP cache.
ARPPacket	Represents the ARP layer of a network frame.

3.7.17.5.2.2 primaite.simulator.network.protocols.arp.ARPEntry

class primaite.simulator.network.protocols.arp.**ARPEntry**(*, *mac_address: str*, *nic_uuid: str*)

Bases: BaseModel

Represents an entry in the ARP cache.

Parameters

- **mac_address** – The MAC address associated with the IP address.
- **nic** – The NIC through which the NIC with the IP address is reachable.

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
mac_address	
nic_uuid	

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

`property model_extra: dict[str, Any] | None`

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

`model_fields: ClassVar[dict[str, FieldInfo]] = {'mac_address': FieldInfo(annotation=str, required=True), 'nic_uuid': FieldInfo(annotation=str, required=True)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

`property model_fields_set: set[str]`

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

`classmethod model_json_schema(`*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*`)` → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

`classmethod model_parametrized_name(`*params: tuple[type[Any], ...]*`)` → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

3.7.17.5.2.3 primaite.simulator.network.protocols.arp.ARPPacket

class primaite.simulator.network.protocols.arp.**ARPPacket**(*, *request: bool = True*, *sender_mac_addr: str*, *sender_ip: IPv4Address*, *target_mac_addr: str | None = None*, *target_ip: IPv4Address*)

Bases: BaseModel

Represents the ARP layer of a network frame.

Parameters

- **request** – ARP operation. True if a request, False if a reply.
- **sender_mac_addr** – Sender MAC address.
- **sender_ip** – Sender IP address.
- **target_mac_addr** – Target MAC address.
- **target_ip** – Target IP address.

Example

```
>>> arp_request = ARPPacket(
...     sender_mac_addr="aa:bb:cc:dd:ee:ff",
...     sender_ip=IPv4Address("192.168.0.1"),
...     target_ip=IPv4Address("192.168.0.2")
... )
>>> arp_response = ARPPacket(
...     sender_mac_addr="aa:bb:cc:dd:ee:ff",
...     sender_ip=IPv4Address("192.168.0.1"),
...     target_ip=IPv4Address("192.168.0.2")
... )
```

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
generate_reply	Generate a new ARPPacket to be sent as a response with a given mac address.
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
request	ARP operation.
sender_mac_addr	Sender MAC address.
sender_ip	Sender IP address.
target_mac_addr	Target MAC address.
target_ip	Target IP address.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(**, include: AbstractSetIntStr | MappingIntStrAny | None = None, exclude: AbstractSetIntStr | MappingIntStrAny | None = None, update: Dict[str, Any] | None = None, deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

generate_reply(*mac_address: str*) → *ARPPacket*

Generate a new ARPPacket to be sent as a response with a given mac address.

Parameters
mac_address – The mac_address that was being sought after from the original target IP address.

Returns
A new instance of ARPPacket.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value

of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'request': FieldInfo(annotation=bool, required=False, default=True), 'sender_ip': FieldInfo(annotation=IPv4Address, required=True), 'sender_mac_addr': FieldInfo(annotation=str, required=True), 'target_ip': FieldInfo(annotation=IPv4Address, required=True), 'target_mac_addr': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

request: bool

ARP operation. True if a request, False if a reply.

sender_ip: IPv4Address

Sender IP address.

sender_mac_addr: str

Sender MAC address.

target_ip: IPv4Address

Target IP address.

target_mac_addr: str | None

Target MAC address.

3.7.17.5.3 primaite.simulator.network.transmission

primaite.simulator.network.transmission.data_link_layer	
primaite.simulator.network.transmission.network_layer	
primaite.simulator.network.transmission.primaite_layer	
primaite.simulator.network.transmission.transport_layer	

3.7.17.5.3.1 primaite.simulator.network.transmission.data_link_layer

Classes

EthernetHeader	Represents the Ethernet layer of a network frame.
Frame	Represents a complete network frame with all layers.

3.7.17.5.3.2 primaite.simulator.network.transmission.data_link_layer.EthernetHeader

class primaite.simulator.network.transmission.data_link_layer.**EthernetHeader**(*, *src_mac_addr: str*, *dst_mac_addr: str*)

Bases: BaseModel

Represents the Ethernet layer of a network frame.

Parameters

- **src_mac_addr** – Source MAC address.
- **dst_mac_addr** – Destination MAC address.

Example

```
>>> ethernet = EthernetHeader(
...     src_mac_addr='AA:BB:CC:DD:EE:FF',
...     dst_mac_addr='11:22:33:44:55:66'
... )
```

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
src_mac_addr	Source MAC address.
dst_mac_addr	Destination MAC address.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

dst_mac_addr: str

Destination MAC address.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'dst_mac_addr': FieldInfo(annotation=str, required=True), 'src_mac_addr': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

src_mac_addr: str
Source MAC address.

3.7.17.5.3.3 primaite.simulator.network.transmission.data_link_layer.Frame

class primaite.simulator.network.transmission.data_link_layer.**Frame**(*, *ethernet:* EthernetHeader, *ip:* IPPacket, *tcp:* TCPHeader | *None* = *None*, *udp:* UDPHeader | *None* = *None*, *icmp:* ICMPPacket | *None* = *None*, *arp:* ARPPacket | *None* = *None*, *primaite:* PrimaiteHeader, *payload:* *Any* | *None* = *None*, *sent_timestamp:* *datetime* | *None* = *None*, *received_timestamp:* *datetime* | *None* = *None*)

Bases: BaseModel

Represents a complete network frame with all layers.

Parameters

- **ethernet** – Ethernet layer.
- **ip** – IP layer.
- **tcp** – TCP layer.
- **payload** – Payload data in the frame.

Example

```
>>> from ipaddress import IPv4Address
>>> frame=Frame(
...     ethernet=EthernetHeader(
...         src_mac_addr='AA:BB:CC:DD:EE:FF',
...         dst_mac_addr='11:22:33:44:55:66'
...     ),
...     ip=IPPacket(
...         src_ip=IPv4Address('192.168.0.1'),
...         dst_ip=IPv4Address('10.0.0.1'),
...     ),
...     tcp=TCPHeader(
...         src_port=8080,
...         dst_port=80,
...     ),
...     payload=b"Hello, World!"
... )
```

Methods

construct	
copy	Returns a copy of the model.
decrement_ttl	Decrement the IPPacket ttl by 1.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
set_received_timestamp	Set the received_timestamp.
set_sent_timestamp	Set the sent_timestamp.
update_forward_refs	
validate	

Attributes

can_transmit	Informs whether the Frame can transmit based on the IPPacket tll being >= 1.
model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
size	The size of the Frame in Bytes.
size_Mbits	The daa transfer size of the Frame in Mbits.
ethernet	Ethernet header.
ip	IP packet.
tcp	TCP header.
udp	UDP header.
icmp	ICMP header.
arp	ARP packet.
primaite	PrimAITE header.
payload	Raw data payload.
sent_timestamp	The time the Frame was sent from the original source NIC.
received_timestamp	The time the Frame was received at the final destination NIC.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

arp: *ARPPacket* | None

ARP packet.

property can_transmit: bool

Informs whether the Frame can transmit based on the IPPacket tll being >= 1.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`decrement_ttl()`

Decrement the IPPacket ttl by 1.

`ethernet: EthernetHeader`

Ethernet header.

`icmp: ICMPPacket | None`

ICMP header.

`ip: IPPacket`

IP packet.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`classmethod model_construct(_fields_set: set[str] | None = None, **values: Any) → Model`

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

`model_copy(*, update: dict[str, Any] | None = None, deep: bool = False) → Model`

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'arp': FieldInfo(annotation=Union[ARPPacket, NoneType], required=False), 'ethernet': FieldInfo(annotation=EthernetHeader, required=True), 'icmp': FieldInfo(annotation=Union[ICMPPacket, NoneType], required=False), 'ip': FieldInfo(annotation=IPPacket, required=True), 'payload': FieldInfo(annotation=Union[Any, NoneType], required=False), 'primaite': FieldInfo(annotation=PrimaiteHeader, required=True), 'received_timestamp': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'sent_timestamp': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'tcp': FieldInfo(annotation=Union[TCPHeader, NoneType], required=False), 'udp': FieldInfo(annotation=Union[UDPHeader, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

payload: Any | None

Raw data payload.

primaite: *PrimaiteHeader*

PrimAITE header.

received_timestamp: datetime | None

The time the Frame was received at the final destination NIC.

sent_timestamp: datetime | None

The time the Frame was sent from the original source NIC.

set_received_timestamp()

Set the received_timestamp.

set_sent_timestamp()

Set the sent_timestamp.

property size: float

The size of the Frame in Bytes.

property size_Mbits: float

The daa transfer size of the Frame in Mbits.

tcp: *TCPHeader* | None

TCP header.

udp: *UDPHeader* | None

UDP header.

3.7.17.5.3.4 primaite.simulator.network.transmission.network_layer

Functions

get_icmp_type_code_description	Maps ICMPType and code pairings to their respective description.

3.7.17.5.3.5 primaite.simulator.network.transmission.network_layer.get_icmp_type_code_description

primaite.simulator.network.transmission.network_layer.**get_icmp_type_code_description**(*icmp_type:* ICMPType, *icmp_code: int*) → str | None

Maps ICMPType and code pairings to their respective description.

Parameters

- **icmp_type** – An ICMPType.
- **icmp_code** – An icmp code.

Returns

The icmp type and code pairing description if it exists, otherwise returns None.

Classes

ICMPPacket	Models an ICMP Packet.
ICMPType	Enumeration of common ICMP (Internet Control Message Protocol) types.
IPPacket	Represents the IP layer of a network frame.
IPProtocol	Enum representing transport layer protocols in IP header.
Precedence	Enum representing the Precedence levels in Quality of Service (QoS) for IP packets.

3.7.17.5.3.6 primaite.simulator.network.transmission.network_layer.ICMPPacket

class primaite.simulator.network.transmission.network_layer.**ICMPPacket**(*, *icmp_type:* ICMPType = *ICMPType.ECHO_REQUEST*, *icmp_code: int = 0*, *identifier: int*, *sequence: int = 0*)

Bases: BaseModel

Models an ICMP Packet.

Methods

code_description	The icmp_code description.
construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
icmp_type	ICMP Type.
icmp_code	ICMP Code.
identifier	ICMP identifier (16 bits randomly generated).
sequence	ICMP message sequence number.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

code_description() → str

The icmp_code description.

copy(**, include: AbstractSetIntStr | MappingIntStrAny | None = None, exclude: AbstractSetIntStr | MappingIntStrAny | None = None, update: Dict[str, Any] | None = None, deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

icmp_code: int

ICMP Code.

icmp_type: *ICMPType*

ICMP Type.

identifier: int

ICMP identifier (16 bits randomly generated).

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(***, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(***, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to

exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'icmp_code': FieldInfo(annotation=int, required=False, default=0), 'icmp_type': FieldInfo(annotation=ICMPType, required=False, default=<ICMPType.ECHO_REQUEST: 8>), 'identifier': FieldInfo(annotation=int, required=True), 'sequence': FieldInfo(annotation=int, required=False, default=0)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

sequence: int

ICMP message sequence number.

3.7.17.5.3.7 primaite.simulator.network.transmission.network_layer.ICMPType

class primaite.simulator.network.transmission.network_layer.**ICMPType**(*value*)

Bases: Enum

Enumeration of common ICMP (Internet Control Message Protocol) types.

Attributes

ECHO_REPLY	Echo Reply message.
DESTINATION_UNREACHABLE	Destination Unreachable.
REDIRECT	Redirect.
ECHO_REQUEST	Echo Request (ping).
ROUTER_ADVERTISEMENT	Router Advertisement.
ROUTER_SOLICITATION	Router discovery/selection/solicitation.
TIME_EXCEEDED	Time Exceeded.
TIMESTAMP_REQUEST	Timestamp Request.
TIMESTAMP_REPLY	Timestamp Reply.

DESTINATION_UNREACHABLE = 3

Destination Unreachable.

ECHO_REPLY = 0

Echo Reply message.

ECHO_REQUEST = 8

Echo Request (ping).

REDIRECT = 5

Redirect.

ROUTER_ADVERTISEMENT = 10

Router Advertisement.

ROUTER_SOLICITATION = 11

Router discovery/selection/solicitation.

TIMESTAMP_REPLY = 14

Timestamp Reply.

TIMESTAMP_REQUEST = 13

Timestamp Request.

TIME_EXCEEDED = 11

Time Exceeded.

3.7.17.5.3.8 primaite.simulator.network.transmission.network_layer.IPPacket

class primaite.simulator.network.transmission.network_layer.**IPPacket**(*, *src_ip: IPv4Address*, *dst_ip: IPv4Address*, *protocol:* IPProtocol = *IPProtocol.TCP*, *ttl: int* = *64*, *precedence:* Precedence = *Precedence.ROUTINE*)

Bases: `BaseModel`

Represents the IP layer of a network frame.

Parameters

- **src_ip** – Source IP address.
- **dst_ip** – Destination IP address.
- **protocol** – The IP protocol (default is TCP).
- **ttl** – Time to Live (TTL) for the packet.
- **precedence** – Precedence level for Quality of Service (QoS).

Example

```
>>> from ipaddress import IPv4Address
>>> ip_packet = IPPacket(
...     src_ip=IPv4Address('192.168.0.1'),
...     dst_ip=IPv4Address('10.0.0.1'),
...     protocol=IPProtocol.TCP,
...     ttl=64,
...     precedence=Precedence.CRITICAL
... )
```

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
src_ip	Source IP address.
dst_ip	Destination IP address.
protocol	IPProtocol.
ttl	Time to Live (TTL) for the packet.
precedence	Precedence level for Quality of Service (default is Precedence.ROUTINE).

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

dst_ip: IPv4Address

Destination IP address.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'dst_ip': FieldInfo(annotation=IPv4Address, required=True), 'precedence': FieldInfo(annotation=Precedence, required=False, default=<Precedence.ROUTINE: 0>), 'protocol': FieldInfo(annotation=IPProtocol, required=False, default=<IPProtocol.TCP: 'tcp'>), 'src_ip': FieldInfo(annotation=IPv4Address, required=True), 'ttl': FieldInfo(annotation=int, required=False, default=64)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

precedence: *Precedence*

Precedence level for Quality of Service (default is Precedence.ROUTINE).

protocol: *IPProtocol*

IPProtocol.

src_ip: IPv4Address

Source IP address.

ttl: int

Time to Live (TTL) for the packet.

3.7.17.5.3.9 primaite.simulator.network.transmission.network_layer.IPProtocol

class primaite.simulator.network.transmission.network_layer.**IPProtocol**(*value*)

Bases: Enum

Enum representing transport layer protocols in IP header.

Attributes

TCP
UDP
ICMP

3.7.17.5.3.10 primaite.simulator.network.transmission.network_layer.Precedence

class primaite.simulator.network.transmission.network_layer.**Precedence**(*value*)

Bases: Enum

Enum representing the Precedence levels in Quality of Service (QoS) for IP packets.

Precedence values range from 0 to 7, indicating different levels of priority.

Members: - ROUTINE: 0 - Lowest priority level, used for ordinary data traffic that does not require special treatment. - PRIORITY: 1 - Higher priority than ROUTINE, used for traffic that needs a bit more importance. - IMMEDIATE: 2 - Used for more urgent traffic that requires immediate handling and minimal delay. - FLASH: 3 - Used for highly urgent and important traffic that should be processed with high priority. - FLASH_OVERRIDE: 4 - Higher priority than FLASH, used for critical traffic that takes precedence over most traffic. - CRITICAL: 5 - Reserved for critical commands or control messages that are vital to the operation of the network. - INTERNET: 6

- Used for network control messages, such as routing updates, for maintaining network operations. - NETWORK: 7 - Highest priority for the most critical network control messages, such as routing protocol hellos.

Attributes

ROUTINE	Lowest priority level, used for ordinary data traffic that does not require special treatment.
PRIORITY	Higher priority than ROUTINE, used for traffic that needs a bit more importance.
IMMEDIATE	Used for more urgent traffic that requires immediate handling and minimal delay.
FLASH	Used for highly urgent and important traffic that should be processed with high priority.
FLASH_OVERRIDE	Has higher priority than FLASH, used for critical traffic that takes precedence over most other traffic.
CRITICAL	Reserved for critical commands or emergency control messages that are vital to the operation of the network.
INTERNET	Used for network control messages, such as routing updates, essential for maintaining network operations.
NETWORK	Highest priority level, used for the most critical network control messages, such as routing protocol hellos.

CRITICAL = 5

Reserved for critical commands or emergency control messages that are vital to the operation of the network.

FLASH = 3

Used for highly urgent and important traffic that should be processed with high priority.

FLASH_OVERRIDE = 4

Has higher priority than FLASH, used for critical traffic that takes precedence over most other traffic.

IMMEDIATE = 2

Used for more urgent traffic that requires immediate handling and minimal delay.

INTERNET = 6

Used for network control messages, such as routing updates, essential for maintaining network operations.

NETWORK = 7

Highest priority level, used for the most critical network control messages, such as routing protocol hellos.

PRIORITY = 1

Higher priority than ROUTINE, used for traffic that needs a bit more importance.

ROUTINE = 0

Lowest priority level, used for ordinary data traffic that does not require special treatment.

3.7.17.5.3.11 primaite.simulator.network.transmission.primaite_layer

Classes

AgentSource	The agent source of the transmission.
DataStatus	The status of the data in transmission.
PrimaiteHeader	A custom header for carrying PrimAITE transmission metadata required for RL.

3.7.17.5.3.12 primaite.simulator.network.transmission.primaite_layer.AgentSource

class primaite.simulator.network.transmission.primaite_layer.**AgentSource**(*value*)

Bases: Enum

The agent source of the transmission.

Members:

- RED: 1
- GREEN: 2
- BLUE: 3

Attributes

RED
GREEN
BLUE

3.7.17.5.3.13 primaite.simulator.network.transmission.primaite_layer.DataStatus

class primaite.simulator.network.transmission.primaite_layer.**DataStatus**(*value*)

Bases: Enum

The status of the data in transmission.

Members:

- GOOD: 1
- COMPROMISED: 2
- CORRUPT: 3

Attributes

GOOD
COMPROMISED
CORRUPT

3.7.17.5.3.14 primaite.simulator.network.transmission.primaite_layer.PrimaiteHeader

class primaite.simulator.network.transmission.primaite_layer.**PrimaiteHeader**(*, *agent_source:* AgentSource = *AgentSource.GREEN*, *data_status:* DataStatus = *DataStatus.GOOD*)

Bases: BaseModel

A custom header for carrying PrimAITE transmission metadata required for RL.

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
agent_source	
data_status	

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`classmethod model_construct(_fields_set: set[str] | None = None, **values: Any) → Model`

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

`model_copy(*, update: dict[str, Any] | None = None, deep: bool = False) → Model`

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

`model_dump(*, mode: Literal['json', 'python'] | str = 'python', include: IncEx = None, exclude: IncEx = None, by_alias: bool = False, exclude_unset: bool = False, exclude_defaults: bool = False, exclude_none: bool = False, round_trip: bool = False, warnings: bool = True) → dict[str, Any]`

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

`model_dump_json(*, indent: int | None = None, include: IncEx = None, exclude: IncEx = None, by_alias: bool = False, exclude_unset: bool = False, exclude_defaults: bool = False, exclude_none: bool = False, round_trip: bool = False, warnings: bool = True) → str`

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'agent_source': FieldInfo(annotation=AgentSource, required=False, default=<AgentSource.GREEN: 2>), 'data_status': FieldInfo(annotation=DataStatus, required=False, default=<DataStatus.GOOD: 1>)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

3.7.17.5.3.15 primaite.simulator.network.transmission.transport_layer

Classes

Port	Enumeration of common known TCP/UDP ports used by protocols for operation of network applications.
TCPFlags	Enum representing TCP control flags used in a TCP connection.
TCPHeader	Represents a TCP header for the transport layer of a Network Frame.
UDPHeader	Represents a UDP header for the transport layer of a Network Frame.

3.7.17.5.3.16 primaite.simulator.network.transmission.transport_layer.Port

class primaite.simulator.network.transmission.transport_layer.**Port**(*value*)

Bases: Enum

Enumeration of common known TCP/UDP ports used by protocols for operation of network applications.

Attributes

WOL	Wake-on-Lan (WOL) - Used to turn or awaken a computer from sleep mode by a network message.
FTP_DATA	File Transfer [Default Data]
FTP	File Transfer Protocol (FTP) - FTP control (command)
SSH	Secure Shell (SSH) - Used for secure remote access and command execution.
SMTP	Simple Mail Transfer Protocol (SMTP) - Used for email delivery between servers.
DNS	Domain Name System (DNS) - Used for translating domain names to IP addresses.
HTTP	HyperText Transfer Protocol (HTTP) - Used for web traffic.
POP3	Post Office Protocol version 3 (POP3) - Used for retrieving emails from a mail server.
SFTP	Secure File Transfer Protocol (SFTP) - Used for secure file transfer over SSH.
NTP	Network Time Protocol (NTP) - Used for clock synchronization between computer systems.
IMAP	Internet Message Access Protocol (IMAP) - Used for retrieving emails from a mail server.
SNMP	Simple Network Management Protocol (SNMP) - Used for network device management.
SNMP_TRAP	SNMP Trap - Used for sending SNMP notifications (traps) to a network management system.
ARP	Address resolution Protocol - Used to connect a MAC address to an IP address.
LDAP	Lightweight Directory Access Protocol (LDAP) - Used for accessing and modifying directory information.
HTTPS	HyperText Transfer Protocol Secure (HTTPS) - Used for secure web traffic.
SMB	Server Message Block (SMB) - Used for file sharing and printer sharing in Windows environments.
IPP	Internet Printing Protocol (IPP) - Used for printing over the internet or an intranet.
SQL_SERVER	Microsoft SQL Server Database Engine - Used for communication with the SQL Server.
MYSQL	MySQL Database Server - Used for MySQL database communication.
RDP	Remote Desktop Protocol (RDP) - Used for remote desktop access to Windows machines.
RTP	Real-time Transport Protocol (RTP) - Used for transmitting real-time media, e.g., audio and video.
RTP_ALT	Alternative port for RTP (RTP_ALT) - Used in some configurations for transmitting real-time media.
DNS_ALT	Alternative port for DNS (DNS_ALT) - Used in some configurations for DNS service.
HTTP_ALT	Alternative port for HTTP (HTTP_ALT) - Often used as an alternative HTTP port for web applications.
HTTPS_ALT	Alternative port for HTTPS (HTTPS_ALT) - Used in some configurations for secure web traffic.

ARP = 219

Address resolution Protocol - Used to connect a MAC address to an IP address.

DNS = 53

Domain Name System (DNS) - Used for translating domain names to IP addresses.

DNS_ALT = 5353

Alternative port for DNS (DNS_ALT) - Used in some configurations for DNS service.

FTP = 21

File Transfer Protocol (FTP) - FTP control (command)

FTP_DATA = 20

File Transfer [Default Data]

HTTP = 80

HyperText Transfer Protocol (HTTP) - Used for web traffic.

HTTPS = 443

HyperText Transfer Protocol Secure (HTTPS) - Used for secure web traffic.

HTTPS_ALT = 8443

Alternative port for HTTPS (HTTPS_ALT) - Used in some configurations for secure web traffic.

HTTP_ALT = 8080

Alternative port for HTTP (HTTP_ALT) - Often used as an alternative HTTP port for web applications.

IMAP = 143

Internet Message Access Protocol (IMAP) - Used for retrieving emails from a mail server.

IPP = 631

Internet Printing Protocol (IPP) - Used for printing over the internet or an intranet.

LDAP = 389

Lightweight Directory Access Protocol (LDAP) - Used for accessing and modifying directory information.

MYSQL = 3306

MySQL Database Server - Used for MySQL database communication.

NTP = 123

Network Time Protocol (NTP) - Used for clock synchronization between computer systems.

POP3 = 110

Post Office Protocol version 3 (POP3) - Used for retrieving emails from a mail server.

RDP = 3389

Remote Desktop Protocol (RDP) - Used for remote desktop access to Windows machines.

RTP = 5004

Real-time Transport Protocol (RTP) - Used for transmitting real-time media, e.g., audio and video.

RTP_ALT = 5005

Alternative port for RTP (RTP_ALT) - Used in some configurations for transmitting real-time media.

SFTP = 115

Secure File Transfer Protocol (SFTP) - Used for secure file transfer over SSH.

SMB = 445

Server Message Block (SMB) - Used for file sharing and printer sharing in Windows environments.

SMTP = 25

Simple Mail Transfer Protocol (SMTP) - Used for email delivery between servers.

SNMP = 161

Simple Network Management Protocol (SNMP) - Used for network device management.

SNMP_TRAP = 162

SNMP Trap - Used for sending SNMP notifications (traps) to a network management system.

SQL_SERVER = 1433

Microsoft SQL Server Database Engine - Used for communication with the SQL Server.

SSH = 22

Secure Shell (SSH) - Used for secure remote access and command execution.

WOL = 9

Wake-on-Lan (WOL) - Used to turn or awaken a computer from sleep mode by a network message.

3.7.17.5.3.17 primaite.simulator.network.transmission.transport_layer.TCPFlags

class primaite.simulator.network.transmission.transport_layer.**TCPFlags**(*value*)

Bases: Enum

Enum representing TCP control flags used in a TCP connection.

Flags are used to indicate a particular state of the connection or provide additional information.

Members: - SYN: (1) - Used in the first step of connection establishment phase or 3-way handshake process between two hosts. - ACK: (2) - Used to acknowledge packets that are successfully received by the host. - FIN: (4) - Used to request connection termination when there is no more data from the sender. - RST: (8) - Used to terminate the connection if there is an issue with the TCP connection.

Attributes

SYN
ACK
FIN
RST

3.7.17.5.3.18 primaite.simulator.network.transmission.transport_layer.TCPHeader

class primaite.simulator.network.transmission.transport_layer.**TCPHeader**(*, *src_port: ~primaite.simulator.network.transmission.tra*, *dst_port: ~primaite.simulator.network.transmission.tra*, *flags: ~typing.List[~primaite.simulator.network.tra* = [<TCPFlags.SYN: 1>])

Bases: BaseModel

Represents a TCP header for the transport layer of a Network Frame.

Parameters

- **src_port** – Source port.
- **dst_port** – Destination port.
- **flags** – TCP flags (list of TCPFlags members).

Example

```
>>> tcp_header = TCPHeader(
...     src_port=Port.HTTP_ALT,
...     dst_port=Port.HTTP,
...     flags=[TCPFlags.SYN, TCPFlags.ACK]
... )
```

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
src_port	
dst_port	
flags	

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`classmethod model_construct`(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

`model_copy`(***, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

`model_dump`(***, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

`model_dump_json`(***, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'dst_port': FieldInfo(annotation=Port, required=True), 'flags': FieldInfo(annotation=List[TCPFlags], required=False, default=[<TCPFlags.SYN: 1>]), 'src_port': FieldInfo(annotation=Port, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

3.7.17.5.3.19 primaite.simulator.network.transmission.transport_layer.UDPHeader

class primaite.simulator.network.transmission.transport_layer.**UDPHeader**(*, *src_port:* Port | *int*, *dst_port:* Port | *int*)

Bases: `BaseModel`

Represents a UDP header for the transport layer of a Network Frame.

Parameters

- **src_port** – Source port.
- **dst_port** – Destination port.

Example

```
>>> udp_header = UDPHeader(
...     src_port=Port.HTTP_ALT,
...     dst_port=Port.HTTP,
... )
```

Methods

construct	
copy	Returns a copy of the model.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
src_port	
dst_port	

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`classmethod model_construct(_fields_set: set[str] | None = None, **values: Any) → Model`

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

`model_copy(*, update: dict[str, Any] | None = None, deep: bool = False) → Model`

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

`model_dump(*, mode: Literal['json', 'python'] | str = 'python', include: IncEx = None, exclude: IncEx = None, by_alias: bool = False, exclude_unset: bool = False, exclude_defaults: bool = False, exclude_none: bool = False, round_trip: bool = False, warnings: bool = True) → dict[str, Any]`

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

`model_dump_json(*, indent: int | None = None, include: IncEx = None, exclude: IncEx = None, by_alias: bool = False, exclude_unset: bool = False, exclude_defaults: bool = False, exclude_none: bool = False, round_trip: bool = False, warnings: bool = True) → str`

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:

A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'dst_port': FieldInfo(annotation=Union[Port, int], required=True), 'src_port': FieldInfo(annotation=Union[Port, int], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

3.7.17.5.4 primaite.simulator.network.utils

Functions

convert_bytes_to_megabits	Convert Bytes (file size) to Megabits (data transfer).
convert_megabits_to_bytes	Convert Megabits (data transfer) to Bytes (file size).

3.7.17.5.4.1 primaite.simulator.network.utils.convert_bytes_to_megabits

primaite.simulator.network.utils.**convert_bytes_to_megabits**(*B: int | float*) → float

Convert Bytes (file size) to Megabits (data transfer).

Parameters
B – The file size in Bytes.

Returns
File bits to transfer in Megabits.

3.7.17.5.4.2 primaite.simulator.network.utils.convert_megabits_to_bytes

primaite.simulator.network.utils.**convert_megabits_to_bytes**(*Mbits: int | float*) → float

Convert Megabits (data transfer) to Bytes (file size).

:param Mbits bits to transfer in Megabits. :return: The file size in Bytes.

3.7.17.6 primaite.simulator.system

primaite.simulator.system.applications
primaite.simulator.system.core
primaite.simulator.system.processes
primaite.simulator.system.services
primaite.simulator.system.software

3.7.17.6.1 primaite.simulator.system.applications

primaite.simulator.system.applications.application

3.7.17.6.1.1 primaite.simulator.system.applications.application

Module attributes

RUNNING	The application is running.
CLOSED	The application is closed or not running.
INSTALLING	The application is being installed or updated.

3.7.17.6.1.2 primaite.simulator.system.applications.application.RUNNING

primaite.simulator.system.applications.application.**RUNNING = 1**

The application is running.

3.7.17.6.1.3 primaite.simulator.system.applications.application.CLOSED

primaite.simulator.system.applications.application.**CLOSED = 2**

The application is closed or not running.

3.7.17.6.1.4 primaite.simulator.system.applications.application.INSTALLING

primaite.simulator.system.applications.application.**INSTALLING = 3**

The application is being installed or updated.

Classes

Application	Represents an Application in the simulation environment.
ApplicationOperatingState	Enumeration of Application Operating States.

3.7.17.6.1.5 primaite.simulator.system.applications.application.Application

class primaite.simulator.system.applications.application.**Application**(*, *uuid: str*, *name: str*, *health_state_actual: SoftwareHealthState*, *health_state_visible: SoftwareHealthState*, *criticality: SoftwareCriticality*, *patching_count: int = 0*, *scanning_count: int = 0*, *revealed_to_red: bool = False*, *installing_count: int = 0*, *max_sessions: int = 1*, *tcp: bool = True*, *udp: bool = True*, *ports: Set[Port]*, *operating_state: ApplicationOperatingState*, *execution_control_status: str*, *num_executions: int = 0*, *groups: Set[str] = {}*, ***kwargs*)

Bases: *IOSoftware*

Represents an Application in the simulation environment.

Applications are user-facing programs that may perform input/output operations.

Methods

apply_action	Applies a list of actions to the Application.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the software.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
receive	Receives a payload from the SessionManager.
reset_component_for_episode	Resets the Application component for a new episode.
schema	
schema_json	
send	Sends a payload to the SessionManager.
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
operating_state	The current operating state of the Application.
execution_control_status	Control status of the application's execution.
num_executions	The number of times the application has been executed.
groups	The set of groups to which the application belongs.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*) → None

Applies a list of actions to the Application.

Parameters
action – A list of actions to apply.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

criticality: *SoftwareCriticality*

The criticality level of the software.

abstract describe_state() → Dict

Describes the current state of the software.

The specifics of the software's state, including its health, criticality, and any other pertinent information, should be implemented in subclasses.

Returns
A dictionary containing key-value pairs representing the current state of the software.

Return type
Dict

execution_control_status: str

Control status of the application's execution. It could be 'manual' or 'automatic'.

groups: Set[str]

The set of groups to which the application belongs.

health_state_actual: *SoftwareHealthState*

The actual health state of the software.

health_state_visible: *SoftwareHealthState*

The health state of the software visible to the red agent.

installing_count: int

The number of times the software has been installed. Default is 0.

max_sessions: int

The maximum number of sessions that the software can handle simultaneously. Default is 0.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'criticality': FieldInfo(annotation=SoftwareCriticality, required=True), 'execution_control_status': FieldInfo(annotation=str, required=True), 'groups': FieldInfo(annotation=Set[str], required=False, default=set()), 'health_state_actual': FieldInfo(annotation=SoftwareHealthState, required=True), 'health_state_visible': FieldInfo(annotation=SoftwareHealthState, required=True), 'installing_count': FieldInfo(annotation=int, required=False, default=0), 'max_sessions': FieldInfo(annotation=int, required=False, default=1), 'name': FieldInfo(annotation=str, required=True), 'num_executions': FieldInfo(annotation=int, required=False, default=0), 'operating_state': FieldInfo(annotation=ApplicationOperatingState, required=True), 'patching_count': FieldInfo(annotation=int, required=False, default=0), 'ports': FieldInfo(annotation=Set[Port], required=True), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'scanning_count': FieldInfo(annotation=int, required=False, default=0), 'tcp': FieldInfo(annotation=bool, required=False, default=True), 'udp': FieldInfo(annotation=bool, required=False, default=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str

The name of the software.

num_executions: int

The number of times the application has been executed. Default is 0.

operating_state: *ApplicationOperatingState*

The current operating state of the Application.

patching_count: int

The count of patches applied to the software, defaults to 0.

ports: Set[*Port*]

The set of ports to which the software is connected.

receive(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Receives a payload from the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters
payload – The payload to receive.

Returns
True if successful, False otherwise.

reset_component_for_episode(*episode: int*)

Resets the Application component for a new episode.

This method ensures the Application is ready for a new episode, including resetting any stateful properties or statistics, and clearing any message queues.

revealed_to_red: bool

Indicates if the software has been revealed to red agent, defaults is False.

scanning_count: int

The count of times the software has been scanned, defaults to 0.

send(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Sends a payload to the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters
payload – The payload to send.

Returns
True if successful, False otherwise.

tcp: bool

Indicates if the software uses TCP protocol for communication. Default is True.

udp: bool

Indicates if the software uses UDP protocol for communication. Default is True.

uuid: str

The component UUID.

3.7.17.6.1.6 primaite.simulator.system.applications.application.ApplicationOperatingState

class primaite.simulator.system.applications.application.**ApplicationOperatingState**(*value*)

Bases: Enum

Enumeration of Application Operating States.

3.7.17.6.2 primaite.simulator.system.core

primaite.simulator.system.core.packet_capture	
primaite.simulator.system.core.session_manager	
primaite.simulator.system.core.software_manager	
primaite.simulator.system.core.sys_log	

3.7.17.6.2.1 primaite.simulator.system.core.packet_capture

Classes

PacketCapture	Represents a PacketCapture component on a Node in the simulation environment.

3.7.17.6.2.2 primaite.simulator.system.core.packet_capture.PacketCapture

class primaite.simulator.system.core.packet_capture.**PacketCapture**(*hostname: str*, *ip_address: str | None = None*, *switch_port_number: int | None = None*)

Bases: object

Represents a PacketCapture component on a Node in the simulation environment.

PacketCapture is a service that logs Frames as json strings; It's Wireshark for PrimAITE.

The PCAPs are logged to: <simulation output directory>/<hostname>/<hostname>_<ip address>_pcap.log

Methods

capture	Capture a Frame and log it.

Attributes

hostname	The hostname for which PCAP logs are being recorded.
ip_address	The IP address associated with the PCAP logs.
switch_port_number	The SwitchPort number.

__init__(*hostname: str*, *ip_address: str | None = None*, *switch_port_number: int | None = None*)

Initialize the PacketCapture process.

Parameters

- **hostname** – The hostname for which PCAP logs are being recorded.
- **ip_address** – The IP address associated with the PCAP logs.

capture(*frame*)

Capture a Frame and log it.

Parameters

frame – The PCAP frame to capture.

hostname: str

The hostname for which PCAP logs are being recorded.

ip_address: str

The IP address associated with the PCAP logs.

switch_port_number

The SwitchPort number.

3.7.17.6.2.3 primaite.simulator.system.core.session_manager

Classes

Session	Models a network session.
SessionManager	Manages network sessions, including session creation, lookup, and communication with other components.

3.7.17.6.2.4 primaite.simulator.system.core.session_manager.Session

class primaite.simulator.system.core.session_manager.**Session**(*, *uuid: str*, *protocol:* IPProtocol, *src_ip: IPv4Address*, *dst_ip: IPv4Address*, *src_port:* Port | *None*, *dst_port:* Port | *None*, *connected: bool = False*, ***kwargs*)

Bases: *SimComponent*

Models a network session.

Encapsulates information related to communication between two network endpoints, including the protocol, source and destination IPs and ports.

Parameters

- **protocol** – The IP protocol used in the session.
- **src_ip** – The source IP address.
- **dst_ip** – The destination IP address.
- **src_port** – The source port number (optional).
- **dst_port** – The destination port number (optional).
- **connected** – A flag indicating whether the session is connected.

Methods

apply_action	Apply an action to a simulation component.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the session as a dictionary.
dict	
from_orm	
from_session_key	Create a Session instance from a session key tuple.
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Reset this component to its original state for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
protocol	
src_ip	
dst_ip	
src_port	
dst_port	
connected	

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*, *context: Dict = {}*) → None

Apply an action to a simulation component. Action data is passed in as a 'namespaced' list of strings.

If the list only has one element, the action is intended to be applied directly to this object. If the list has multiple entries, the action is passed to the child of this object specified by the first one or two entries. This is essentially a namespace.

For example, ["turn_on",] is meant to apply an action of 'turn on' to this component.

However, ["services", "email_client", "turn_on"] is meant to 'turn on' this component's email client service.

Parameters

action (*List[str]*) – List describing the action to apply to this object.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning “Deprecated”
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`describe_state`() → Dict

Describes the current state of the session as a dictionary.

Returns
A dictionary containing the current state of the session.

`classmethod from_session_key`(*session_key: Tuple[*IPProtocol, *IPv4Address, IPv4Address,* Port | *None,* Port | *None]*) → *Session*

Create a Session instance from a session key tuple.

Parameters
`session_key` – Tuple containing the session details.

Returns
A Session instance.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct`(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = ‘allow’* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'connected': FieldInfo(annotation=bool, required=False, default=False), 'dst_ip': FieldInfo(annotation=IPv4Address, required=True), 'dst_port': FieldInfo(annotation=Union[Port, NoneType], required=True), 'protocol': FieldInfo(annotation=IPProtocol, required=True), 'src_ip': FieldInfo(annotation=IPv4Address, required=True), 'src_port': FieldInfo(annotation=Union[Port, NoneType], required=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

reset_component_for_episode(*episode: int*)

Reset this component to its original state for a new episode.

Override this method with anything that needs to happen within the component for it to be reset.

uuid: str

The component UUID.

3.7.17.6.2.5 primaite.simulator.system.core.session_manager.SessionManager

class primaite.simulator.system.core.session_manager.**SessionManager**(*sys_log:* SysLog, *arp_cache:* ARPCache)

Bases: `object`

Manages network sessions, including session creation, lookup, and communication with other components.

Parameters

- **sys_log** – A reference to the system log component.
- **arp_cache** – A reference to the ARP cache component.

Methods

describe_state	Describes the current state of the session manager as a dictionary.
receive_payload_from_nic	Receive a Frame from the NIC.
receive_payload_from_software_manager	Receive a payload from the SoftwareManager.
send_payload_to_nic	Send a payload across the Network.
send_payload_to_software_manager	Send a payload to the software manager.

__init__(*sys_log:* SysLog, *arp_cache:* ARPCache)

describe_state() → Dict

Describes the current state of the session manager as a dictionary.

Returns
A dictionary containing the current state of the session manager.

receive_payload_from_nic(*frame:* Frame)

Receive a Frame from the NIC.

Extract the session key using the _get_session_key method, and forward the payload to the appropriate session. If the session does not exist, a new one is created.

Parameters
frame – The frame being received.

receive_payload_from_software_manager(*payload: Any*, *session_id: int | None = None*)

Receive a payload from the SoftwareManager.

If no session_id, a Session is established. Once established, the payload is sent to `send_payload_to_nic`.

Parameters

- **payload** – The payload to be sent.
- **session_id** – The Session ID the payload is to originate from. Optional. If None, one will be created.

send_payload_to_nic(*payload: Any*, *session_id: int*)

Send a payload across the Network.

Takes a payload and a session_id. Builds a Frame and sends it across the network via a NIC.

Parameters

- **payload** – The payload to be sent.
- **session_id** – The Session ID the payload originates from

send_payload_to_software_manager(*payload: Any*, *session_id: int*)

Send a payload to the software manager.

Parameters

- **payload** – The payload to be sent.
- **session_id** – The Session ID the payload originates from.

3.7.17.6.2.6 primaite.simulator.system.core.software_manager

Classes

SoftwareManager	A class that manages all running Services and Applications on a Node and facilitates their communication.

3.7.17.6.2.7 primaite.simulator.system.core.software_manager.SoftwareManager

class primaite.simulator.system.core.software_manager.**SoftwareManager**(*session_manager:* SessionManager, *sys_log:* SysLog)

Bases: `object`

A class that manages all running Services and Applications on a Node and facilitates their communication.

Methods

add_application	Add an Application to the manager.
add_service	Add a Service to the manager.
receive_payload_from_session_manger	Receive a payload from the SessionManager and forward it to the corresponding service or application.
send_internal_payload	Send a payload to a specific service or application.
send_payload_to_session_manger	Send a payload to the SessionManager.

__init__(*session_manager:* SessionManager, *sys_log:* SysLog)

Initialize a new instance of SoftwareManager.

Parameters

session_manager – The session manager handling network communications.

add_application(*name: str*, *application:* Application, *port:* Port, *protocol:* IPProtocol)

Add an Application to the manager.

Parameters

- **name** – The name of the application.
- **application** – The application instance.

- **port** – The port used by the application.
- **protocol** – The network protocol used by the application.

add_service(*name: str*, *service:* Service, *port:* Port, *protocol:* IPProtocol)

Add a Service to the manager.

Parameters

- **name** – The name of the service.
- **service** – The service instance.
- **port** – The port used by the service.
- **protocol** – The network protocol used by the service.

receive_payload_from_session_manger(*payload: Any*, *session:* Session)

Receive a payload from the SessionManager and forward it to the corresponding service or application.

Parameters

- **payload** – The payload being received.
- **session** – The transport session the payload originates from.

send_internal_payload(*target_software: str*, *target_software_type:* SoftwareType, *payload: Any*)

Send a payload to a specific service or application.

Parameters

- **target_software** – The name of the target service or application.
- **target_software_type** – The type of software (Service, Application, Process).
- **payload** – The data to be sent.
- **receiver_type** – The type of the target, either 'service' or 'application'.

send_payload_to_session_manger(*payload: Any*, *session_id: int | None = None*)

Send a payload to the SessionManager.

Parameters

- **payload** – The payload to be sent.
- **session_id** – The Session ID the payload is to originate from. Optional.

3.7.17.6.2.8 primaite.simulator.system.core.sys_log

Classes

SysLog	A SysLog class is a simple logger dedicated to managing and writing system logs for a Node.

3.7.17.6.2.9 primaite.simulator.system.core.sys_log.SysLog

class primaite.simulator.system.core.sys_log.**SysLog**(*hostname: str*)

Bases: `object`

A SysLog class is a simple logger dedicated to managing and writing system logs for a Node.

Each log message is written to a file located at: <simulation output directory>/<hostname>/<hostname>_sys.log

Methods

critical	Logs a message with the CRITICAL level.
debug	Logs a message with the DEBUG level.
error	Logs a message with the ERROR level.
info	Logs a message with the INFO level.
warning	Logs a message with the WARNING level.

__init__(*hostname: str*)

Constructs a SysLog instance for a given hostname.

Parameters
hostname – The hostname associated with the system logs being recorded.

critical(*msg: str*)

Logs a message with the CRITICAL level.

Parameters
msg – The message to be logged.

debug(*msg: str*)

Logs a message with the DEBUG level.

Parameters
msg – The message to be logged.

error(*msg: str*)

Logs a message with the ERROR level.

Parameters
msg – The message to be logged.

info(*msg: str*)

Logs a message with the INFO level.

Parameters
msg – The message to be logged.

warning(*msg: str*)

Logs a message with the WARNING level.

Parameters
msg – The message to be logged.

3.7.17.6.3 primaite.simulator.system.processes

primaite.simulator.system.processes.process

3.7.17.6.3.1 primaite.simulator.system.processes.process

Classes

Process	Represents a Process, a program in execution, in the simulation environment.
ProcessOperatingState	Enumeration of Process Operating States.

3.7.17.6.3.2 primaite.simulator.system.processes.process.Process

class primaite.simulator.system.processes.process.**Process**(*, *uuid: str*, *name: str*, *health_state_actual:* SoftwareHealthState, *health_state_visible:* SoftwareHealthState, *criticality:* SoftwareCriticality, *patching_count: int = 0*, *scanning_count: int = 0*, *revealed_to_red: bool = False*, *operating_state:* ProcessOperatingState, ***kwargs*)

Bases: *Software*

Represents a Process, a program in execution, in the simulation environment.

Processes are executed by a Node and do not have the ability to performing input/output operations.

Methods

apply_action	Applies a list of actions to the software.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the software.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Resets the software component for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
operating_state	The current operating state of the Process.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*) → None

Applies a list of actions to the software.

The specifics of how these actions are applied should be implemented in subclasses.

Parameters
action (*List[str]*) – A list of actions to apply.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

criticality: *SoftwareCriticality*

The criticality level of the software.

abstract describe_state() → Dict

Describes the current state of the software.

The specifics of the software's state, including its health, criticality, and any other pertinent information, should be implemented in subclasses.

Returns
A dictionary containing key-value pairs representing the current state of the software.

Return type
Dict

health_state_actual: *SoftwareHealthState*

The actual health state of the software.

health_state_visible: *SoftwareHealthState*

The health state of the software visible to the red agent.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'criticality': FieldInfo(annotation=SoftwareCriticality, required=True), 'health_state_actual': FieldInfo(annotation=SoftwareHealthState, required=True), 'health_state_visible': FieldInfo(annotation=SoftwareHealthState, required=True), 'name': FieldInfo(annotation=str, required=True), 'operating_state': FieldInfo(annotation=ProcessOperatingState, required=True), 'patching_count': FieldInfo(annotation=int, required=False, default=0), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'scanning_count': FieldInfo(annotation=int, required=False, default=0), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, ***, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str
The name of the software.

operating_state: *ProcessOperatingState*
The current operating state of the Process.

patching_count: int
The count of patches applied to the software, defaults to 0.

reset_component_for_episode(*episode: int*)
Resets the software component for a new episode.

This method should ensure the software is ready for a new episode, including resetting any stateful properties or statistics, and clearing any message queues. The specifics of what constitutes a "reset" should be implemented in subclasses.

revealed_to_red: bool
Indicates if the software has been revealed to red agent, defaults is False.

scanning_count: int
The count of times the software has been scanned, defaults to 0.

uuid: str
The component UUID.

3.7.17.6.3.3 primaite.simulator.system.processes.process.ProcessOperatingState

class primaite.simulator.system.processes.process.**ProcessOperatingState**(*value*)

Bases: Enum

Enumeration of Process Operating States.

Attributes

RUNNING	The process is running.
PAUSED	The process is temporarily paused.

PAUSED = 2

The process is temporarily paused.

RUNNING = 1

The process is running.

3.7.17.6.4 primaite.simulator.system.services

primaite.simulator.system.services.dns_service
primaite.simulator.system.services.service

3.7.17.6.4.1 primaite.simulator.system.services.dns_service

Classes

DNSService

3.7.17.6.4.2 primaite.simulator.system.services.dns_service.DNSService

class primaite.simulator.system.services.dns_service.**DNSService**(*, *uuid: str*, *name: str*, *health_state_actual: SoftwareHealthState*, *health_state_visible: SoftwareHealthState*, *criticality: SoftwareCriticality*, *patching_count: int = 0*, *scanning_count: int = 0*, *revealed_to_red: bool = False*, *installing_count: int = 0*, *max_sessions: int = 1*, *tcp: bool = True*, *udp: bool = True*, *ports: Set[Port]*, *operating_state: ServiceOperatingState*, ***kwargs*)

Bases: *Service*

Methods

apply_action	Applies a list of actions to the Service.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the software.
dict	
from_orm	
get_install	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
receive	Receives a payload from the SessionManager.
reset_component_for_episode	Resets the Service component for a new episode.
schema	
schema_json	
send	Sends a payload to the SessionManager.
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*) → None

Applies a list of actions to the Service.

Parameters
action – A list of actions to apply.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

criticality: *SoftwareCriticality*

The criticality level of the software.

describe_state() → Dict

Describes the current state of the software.

The specifics of the software's state, including its health, criticality, and any other pertinent information, should be implemented in subclasses.

Returns
A dictionary containing key-value pairs representing the current state of the software.

Return type
Dict

health_state_actual: *SoftwareHealthState*

The actual health state of the software.

health_state_visible: *SoftwareHealthState*

The health state of the software visible to the red agent.

installing_count: int

The number of times the software has been installed. Default is 0.

max_sessions: int

The maximum number of sessions that the software can handle simultaneously. Default is 0.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(***, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'criticality': FieldInfo(annotation=SoftwareCriticality, required=True), 'health_state_actual': FieldInfo(annotation=SoftwareHealthState, required=True), 'health_state_visible': FieldInfo(annotation=SoftwareHealthState, required=True), 'installing_count': FieldInfo(annotation=int, required=False, default=0), 'max_sessions': FieldInfo(annotation=int, required=False, default=1), 'name': FieldInfo(annotation=str, required=True), 'operating_state': FieldInfo(annotation=ServiceOperatingState, required=True), 'patching_count': FieldInfo(annotation=int, required=False, default=0), 'ports': FieldInfo(annotation=Set[Port], required=True), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'scanning_count': FieldInfo(annotation=int, required=False, default=0), 'tcp': FieldInfo(annotation=bool, required=False, default=True), 'udp': FieldInfo(annotation=bool, required=False, default=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str

The name of the software.

operating_state: *ServiceOperatingState*

The current operating state of the Service.

patching_count: int

The count of patches applied to the software, defaults to 0.

ports: Set[*Port*]

The set of ports to which the software is connected.

receive(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Receives a payload from the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters
payload – The payload to receive.

Returns
True if successful, False otherwise.

reset_component_for_episode(*episode: int*)

Resets the Service component for a new episode.

This method ensures the Service is ready for a new episode, including resetting any stateful properties or statistics, and clearing any message queues.

revealed_to_red: bool

Indicates if the software has been revealed to red agent, defaults is False.

scanning_count: int

The count of times the software has been scanned, defaults to 0.

send(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Sends a payload to the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters
payload – The payload to send.

Returns
True if successful, False otherwise.

tcp: bool

Indicates if the software uses TCP protocol for communication. Default is True.

udp: bool

Indicates if the software uses UDP protocol for communication. Default is True.

uuid: str

The component UUID.

3.7.17.6.4.3 primaite.simulator.system.services.service

Classes

Service	Represents a Service in the simulation environment.
ServiceOperatingState	Enumeration of Service Operating States.

3.7.17.6.4.4 primaite.simulator.system.services.service.Service

class primaite.simulator.system.services.service.**Service**(*, *uuid: str*, *name: str*, *health_state_actual: SoftwareHealthState*, *health_state_visible: SoftwareHealthState*, *criticality: SoftwareCriticality*, *patching_count: int = 0*, *scanning_count: int = 0*, *revealed_to_red: bool = False*, *installing_count: int = 0*, *max_sessions: int = 1*, *tcp: bool = True*, *udp: bool = True*, *ports: Set[Port]*, *operating_state: ServiceOperatingState*, ***kwargs*)

Bases: *IOSoftware*

Represents a Service in the simulation environment.

Services are programs that run in the background and may perform input/output operations.

Methods

apply_action	Applies a list of actions to the Service.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the software.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
receive	Receives a payload from the SessionManager.
reset_component_for_episode	Resets the Service component for a new episode.
schema	
schema_json	
send	Sends a payload to the SessionManager.
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
operating_state	The current operating state of the Service.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*) → None

Applies a list of actions to the Service.

Parameters
action – A list of actions to apply.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

criticality: *SoftwareCriticality*

The criticality level of the software.

abstract describe_state() → Dict

Describes the current state of the software.

The specifics of the software's state, including its health, criticality, and any other pertinent information, should be implemented in subclasses.

Returns
A dictionary containing key-value pairs representing the current state of the software.

Return type
Dict

health_state_actual: *SoftwareHealthState*

The actual health state of the software.

health_state_visible: *SoftwareHealthState*

The health state of the software visible to the red agent.

installing_count: int

The number of times the software has been installed. Default is 0.

max_sessions: int

The maximum number of sessions that the software can handle simultaneously. Default is 0.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'criticality': FieldInfo(annotation=SoftwareCriticality, required=True), 'health_state_actual': FieldInfo(annotation=SoftwareHealthState, required=True), 'health_state_visible': FieldInfo(annotation=SoftwareHealthState, required=True), 'installing_count': FieldInfo(annotation=int, required=False, default=0), 'max_sessions': FieldInfo(annotation=int, required=False, default=1), 'name': FieldInfo(annotation=str, required=True), 'operating_state': FieldInfo(annotation=ServiceOperatingState, required=True), 'patching_count': FieldInfo(annotation=int, required=False, default=0), 'ports': FieldInfo(annotation=Set[Port], required=True), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'scanning_count': FieldInfo(annotation=int, required=False, default=0), 'tcp': FieldInfo(annotation=bool, required=False, default=True), 'udp': FieldInfo(annotation=bool, required=False, default=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray, *, strict: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str

The name of the software.

operating_state: *ServiceOperatingState*

The current operating state of the Service.

patching_count: int

The count of patches applied to the software, defaults to 0.

ports: Set[*Port*]

The set of ports to which the software is connected.

receive(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Receives a payload from the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters
payload – The payload to receive.

Returns
True if successful, False otherwise.

reset_component_for_episode(*episode: int*)

Resets the Service component for a new episode.

This method ensures the Service is ready for a new episode, including resetting any stateful properties or statistics, and clearing any message queues.

revealed_to_red: bool

Indicates if the software has been revealed to red agent, defaults is False.

scanning_count: int

The count of times the software has been scanned, defaults to 0.

send(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Sends a payload to the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters
payload – The payload to send.

Returns
True if successful, False otherwise.

tcp: bool

Indicates if the software uses TCP protocol for communication. Default is True.

udp: bool

Indicates if the software uses UDP protocol for communication. Default is True.

uuid: str

The component UUID.

3.7.17.6.4.5 primaite.simulator.system.services.service.ServiceOperatingState

class primaite.simulator.system.services.service.**ServiceOperatingState**(*value*)

Bases: Enum

Enumeration of Service Operating States.

Attributes

RUNNING	The service is currently running.
STOPPED	The service is not running.
INSTALLING	The service is being installed or updated.
RESTARTING	The service is in the process of restarting.
PAUSED	The service is temporarily paused.
DISABLED	The service is disabled and cannot be started.

DISABLED = 6

The service is disabled and cannot be started.

INSTALLING = 3

The service is being installed or updated.

PAUSED = 5

The service is temporarily paused.

RESTARTING = 4

The service is in the process of restarting.

RUNNING = 1

The service is currently running.

STOPPED = 2

The service is not running.

3.7.17.6.5 primaite.simulator.system.software

Classes

IOSoftware	Represents software in a simulator environment that is capable of input/output operations.
Software	A base class representing software in a simulator environment.
SoftwareCriticality	Enumeration of Software Criticality Levels.
SoftwareHealthState	Enumeration of the Software Health States.
SoftwareType	An enumeration representing the different types of software within a simulated environment.

3.7.17.6.5.1 primaite.simulator.system.software.IOSoftware

class primaite.simulator.system.software.**IOSoftware**(*, *uuid: str*, *name: str*, *health_state_actual:* SoftwareHealthState, *health_state_visible:* SoftwareHealthState, *criticality:* SoftwareCriticality, *patching_count: int = 0*, *scanning_count: int = 0*, *revealed_to_red: bool = False*, *installing_count: int = 0*, *max_sessions: int = 1*, *tcp: bool = True*, *udp: bool = True*, *ports: Set[*Port*]*, ***kwargs*)

Bases: *Software*

Represents software in a simulator environment that is capable of input/output operations.

This base class is meant to be sub-classed by Application and Service classes. It provides the blueprint for Applications and Services that can receive payloads from a Node's SessionManager (corresponding to layer 5 in the OSI Model), process them according to their internals, and send a response payload back to the SessionManager if required.

Methods

apply_action	Applies a list of actions to the software.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the software.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
receive	Receives a payload from the SessionManager.
reset_component_for_episode	Resets the software component for a new episode.
schema	
schema_json	
send	Sends a payload to the SessionManager.
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
installing_count	The number of times the software has been installed.
max_sessions	The maximum number of sessions that the software can handle simultaneously.
tcp	Indicates if the software uses TCP protocol for communication.
udp	Indicates if the software uses UDP protocol for communication.
ports	The set of ports to which the software is connected.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*) → None

Applies a list of actions to the software.

The specifics of how these actions are applied should be implemented in subclasses.

Parameters
action (*List[str]*) – A list of actions to apply.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

`criticality: SoftwareCriticality`

The criticality level of the software.

`abstract describe_state() → Dict`

Describes the current state of the software.

The specifics of the software's state, including its health, criticality, and any other pertinent information, should be implemented in subclasses.

Returns
A dictionary containing key-value pairs representing the current state of the software.

Return type
Dict

`health_state_actual: SoftwareHealthState`

The actual health state of the software.

`health_state_visible: SoftwareHealthState`

The health state of the software visible to the red agent.

`installing_count: int`

The number of times the software has been installed. Default is 0.

`max_sessions: int`

The maximum number of sessions that the software can handle simultaneously. Default is 0.

`property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]`

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}`

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

`classmethod model_construct(_fields_set: set[str] | None = None, **values: Any) → Model`

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'criticality': FieldInfo(annotation=SoftwareCriticality, required=True), 'health_state_actual': FieldInfo(annotation=SoftwareHealthState, required=True), 'health_state_visible': FieldInfo(annotation=SoftwareHealthState, required=True), 'installing_count': FieldInfo(annotation=int, required=False, default=0), 'max_sessions': FieldInfo(annotation=int, required=False, default=1), 'name': FieldInfo(annotation=str, required=True), 'patching_count': FieldInfo(annotation=int, required=False, default=0), 'ports': FieldInfo(annotation=Set[Port], required=True), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'scanning_count': FieldInfo(annotation=int, required=False, default=0), 'tcp': FieldInfo(annotation=bool, required=False, default=True), 'udp': FieldInfo(annotation=bool, required=False, default=True), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:
String representing the new class where *params* are passed to *cls* as type variables.

Raises:
TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:
force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

name: str

The name of the software.

patching_count: int

The count of patches applied to the software, defaults to 0.

ports: Set[*Port*]

The set of ports to which the software is connected.

receive(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Receives a payload from the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters

- **payload** – The payload to receive.
- **session_id** – The identifier of the session that the payload is associated with.
- **kwargs** – Additional keyword arguments specific to the implementation.

Returns

True if the payload was successfully received and processed, False otherwise.

reset_component_for_episode(*episode: int*)

Resets the software component for a new episode.

This method should ensure the software is ready for a new episode, including resetting any stateful properties or statistics, and clearing any message queues. The specifics of what constitutes a "reset" should be implemented in subclasses.

revealed_to_red: bool

Indicates if the software has been revealed to red agent, defaults is False.

scanning_count: int

The count of times the software has been scanned, defaults to 0.

send(*payload: Any*, *session_id: str*, ***kwargs*) → bool

Sends a payload to the SessionManager.

The specifics of how the payload is processed and whether a response payload is generated should be implemented in subclasses.

Parameters

- **payload** – The payload to send.
- **session_id** – The identifier of the session that the payload is associated with.
- **kwargs** – Additional keyword arguments specific to the implementation.

Returns

True if the payload was successfully sent, False otherwise.

tcp: bool

Indicates if the software uses TCP protocol for communication. Default is True.

udp: bool

Indicates if the software uses UDP protocol for communication. Default is True.

uuid: str

The component UUID.

3.7.17.6.5.2 primaite.simulator.system.software.Software

class primaite.simulator.system.software.**Software**(*, *uuid: str*, *name: str*, *health_state_actual: SoftwareHealthState*, *health_state_visible: SoftwareHealthState*, *criticality: SoftwareCriticality*, *patching_count: int = 0*, *scanning_count: int = 0*, *revealed_to_red: bool = False*, ***kwargs*)

Bases: *SimComponent*

A base class representing software in a simulator environment.

This class is intended to be subclassed by specific types of software entities. It outlines the fundamental attributes and behaviors expected of any software in the simulation.

Methods

apply_action	Applies a list of actions to the software.
apply_timestep	Apply a timestep evolution to this component.
construct	
copy	Returns a copy of the model.
describe_state	Describes the current state of the software.
dict	
from_orm	
json	
model_construct	Creates a new instance of the *Model* class with validated data.
model_copy	Returns a copy of the model.
model_dump	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump
model_dump_json	Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json
model_json_schema	Generates a JSON schema for a model class.
model_parametrized_name	Compute the class name for parametrizations of generic classes.
model_post_init	Override this method to perform additional initialization after *__init__* and *model_construct*.
model_rebuild	Try to rebuild the pydantic-core schema for the model.
model_validate	Validate a pydantic model instance.
model_validate_json	Validate the given JSON data against the Pydantic model.
parse_file	
parse_obj	
parse_raw	
reset_component_for_episode	Resets the software component for a new episode.
schema	
schema_json	
update_forward_refs	
validate	

Attributes

model_computed_fields	Get the computed fields of this model instance.
model_config	Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.
model_extra	Get extra fields set during validation.
model_fields	Metadata about the fields defined on the model, mapping of field names to [*Field-Info*][pydantic.fields.FieldInfo].
model_fields_set	Returns the set of fields that have been set on this model instance.
name	The name of the software.
health_state_actual	The actual health state of the software.
health_state_visible	The health state of the software visible to the red agent.
criticality	The criticality level of the software.
patching_count	The count of patches applied to the software, defaults to 0.
scanning_count	The count of times the software has been scanned, defaults to 0.
revealed_to_red	Indicates if the software has been revealed to red agent, defaults is False.

__init__(***kwargs*)

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

__init__ uses *__pydantic_self__* instead of the more common *self* for the first arg to allow *self* as a field name.

apply_action(*action: List[str]*) → None

Applies a list of actions to the software.

The specifics of how these actions are applied should be implemented in subclasses.

Parameters
action (*List[str]*) – A list of actions to apply.

apply_timestep(*timestep: int*) → None

Apply a timestep evolution to this component.

Override this method with anything that happens automatically in the component such as scheduled restarts or sending data.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping
specifying which fields to include in the copied model.

exclude: Optional set or mapping
specifying which fields to exclude in the copied model.

update: Optional dictionary of field-value pairs to override field values
in the copied model.

deep: If True, the values of fields that are Pydantic models will be deep copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

criticality: *SoftwareCriticality*

The criticality level of the software.

abstract describe_state() → Dict

Describes the current state of the software.

The specifics of the software's state, including its health, criticality, and any other pertinent information, should be implemented in subclasses.

Returns
A dictionary containing key-value pairs representing the current state of the software.

Return type
Dict

health_state_actual: *SoftwareHealthState*

The actual health state of the software.

health_state_visible: *SoftwareHealthState*

The health state of the software visible to the red agent.

property model_computed_fields: dict[str, pydantic.fields.ComputedFieldInfo]

Get the computed fields of this model instance.

Returns:
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'extra': 'allow'}

Configure pydantic to allow arbitrary types and to let the instance have attributes not present in model.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that are unset or None from the output. exclude_defaults: Whether to exclude fields that are set to their default value from the output. exclude_none: Whether to exclude fields that have a value of *None* from the output. round_trip: Whether to enable serialization and deserialization round-trip support. warnings: Whether to log warnings when invalid fields are encountered.

Returns:
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:
indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. Can take either a string or set of strings. exclude: Field(s) to exclude from the JSON output. Can take either a string or set of strings. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that have the default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: Whether to use serialization/deserialization between JSON and class instance. warnings: Whether to show any warnings that occurred during serialization.

Returns:
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields: ClassVar[dict[str, FieldInfo]] = {'criticality': FieldInfo(annotation=SoftwareCriticality, required=True), 'health_state_actual': FieldInfo(annotation=SoftwareHealthState, required=True), 'health_state_visible': FieldInfo(annotation=SoftwareHealthState, required=True), 'name': FieldInfo(annotation=str, required=True), 'patching_count': FieldInfo(annotation=int, required=False, default=0), 'revealed_to_red': FieldInfo(annotation=bool, required=False, default=False), 'scanning_count': FieldInfo(annotation=int, required=False, default=0), 'uuid': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, ass a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class
Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to raise an exception on invalid fields. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

name: str

The name of the software.

patching_count: int

The count of patches applied to the software, defaults to 0.

reset_component_for_episode(*episode: int*)

Resets the software component for a new episode.

This method should ensure the software is ready for a new episode, including resetting any stateful properties or statistics, and clearing any message queues. The specifics of what constitutes a "reset" should be implemented in subclasses.

revealed_to_red: bool

Indicates if the software has been revealed to red agent, defaults is False.

scanning_count: int

The count of times the software has been scanned, defaults to 0.

uuid: str

The component UUID.

3.7.17.6.5.3 primaite.simulator.system.software.SoftwareCriticality

class primaite.simulator.system.software.**SoftwareCriticality**(*value*)

Bases: Enum

Enumeration of Software Criticality Levels.

Attributes

LOWEST	The lowest level of criticality.
LOW	A low level of criticality.
MEDIUM	A medium level of criticality.
HIGH	A high level of criticality.
HIGHEST	The highest level of criticality.

HIGH = 4

A high level of criticality.

HIGHEST = 5

The highest level of criticality.

LOW = 2

A low level of criticality.

LOWEST = 1

The lowest level of criticality.

MEDIUM = 3

A medium level of criticality.

3.7.17.6.5.4 primaite.simulator.system.software.SoftwareHealthState

class primaite.simulator.system.software.**SoftwareHealthState**(*value*)

Bases: Enum

Enumeration of the Software Health States.

Attributes

GOOD	The software is in a good and healthy condition.
COMPROMISED	The software's security has been compromised.
OVERWHELMED	he software is overwhelmed and not functioning properly.
PATCHING	The software is undergoing patching or updates.

COMPROMISED = 2

The software's security has been compromised.

GOOD = 1

The software is in a good and healthy condition.

OVERWHELMED = 3

he software is overwhelmed and not functioning properly.

PATCHING = 4

The software is undergoing patching or updates.

3.7.17.6.5.5 primaite.simulator.system.software.SoftwareType

class primaite.simulator.system.software.**SoftwareType**(*value*)

Bases: Enum

An enumeration representing the different types of software within a simulated environment.

Members: - APPLICATION: User-facing programs that may perform input/output operations. - SERVICE: Represents programs that run in the background and may perform input/output operations. - PROCESS: Software executed by a Node that does not have the ability to performing input/output operations.

Attributes

APPLICATION	User-facing software that may perform input/output operations.
SERVICE	Software that runs in the background and may perform input/output operations.
PROCESS	Software executed by a Node that does not have the ability to performing input/output operations.

APPLICATION = 1

User-facing software that may perform input/output operations.

PROCESS = 3

Software executed by a Node that does not have the ability to performing input/output operations.

SERVICE = 2

Software that runs in the background and may perform input/output operations.

3.7.18 primaite.transactions

Record data of the system's state and agent's observations and actions.

primaite.transactions.transaction	The Transaction class.

3.7.18.1 primaite.transactions.transaction

The Transaction class.

Classes

Transaction	Transaction class.

3.7.18.1.1 primaite.transactions.transaction.Transaction

class primaite.transactions.transaction.**Transaction**(*agent_identifier: AgentIdentifier*, *episode_number: int*, *step_number: int*)

Bases: `object`

Transaction class.

Methods

as_csv_data	Converts the Transaction to a csv data row and provides a header.

Attributes

timestamp	The datetime of the transaction
agent_identifier	The agent identifier
episode_number	The episode number
step_number	The step number
obs_space	The observation space (pre)
obs_space_pre	The observation space before any actions are taken
obs_space_post	The observation space after any actions are taken
reward	The reward value
action_space	The action space invoked by the agent
obs_space_description	The env observation space description

__init__(*agent_identifier: AgentIdentifier*, *episode_number: int*, *step_number: int*) → None

Transaction constructor.

Parameters

- **agent_identifier** – An identifier for the agent in use

- **episode_number** – The episode number
- **step_number** – The step number

action_space: int | None

The action space invoked by the agent

agent_identifier: *AgentIdentifier*

The agent identifier

as_csv_data() → Tuple[List, List]

Converts the Transaction to a csv data row and provides a header.

Returns

A tuple consisting of (header, data).

episode_number: int

The episode number

obs_space: spaces.Space

The observation space (pre)

obs_space_description: List[str] | None

The env observation space description

obs_space_post: 'np.ndarray' | Tuple['np.ndarray'] | None

The observation space after any actions are taken

obs_space_pre: 'np.ndarray' | Tuple['np.ndarray'] | None

The observation space before any actions are taken

reward: float | None

The reward value

step_number: int

The step number

timestamp: datetime

The datetime of the transaction

3.7.19 primaite.utils

Utilities for PrimAITE.

primaite.utils.package_data
primaite.utils.session_metadata_parser
primaite.utils.session_output_reader
primaite.utils.session_output_writer

3.7.19.1 primaite.utils.package_data

Functions

get_file_path	Get PrimAITE package data.

3.7.19.1.1 primaite.utils.package_data.get_file_path

primaite.utils.package_data.**get_file_path**(*path: str*) → Path

Get PrimAITE package data.

Example

```
>>> from primaite.utils.package_data import get_file_path
>>> main_env_config = get_file_path("config/_package_data/training_config_main.yaml
↪")
```

Parameters
path – The path from the primaite root.

Returns
The file path of the package data file.

Raises
FileNotFoundError – When the filepath does not exist.

3.7.19.2 primaite.utils.session_metadata_parser

Functions

parse_session_metadata	Loads a session metadata from the given directory path.

3.7.19.2.1 primaite.utils.session_metadata_parser.parse_session_metadata

primaite.utils.session_metadata_parser.**parse_session_metadata**(*session_path: Path | str*, *dict_only: bool = False*) → Dict[str, Any]

Loads a session metadata from the given directory path.

Parameters

- **session_path** – Directory where the session metadata file is in
- **dict_only** – If dict_only is true, the function will only return the dict contents of session metadata

Returns
Dictionary which has all the session metadata contents

Return type
Dict

Returns
Path where the YAML copy of the training config is dumped into

Return type
str

Returns
Path where the YAML copy of the laydown config is dumped into

Return type
str

3.7.19.3 primaite.utils.session_output_reader

Functions

all_transactions_dict	Read an all transactions csv file and return as a dict.
av_rewards_dict	Read an average rewards per episode csv file and return as a dict.

3.7.19.3.1 primaite.utils.session_output_reader.all_transactions_dict

primaite.utils.session_output_reader.**all_transactions_dict**(*all_transactions_csv_file: str | Path*) → Dict[Tuple[int, int], Dict[str, Any]]

Read an all transactions csv file and return as a dict.

The dict keys are a tuple with the structure (episode, step). The dict values are the remaining columns as a dict.

Parameters
all_transactions_csv_file – The all transactions csv file path.

Returns
The all transactions csv file as a dict.

3.7.19.3.2 primaite.utils.session_output_reader.av_rewards_dict

primaite.utils.session_output_reader.**av_rewards_dict**(*av_rewards_csv_file: str | Path*) → Dict[int, float]

Read an average rewards per episode csv file and return as a dict.

The dictionary keys are the episode number, and the values are the mean reward that episode.

Parameters
av_rewards_csv_file – The average rewards per episode csv file path.

Returns
The average rewards per episode csv as a dict.

3.7.19.4 primaite.utils.session_output_writer

Classes

SessionOutputWriter	A session output writer class.

3.7.19.4.1 primaite.utils.session_output_writer.SessionOutputWriter

class primaite.utils.session_output_writer.**SessionOutputWriter**(*env:* Primaite, *transaction_writer: bool = False*, *learning_session: bool = True*)

Bases: `object`

A session output writer class.

Is used to write session outputs to csv file.

Methods

close	Close the cvs file.
write	Write a row of session data.

__init__(*env:* Primaite, *transaction_writer: bool = False*, *learning_session: bool = True*) → None

Initialise the Session Output Writer.

Parameters

- **env** (Primaite) – PrimAITE gym environment.
- **transaction_writer** (*bool, optional*) – If *true*, this will output a full account of every transaction taken by the agent. If *false* it will output the average reward per episode, defaults to False
- **learning_session** (*bool, optional*) – Set to *true* to indicate that the current session is a training session. This determines the name of the folder which contains the final output csv. Defaults to True

close() → None

Close the cvs file.

write(*data: Tuple | Transaction*) → None

Write a row of session data.

Parameters

data – The row of data to write. Can be a Tuple or an instance of Transaction.

3.8 tests

Module attributes

TEST_CONFIG_ROOT	The tests config root directory.
TEST_ASSETS_ROOT	The tests assets root directory.

3.8.1 tests.TEST_CONFIG_ROOT

tests.**TEST_CONFIG_ROOT: Final[Path] = WindowsPath('C:/Users/ChristopherMcCarthy/source/azure_devops/ma-dev-uk/PrimAITE/tests/config')**

The tests config root directory.

3.8.2 tests.TEST_ASSETS_ROOT

tests.**TEST_ASSETS_ROOT: Final[Path] = WindowsPath('C:/Users/ChristopherMcCarthy/source/azure_devops/ma-dev-uk/PrimAITE/tests/assets')**

The tests assets root directory.

tests.conftest	
tests.integration_tests	
tests.mock_and_patch	
tests.test_acl	Used to tes the ACL functions.
tests.test_active_node	Used to test Active Node functions.
tests.test_full_legacy_config_session	
tests.test_lay_down_config	
tests.test_observation_space	Test env creation and behaviour with different observation spaces.
tests.test_primaite_session	
tests.test_red_random_agent_behaviour	
tests.test_resetting_node	Used to test Active Node functions.
tests.test_reward	
tests.test_seeding_and_deterministic_sessio	
tests.test_service_node	Used to test Service Node functions.
tests.test_session_loading	
tests.test_single_action_space	
tests.test_train_eval_episode_steps	
tests.test_training_config	
tests.unit_tests	

3.8.3 tests.conftest

Functions

temp_primaite_session	Provides a temporary PrimaiteSession instance.
temp_session_path	Get a temp directory session path the test session will output to.

3.8.3.1 tests.conftest.temp_primaite_session

tests.conftest.**temp_primaite_session**(*request*)

Provides a temporary PrimaiteSession instance.

It's temporary as it uses a temporary directory as the session path.

To use this fixture you need to:

- parametrize your test function with:
  - "temp_primaite_session"
  - [[path to training config, path to lay down config]]
- Include the temp_primaite_session fixture as a param in your test

function. - use the temp_primaite_session as a context manager assigning is the name 'session'.

```
from primaite.config.lay_down_config import dos_very_basic_config_path
from primaite.config.training_config import main_training_config_path
@pytest.mark.parametrize(
    "temp_primaite_session",
    [
        [main_training_config_path(), dos_very_basic_config_path()]
    ],
    indirect=True
)
def test_primaite_session(temp_primaite_session):
    with temp_primaite_session as session:
        # Learning outputs are saved in session.learning_path
        session.learn()

        # Evaluation outputs are saved in session.evaluation_path
        session.evaluate()

        # To ensure that all files are written, you must call .close()
        session.close()

        # If you need to inspect any session outputs, it must be done
        # inside the context manager

    # Now that we've exited the context manager, the
    # session.session_path directory and its contents are deleted
```

3.8.3.2 tests.conftest.temp_session_path

tests.conftest.**temp_session_path**() → Path

Get a temp directory session path the test session will output to.

Returns

The session directory path.

Classes

TempPrimaiteSession	A temporary PrimaiteSession class.

3.8.3.3 tests.conftest.TempPrimaiteSession

class tests.conftest.**TempPrimaiteSession**(*training_config_path: str | Path*, *lay_down_config_path: str | Path*)

Bases: *PrimaiteSession*

A temporary PrimaiteSession class.

Uses context manager for deletion of files upon exit.

Methods

close	Closes the agent.
eval_all_transactions_dict	Get the eval all transactions from file.
eval_av_reward_per_episode_dict	Get the eval av reward per episode from file.
evaluate	Evaluate the agent.
learn	Train the agent.
learn_all_transactions_dict	Get the learn all transactions from file.
learn_av_reward_per_episode_dict	Get the learn av reward per episode from file.
metadata_file_as_dict	Read the session_metadata.json file and return as a dict.
setup	Performs the session setup.

Attributes

env	Direct access to the env for ease of testing.

__init__(*training_config_path: str | Path*, *lay_down_config_path: str | Path*)

The PrimaiteSession constructor.

Parameters

- **training_config_path** (*Union[path, str]*) – YAML file containing configurable items defined in *primaite.config.training_config.TrainingConfig*
- **lay_down_config_path** (*Union[path, str]*) – YAML file containing configurable items for generating network laydown.
- **session_path** – directory path of the session to load
- **legacy_training_config** – True if the training config file is a legacy file from PrimAITE < 2.0, otherwise False.
- **legacy_lay_down_config** – True if the lay_down config file is a legacy file from PrimAITE < 2.0, otherwise False.

close() → None

Closes the agent.

property env: *Primaite*

Direct access to the env for ease of testing.

eval_all_transactions_dict() → Dict[Tuple[int, int], Dict[str, Any]]

Get the eval all transactions from file.

eval_av_reward_per_episode_dict() → Dict[int, float]

Get the eval av reward per episode from file.

evaluate(***kwargs: Any*) → None

Evaluate the agent.

Parameters

kwargs – Any agent-framework specific key word args.

learn(***kwargs: Any*) → None

Train the agent.

Parameters

kwargs – Any agent-framework specific key word args.

learn_all_transactions_dict() → Dict[Tuple[int, int], Dict[str, Any]]

Get the learn all transactions from file.

learn_av_reward_per_episode_dict() → Dict[int, float]

Get the learn av reward per episode from file.

metadata_file_as_dict() → Dict[str, Any]

Read the session_metadata.json file and return as a dict.

setup() → None

Performs the session setup.

3.8.4 tests.integration_tests

tests.integration_tests.component_creation

tests.integration_tests.network

3.8.4.1 tests.integration_tests.component_creation

tests.integration_tests.
component_creation.test_permission_system

3.8.4.1.1 tests.integration_tests.component_creation.test_permission_system

Functions

test_group_action_validation	Check that actions are denied when an unauthorised request is made.
test_hierarchical_action_with_validation	Check that validation works with sub-objects.

3.8.4.1.1.1 tests.integration_tests.component_creation.test_permission_system.test_group_action_validation

tests.integration_tests.component_creation.test_permission_system.**test_group_action_validation**() → None

Check that actions are denied when an unauthorised request is made.

This test checks the integration between SimComponent and the permissions validation system. First, we create a basic node and folder class. We configure the node so that only admins can create a folder. Then, we try to create a folder as both an admin user and a non-admin user.

3.8.4.1.1.2 tests.integration_tests.component_creation.test_permission_system.test_hierarchical_action_with_val

tests.integration_tests.component_creation.test_permission_system.**test_hierarchical_action_with_validat**

Check that validation works with sub-objects.

This test creates a parent object (Node) and a child object (Application) which both accept actions. The node allows action passthrough to applications. The purpose of this test is to check that after an action is passed through to a child object, that the permission system still works as intended.

3.8.4.2 tests.integration_tests.network

tests.integration_tests.network.test_frame_transmission
tests.integration_tests.network.test_link_connection
tests.integration_tests.network.test_nic_link_connection

3.8.4.2.1 tests.integration_tests.network.test_frame_transmission

Functions

test_multi_nic	Tests that Nodes with multiple NICs can ping each other and the data go across the correct links.
test_node_to_node_ping	Tests two Nodes are able to ping each other.
test_switched_network	Tests a larges network of Nodes and Switches with one node pinging another.

3.8.4.2.1.1 tests.integration_tests.network.test_frame_transmission.test_multi_nic

tests.integration_tests.network.test_frame_transmission.**test_multi_nic**()

Tests that Nodes with multiple NICs can ping each other and the data go across the correct links.

3.8.4.2.1.2 tests.integration_tests.network.test_frame_transmission.test_node_to_node_ping

tests.integration_tests.network.test_frame_transmission.**test_node_to_node_ping**()

Tests two Nodes are able to ping each other.

3.8.4.2.1.3 tests.integration_tests.network.test_frame_transmission.test_switched_network

tests.integration_tests.network.test_frame_transmission.**test_switched_network**()

Tests a larges network of Nodes and Switches with one node pinging another.

3.8.4.2.2 tests.integration_tests.network.test_link_connection

Functions

test_link_up	Tests Nodes, NICs, and Links can all be connected and be in an enabled/up state.

3.8.4.2.2.1 tests.integration_tests.network.test_link_connection.test_link_up

tests.integration_tests.network.test_link_connection.**test_link_up**()

Tests Nodes, NICs, and Links can all be connected and be in an enabled/up state.

3.8.4.2.3 tests.integration_tests.network.test_nic_link_connection

Functions

test_link_fails_with_same_nic	Tests Link creation fails with endpoint_a and endpoint_b are the same NIC.

3.8.4.2.3.1 tests.integration_tests.network.test_nic_link_connection.test_link_fails_with_same_nic

tests.integration_tests.network.test_nic_link_connection.**test_link_fails_with_same_nic**()

Tests Link creation fails with endpoint_a and endpoint_b are the same NIC.

3.8.5 tests.mock_and_patch

tests.mock_and_patch.get_session_path_mock	

3.8.5.1 tests.mock_and_patch.get_session_path_mock

Functions

get_temp_session_path	Get a temp directory session path the test session will output to.

3.8.5.1.1 tests.mock_and_patch.get_session_path_mock.get_temp_session_path

tests.mock_and_patch.get_session_path_mock.**get_temp_session_path**(*session_timestamp: datetime*) → Path

Get a temp directory session path the test session will output to.

Parameters

session_timestamp – This is the datetime that the session started.

Returns

The session directory path.

3.8.6 tests.test_acl

Used to tes the ACL functions.

Functions

test_acl_address_match_1	Test that matching IP addresses produce True.
test_acl_address_match_2	Test that mismatching IP addresses produce False.
test_acl_address_match_3	Test the ANY condition for source IP addresses produce True.
test_acl_address_match_4	Test the ANY condition for dest IP addresses produce True.
test_check_acl_block_affirmative	Test the block function (affirmative).
test_check_acl_block_negative	Test the block function (negative).
test_delete_rule	Adds 3 rules and deletes 1 rule and checks its deletion.
test_rule_hash	Test the rule hash.

3.8.6.1 tests.test_acl.test_acl_address_match_1

tests.test_acl.**test_acl_address_match_1**()

Test that matching IP addresses produce True.

3.8.6.2 tests.test_acl.test_acl_address_match_2

tests.test_acl.**test_acl_address_match_2**()

Test that mismatching IP addresses produce False.

3.8.6.3 tests.test_acl.test_acl_address_match_3

tests.test_acl.**test_acl_address_match_3**()

Test the ANY condition for source IP addresses produce True.

3.8.6.4 tests.test_acl.test_acl_address_match_4

tests.test_acl.**test_acl_address_match_4**()

Test the ANY condition for dest IP addresses produce True.

3.8.6.5 tests.test_acl.test_check_acl_block_affirmative

tests.test_acl.**test_check_acl_block_affirmative**()

Test the block function (affirmative).

3.8.6.6 tests.test_acl.test_check_acl_block_negative

tests.test_acl.**test_check_acl_block_negative**()

Test the block function (negative).

3.8.6.7 tests.test_acl.test_delete_rule

tests.test_acl.**test_delete_rule**()

Adds 3 rules and deletes 1 rule and checks its deletion.

3.8.6.8 tests.test_acl.test_rule_hash

tests.test_acl.**test_rule_hash**()

Test the rule hash.

3.8.7 tests.test_active_node

Used to test Active Node functions.

Functions

test_file_system_change	Test that a node cannot change its file system state when its hardware state is ON.
test_file_system_change_if_not_compromised	Test that a node cannot change its file system state.
test_os_state_change	Test that a node cannot change its Software State.
test_os_state_change_if_not_compromised	Test that a node cannot change its Software State.

3.8.7.1 tests.test_active_node.test_file_system_change

tests.test_active_node.**test_file_system_change**(*operating_state*, *expected_state*)

Test that a node cannot change its file system state when its hardware state is ON.

3.8.7.2 tests.test_active_node.test_file_system_change_if_not_compromised

tests.test_active_node.**test_file_system_change_if_not_compromised**(*operating_state*, *expected_state*)

Test that a node cannot change its file system state.

If not compromised) when its hardware state is OFF.

3.8.7.3 tests.test_active_node.test_os_state_change

tests.test_active_node.**test_os_state_change**(*operating_state*, *expected_state*)

Test that a node cannot change its Software State.

When its hardware state is OFF.

3.8.7.4 tests.test_active_node.test_os_state_change_if_not_compromised

tests.test_active_node.**test_os_state_change_if_not_compromised**(*operating_state*, *expected_state*)

Test that a node cannot change its Software State.

If not compromised) when its hardware state is OFF.

3.8.8 tests.test_full_legacy_config_session

Functions

test_legacy_training_config_run_session	Tests using legacy training and lay down config files in PrimAITE session end-to-end.

3.8.8.1 tests.test_full_legacy_config_session.test_legacy_training_config_run_session

tests.test_full_legacy_config_session.**test_legacy_training_config_run_session**(*legacy_file*)

Tests using legacy training and lay down config files in PrimAITE session end-to-end.

3.8.9 tests.test_lay_down_config

Functions

test_legacy_lay_down_config_load	Tests converting legacy lay down files into the new format.

3.8.9.1 tests.test_lay_down_config.test_legacy_lay_down_config_load

tests.test_lay_down_config.**test_legacy_lay_down_config_load**(*legacy_file*, *new_path*)

Tests converting legacy lay down files into the new format.

3.8.10 tests.test_observation_space

Test env creation and behaviour with different observation spaces.

Functions

test_default_obs_space	Create environment with no obs space defined in config and check that the default obs space was created.
test_registering_components	Test regitering and deregistering a component.

3.8.10.1 tests.test_observation_space.test_default_obs_space

tests.test_observation_space.**test_default_obs_space**(*temp_primaite_session*)

Create environment with no obs space defined in config and check that the default obs space was created.

3.8.10.2 tests.test_observation_space.test_registering_components

tests.test_observation_space.**test_registering_components**(*temp_primaite_session*)

Test regitering and deregistering a component.

Classes

TestAccessControlList	Test the AccessControlList observation component (in isolation).
TestLinkTrafficLevels	Test the LinkTrafficLevels observation component (in isolation).
TestNodeLinkTable	Test the NodeLinkTable observation component (in isolation).
TestNodeStatuses	Test the NodeStatuses observation component (in isolation).

3.8.10.3 tests.test_observation_space.TestAccessControlList

class tests.test_observation_space.**TestAccessControlList**

Bases: `object`

Test the AccessControlList observation component (in isolation).

Methods

test_obs_shape	Try creating env with MultiDiscrete observation space.
test_observation_space_with_different_pos	Test observation space is what is expected when an agent adds ACLs during an episode.
test_observation_space_with_implicit_rule	Test observation space is what is expected when an agent adds ACLs during an episode.
test_values	Test that traffic values are encoded correctly.

Attributes

pytestmark

test_obs_shape(*temp_primaite_session*)

Try creating env with MultiDiscrete observation space.

The laydown has 3 ACL Rules - that is the maximum_acl_rules it can have. Each ACL Rule in the observation space has 6 different elements:

6 * 3 = 18

test_observation_space_with_different_positions(*temp_primaite_session*)

Test observation space is what is expected when an agent adds ACLs during an episode.

At the start of the episode, there is a single implicit DENY rule In the observation space IMPLICIT DENY: 1,1,1,1,1,0 0 shows the rule is the start (when episode began no other rules were created) so this is correct.

On Step 2, there is an ACL rule added at Position 1: 2,2,3,2,3,1

On Step 4 there is a second ACL rule added at Position 0: 2,4,2,3,3,0

The final observation space should be this:
[2 , 4, 2, 3, 3, 0, 2, 2, 3, 2, 3, 1, 1, 1, 1, 1, 1, 2]

The ACL Rule from Step 2 is added before and has a LOWER position than the ACL rule from Step 4 but both come before the IMPLICIT DENY which will ALWAYS be at the end of the ACL List.

test_observation_space_with_implicit_rule(*temp_primaite_session*)

Test observation space is what is expected when an agent adds ACLs during an episode.

At the start of the episode, there is a single implicit DENY rule In the observation space IMPLICIT DENY: 1,1,1,1,1,0 0 shows the rule is the start (when episode began no other rules were created) so this is correct.

On Step 2, there is an ACL rule added at Position 0: 2,2,3,2,3,0

On Step 4, there is a second ACL rule added at POSITION 1: 2,4,2,3,3,1

The final observation space should be this:
[2, 2, 3, 2, 3, 0, 2, 4, 2, 3, 3, 1, 1, 1, 1, 1, 1, 2]

The ACL Rule from Step 2 is added first and has a HIGHER position than the ACL rule from Step 4 but both come before the IMPLICIT DENY which will ALWAYS be at the end of the ACL List.

test_values(*temp_primaite_session*)

Test that traffic values are encoded correctly.

The laydown has:

- one ACL IMPLICIT DENY rule

Therefore, the ACL is full of NAs aka zeros and just 6 non-zero elements representing DENY ANY ANY ANY at Position 2.

3.8.10.4 tests.test_observation_space.TestLinkTrafficLevels

class tests.test_observation_space.**TestLinkTrafficLevels**

Bases: object

Test the LinkTrafficLevels observation component (in isolation).

Methods

test_obs_shape	Try creating env with MultiDiscrete observation space.
test_values	Test that traffic values are encoded correctly.

Attributes

pytestmark

test_obs_shape(*temp_primaite_session*)

Try creating env with MultiDiscrete observation space.

test_values(*temp_primaite_session*)

Test that traffic values are encoded correctly.

The laydown has:

- two services
- three nodes
- two links
- an IER trying to send 999 bits of data over both links the whole time (via the first service)
- link bandwidth of 1000, therefore the utilisation is 99.9%

3.8.10.5 tests.test_observation_space.TestNodeLinkTable

class tests.test_observation_space.**TestNodeLinkTable**

Bases: object

Test the NodeLinkTable observation component (in isolation).

Methods

test_obs_shape	Try creating env with box observation space.
test_value	Test that the observation is generated correctly.

Attributes

pytestmark

test_obs_shape(*temp_primaite_session*)

Try creating env with box observation space.

test_value(*temp_primaite_session*)

Test that the observation is generated correctly.

The laydown has:

- 3 nodes (2 service nodes and 1 active node)
- 2 services
- 2 links

Both nodes have both services, and all states are GOOD, therefore the expected observation value is:

- **Node 1:**
  - 1 (id)
  - 1 (good hardware state)
  - 3 (compromised OS state)
  - 1 (good file system state)
  - 1 (good TCP state)
  - 1 (good UDP state)
- **Node 2:**
  - 2 (id)
  - 1 (good hardware state)
  - 1 (good OS state)
  - 1 (good file system state)
  - 1 (good TCP state)

  - 4 (overwhelmed UDP state)
- **Node 3 (active node):**
  - 3 (id)
  - 1 (good hardware state)
  - 1 (good OS state)
  - 1 (good file system state)
  - 0 (doesn't have service1)
  - 0 (doesn't have service2)
- **Link 1:**
  - 4 (id)
  - 0 (n/a hardware state)
  - 0 (n/a OS state)
  - 0 (n/a file system state)
  - 999 (999 traffic for service1)
  - 0 (no traffic for service2)
- **Link 2:**
  - 5 (id)
  - 0 (good hardware state)
  - 0 (good OS state)
  - 0 (good file system state)
  - 999 (999 traffic service1)
  - 0 (no traffic for service2)

3.8.10.6 tests.test_observation_space.TestNodeStatuses

class tests.test_observation_space.**TestNodeStatuses**

Bases: object

Test the NodeStatuses observation component (in isolation).

Methods

test_obs_shape	Try creating env with NodeStatuses as the only component.
test_values	Test that the hardware and software states are encoded correctly.

Attributes

```
pytestmark
```

test_obs_shape(*temp_primaite_session*)

Try creating env with NodeStatuses as the only component.

test_values(*temp_primaite_session*)

Test that the hardware and software states are encoded correctly.

The laydown has:

- one node with a compromised operating system state
- one node with two services, and the second service is overwhelmed.
- all other states are good or null

Therefore, the expected state is:

- **node 1:**
  - hardware = good (1)
  - OS = compromised (3)
  - file system = good (1)
  - service 1 = good (1)
  - service 2 = good (1)
- **node 2:**
  - hardware = good (1)
  - OS = good (1)
  - file system = good (1)
  - service 1 = good (1)
  - service 2 = overwhelmed (4)
- **node 3 (switch):**
  - hardware = good (1)
  - OS = good (1)
  - file system = good (1)
  - service 1 = n/a (0)
  - service 2 = n/a (0)

3.8.11 tests.test_primaite_session

Functions

test_primaite_session	Tests the PrimaiteSession class and all of its outputs.

3.8.11.1 tests.test_primaite_session.test_primaite_session

tests.test_primaite_session.**test_primaite_session**(*temp_primaite_session*)

Tests the PrimaiteSession class and all of its outputs.

This test runs for both a Stable Baselines3 agent, and a Ray RLlib agent.

3.8.12 tests.test_red_random_agent_behaviour

Functions

test_random_red_agent_behaviour	Test that red agent POL is randomised each episode.

3.8.12.1 tests.test_red_random_agent_behaviour.test_random_red_agent_behaviour

tests.test_red_random_agent_behaviour.**test_random_red_agent_behaviour**(*temp_primaite_session*)

Test that red agent POL is randomised each episode.

3.8.13 tests.test_resetting_node

Used to test Active Node functions.

Functions

test_node_boots_correctly	Tests that a node boots correctly.
test_node_resets_correctly	Tests that a node resets correctly.
test_node_shutdown_correctly	Tests that a node shutdown correctly.

3.8.13.1 tests.test_resetting_node.test_node_boots_correctly

tests.test_resetting_node.**test_node_boots_correctly**(*operating_state*, *expected_operating_state*)

Tests that a node boots correctly.

3.8.13.2 tests.test_resetting_node.test_node_resets_correctly

tests.test_resetting_node.**test_node_resets_correctly**(*starting_operating_state*, *expected_operating_state*)

Tests that a node resets correctly.

3.8.13.3 tests.test_resetting_node.test_node_shutdown_correctly

tests.test_resetting_node.**test_node_shutdown_correctly**(*operating_state*, *expected_operating_state*)

Tests that a node shutdown correctly.

3.8.14 tests.test_reward

Functions

test_rewards_are_being_penalised_at_each_st	Test that hardware state is penalised at each step.

3.8.14.1 tests.test_reward.test_rewards_are_being_penalised_at_each_step_function

tests.test_reward.**test_rewards_are_being_penalised_at_each_step_function**(*temp_primaite_session*)

Test that hardware state is penalised at each step.

When the initial state is OFF compared to reference state which is ON.

The config 'one_node_states_on_off_lay_down_config.yaml' has 15 steps:

On different steps, the laydown config has Pattern of Life (PoLs) which change a state of the node's attribute. For example, turning the nodes' file system state to CORRUPT from its original state GOOD. As a result these are the following rewards are activated:

File System State: corrupt_should_be_good = -10 * 2 (on Steps 1 & 2) Hardware State: off_should_be_on = -10 * 2 (on Steps 4 & 5) Service State: compromised_should_be_good = -20 * 2 (on Steps 7 & 8) Software State: compromised_should_be_good = -20 * 2 (on Steps 10 & 11)

The Pattern of Life (PoLs) last for 2 steps, so the agent is penalised twice.

Note: This test run inherits from conftest.py where the PrimAITE environment is ran and the blue agent is hard-coded to do NOTHING on every step. We use Pattern of Lifes (PoLs) to change the nodes states and display that the agent is being penalised on all steps where the live network node differs from the network reference node.

Total Reward: -10 + -10 + -10 + -10 + -20 + -20 + -20 + -20 = -120 Step Count: 15

For the 4 steps where this occurs the average reward is:

Average Reward: -8 (-120 / 15)

3.8.15 tests.test_seeding_and_deterministic_session

Functions

test_deterministic_evaluation	Test running deterministic evaluation gives same av eward per episode.
test_seeded_learning	Test running seeded learning produces the same output when ran twice.

3.8.15.1 tests.test_seeding_and_deterministic_session.test_deterministic_evaluation

tests.test_seeding_and_deterministic_session.**test_deterministic_evaluation**(*temp_primaite_session*)

Test running deterministic evaluation gives same av eward per episode.

3.8.15.2 tests.test_seeding_and_deterministic_session.test_seeded_learning

tests.test_seeding_and_deterministic_session.**test_seeded_learning**(*temp_primaite_session*)

Test running seeded learning produces the same output when ran twice.

Note: If this is failing, the hard-coded expected_mean_reward_per_episode from a pre-trained agent will probably need to be updated. If the env changes and those changed how this agent is trained, chances are the mean rewards are going to be different.

Run the test, but print out the session.learn_av_reward_per_episode() before comparing it. Then copy the printed dict and replace the expected_mean_reward_per_episode with those values. The test should now work. If not, then you've got a bug :).

3.8.16 tests.test_service_node

Used to test Service Node functions.

Functions

test_service_state_change	Test that a node cannot change the state of a running service.
test_service_state_change_if_not_comprised	Test that a node cannot change the state of a running service.

3.8.16.1 tests.test_service_node.test_service_state_change

tests.test_service_node.**test_service_state_change**(*operating_state*, *expected_state*)

Test that a node cannot change the state of a running service.

When its hardware state is OFF.

3.8.16.2 tests.test_service_node.test_service_state_change_if_not_comprised

tests.test_service_node.**test_service_state_change_if_not_comprised**(*operating_state*, *expected_state*)

Test that a node cannot change the state of a running service.

If not compromised when its hardware state is ON.

3.8.17 tests.test_session_loading

Functions

copy_session_asset	Copies the asset into a temporary test folder.
test_cli	Test loading session via CLI.
test_load_primaite_session	Test that loading a Primaite session works.
test_load_sb3_session	Test that loading an SB3 agent works.
test_run_loading	Test loading session via main.run.

3.8.17.1 tests.test_session_loading.copy_session_asset

tests.test_session_loading.**copy_session_asset**(*asset_path: str | Path*) → str

Copies the asset into a temporary test folder.

3.8.17.2 tests.test_session_loading.test_cli

tests.test_session_loading.**test_cli**()

Test loading session via CLI.

3.8.17.3 tests.test_session_loading.test_load_primaite_session

tests.test_session_loading.**test_load_primaite_session**()

Test that loading a Primaite session works.

3.8.17.4 tests.test_session_loading.test_load_sb3_session

tests.test_session_loading.**test_load_sb3_session**()

Test that loading an SB3 agent works.

3.8.17.5 tests.test_session_loading.test_run_loading

tests.test_session_loading.**test_run_loading**()

Test loading session via main.run.

3.8.18 tests.test_single_action_space

Functions

run_generic_set_actions	Run against a generic agent with specified blue agent actions.
test_agent_is_executing_actions_from_both_sp	Test to ensure the blue agent is carrying out both kinds of operations (NODE & ACL).
test_single_action_space_is_valid	Test single action space is valid.

3.8.18.1 tests.test_single_action_space.run_generic_set_actions

tests.test_single_action_space.**run_generic_set_actions**(*env:* Primaite)

Run against a generic agent with specified blue agent actions.

3.8.18.2 tests.test_single_action_space.test_agent_is_executing_actions_from_both_spaces

tests.test_single_action_space.**test_agent_is_executing_actions_from_both_spaces**(*temp_primaite_session*)

Test to ensure the blue agent is carrying out both kinds of operations (NODE & ACL).

3.8.18.3 tests.test_single_action_space.test_single_action_space_is_valid

tests.test_single_action_space.**test_single_action_space_is_valid**(*temp_primaite_session*)

Test single action space is valid.

3.8.19 tests.test_train_eval_episode_steps

Functions

test_eval_steps_differ_from_training	Uses PrimaiteSession class to compare number of episodes used for training and evaluation.

3.8.19.1 tests.test_train_eval_episode_steps.test_eval_steps_differ_from_training

tests.test_train_eval_episode_steps.**test_eval_steps_differ_from_training**(*temp_primaite_session*)

Uses PrimaiteSession class to compare number of episodes used for training and evaluation.

Train_episode_step.yaml main config:

num_train_steps = 25 num_train_episodes = 3 num_eval_steps = 17 num_eval_episodes = 1

3.8.20 tests.test_training_config

Functions

test_create_config_values_main_from_file	Tests creating an instance of TrainingConfig from file.
test_create_config_values_main_from_legacy_f	Tests creating an instance of TrainingConfig from legacy file.
test_legacy_lay_down_config_yaml_conversion	Tests the conversion of legacy lay down config files.

3.8.20.1 tests.test_training_config.test_create_config_values_main_from_file

tests.test_training_config.**test_create_config_values_main_from_file**()

Tests creating an instance of TrainingConfig from file.

3.8.20.2 tests.test_training_config.test_create_config_values_main_from_legacy_file

tests.test_training_config.**test_create_config_values_main_from_legacy_file**()

Tests creating an instance of TrainingConfig from legacy file.

3.8.20.3 tests.test_training_config.test_legacy_lay_down_config_yaml_conversion

tests.test_training_config.**test_legacy_lay_down_config_yaml_conversion**()

Tests the conversion of legacy lay down config files.

3.8.21 tests.unit_tests

3.9 Dependencies

3.9.1 PrimAITE Dependencies

Name	Version	License
Babel	2.12.1	BSD License
GPUtil	1.4.0	MIT
Jinja2	3.1.2	BSD License
Markdown	3.4.4	BSD License
MarkupSafe	2.1.3	BSD License
Pillow	10.0.0	Historical Permission Notice and Disclaimer (HPND)

Table 13 – continued from previous

PyLaTeX	1.4.1	MIT License
PyWavelets	1.4.1	MIT License
PyYAML	6.0	MIT License
Pygments	2.15.1	BSD License
Send2Trash	1.8.2	BSD License
Sphinx	6.1.3	BSD License
Werkzeug	2.3.6	BSD License
absl-py	1.4.0	Apache Software License
aiofiles	22.1.0	Apache Software License
aiosignal	1.3.1	Apache Software License
aiosqlite	0.19.0	MIT License
alabaster	0.7.13	BSD License
annotated-types	0.5.0	MIT License
anyio	3.7.1	MIT License
argon2-cffi	21.3.0	MIT License
argon2-cffi-bindings	21.2.0	MIT License
arrow	1.2.3	Apache Software License
asttokens	2.2.1	Apache 2.0
astunparse	1.6.3	BSD License
attrs	23.1.0	MIT License
backcall	0.2.0	BSD License
beautifulsoup4	4.12.2	MIT License
bleach	6.0.0	Apache Software License
build	0.10.0	MIT License
cachetools	5.3.1	MIT License
certifi	2023.7.22	Mozilla Public License 2.0 (MPL 2.0)
cffi	1.15.1	MIT License
cfgv	3.3.1	MIT License
charset-normalizer	3.2.0	MIT License
click	8.1.6	BSD License
cloudpickle	2.2.1	BSD License
colorama	0.4.6	BSD License
comm	0.1.4	BSD License
contourpy	1.1.0	BSD License
coverage	7.2.7	Apache Software License
cycler	0.11.0	BSD License
debugpy	1.6.8	Eclipse Public License 2.0 (EPL-2.0); MIT License
decorator	5.1.1	BSD License
defusedxml	0.7.1	Python Software Foundation License
distlib	0.3.7	Python Software Foundation License
dm-tree	0.1.8	Apache Software License
docutils	0.19	BSD License; GNU General Public License (GPL); Public Domain; Python Software F
entrypoints	0.4	MIT License
exceptiongroup	1.1.2	MIT License
execnet	2.0.2	MIT License
executing	1.2.0	MIT License
fastjsonschema	2.18.0	BSD License
filelock	3.12.2	The Unlicense (Unlicense)
flake8	6.0.0	MIT License
flatbuffers	23.5.26	Apache Software License
fonttools	4.42.0	MIT License
fqdn	1.5.1	Mozilla Public License 2.0 (MPL 2.0)

Table 13 – continued from previous

frozenlist	1.4.0	Apache Software License
furo	2023.3.27	MIT License
gast	0.4.0	BSD License
google-auth	2.22.0	Apache Software License
google-auth-oauthlib	1.0.0	Apache Software License
google-pasta	0.2.0	Apache Software License
grpcio	1.56.2	Apache Software License
gym	0.21.0	UNKNOWN
h5py	3.9.0	BSD License
identify	2.5.26	MIT License
idna	3.4	BSD License
imageio	2.31.1	BSD License
imagesize	1.4.1	MIT License
importlib-metadata	4.13.0	Apache Software License
iniconfig	2.0.0	MIT License
ipykernel	6.25.0	BSD License
ipython	8.14.0	BSD License
ipython-genutils	0.2.0	BSD License
isoduration	20.11.0	ISC License (ISCL)
jax	0.4.14	Apache-2.0
jedi	0.19.0	MIT License
json5	0.9.14	Apache Software License
jsonpointer	2.4	BSD License
jsonschema	4.18.6	MIT License
jsonschema-specifications	2023.7.1	MIT License
jupyter-events	0.7.0	BSD License
jupyter-ydoc	0.2.5	BSD 3-Clause License
jupyter_client	7.4.9	BSD License
jupyter_core	5.3.1	BSD License
jupyter_server	2.7.0	BSD License
jupyter_server_fileid	0.9.0	BSD License
jupyter_server_terminals	0.4.4	BSD License
jupyter_server_ydoc	0.6.1	BSD License
jupyterlab	3.6.1	BSD License
jupyterlab-pygments	0.2.2	BSD
jupyterlab_server	2.24.0	BSD License
kaleido	0.2.1	MIT
keras	2.12.0	Apache Software License
kiwisolver	1.4.4	BSD License
lazy_loader	0.3	BSD License
libclang	16.0.6	Apache Software License
lz4	4.3.2	BSD License
markdown-it-py	3.0.0	MIT License
matplotlib	3.7.1	Python Software Foundation License
matplotlib-inline	0.1.6	BSD 3-Clause
mccabe	0.7.0	MIT License
mdurl	0.1.2	MIT License
mistune	3.0.1	BSD License
ml-dtypes	0.2.0	Apache Software License
mpmath	1.3.0	BSD License
msgpack	1.0.5	Apache Software License
nbclassic	1.0.0	BSD License

Table 13 – continued from previous

nbclient	0.8.0	BSD License
nbconvert	7.7.3	BSD License
nbformat	5.9.2	BSD License
nest-asyncio	1.5.7	BSD License
networkx	3.1	BSD License
nodeenv	1.8.0	BSD License
notebook	6.5.5	BSD License
notebook_shim	0.2.3	BSD License
numpy	1.23.5	BSD License
oauthlib	3.2.2	BSD License
opt-einsum	3.3.0	MIT
ordered-set	4.1.0	MIT License
overrides	7.3.1	Apache License, Version 2.0
packaging	23.1	Apache Software License; BSD License
pandas	2.0.3	BSD License
pandocfilters	1.5.0	BSD License
parso	0.8.3	MIT License
pickleshare	0.7.5	MIT License
platformdirs	3.5.1	MIT License
plotly	5.15.0	MIT License
pluggy	1.2.0	MIT License
polars	0.18.4	MIT License
pre-commit	2.20.0	MIT License
primaite	2.0.0	GFX
primaite	2.0.0	GFX
prometheus-client	0.17.1	Apache Software License
prompt-toolkit	3.0.39	BSD License
protobuf	4.23.4	3-Clause BSD License
psutil	5.9.5	BSD License
pure-eval	0.2.2	MIT License
pyasn1	0.5.0	BSD License
pyasn1-modules	0.3.0	BSD License
pycodestyle	2.10.0	MIT License
pycparser	2.21	BSD License
pydantic	2.1.1	MIT License
pydantic_core	2.4.0	MIT License
pyflakes	3.0.1	MIT License
pyparsing	3.1.1	MIT License
pyproject_hooks	1.0.0	MIT License
pytest	7.2.0	MIT License
pytest-cov	4.0.0	MIT License
pytest-flake8	1.1.1	BSD License
pytest-xdist	3.3.1	MIT License
python-dateutil	2.8.2	Apache Software License; BSD License
python-json-logger	2.0.7	BSD License
pytz	2023.3	MIT License
pywin32	306	Python Software Foundation License
pywinpty	2.0.11	MIT
pyzmq	24.0.1	BSD License; GNU Library or Lesser General Public License (LGPL)
ray	2.2.0	Apache 2.0
referencing	0.30.0	MIT License
requests	2.31.0	Apache Software License

Table 13 – continued from previous

requests-oauthlib	1.3.1	BSD License
rfc3339-validator	0.1.4	MIT License
rfc3986-validator	0.1.1	MIT License
rich	13.5.2	MIT License
rpds-py	0.9.2	MIT License
rsa	4.9	Apache Software License
scikit-image	0.21.0	BSD License
scipy	1.11.1	BSD License
shellingham	1.5.0.post1	ISC License (ISCL)
six	1.16.0	MIT License
sniffio	1.3.0	Apache Software License; MIT License
snowballstemmer	2.2.0	BSD License
soupsieve	2.4.1	MIT License
sphinx-basic-ng	1.0.0b2	MIT License
sphinx-code-tabs	0.5.5	The Unlicense (Unlicense)
sphinx-copybutton	0.5.2	MIT License
sphinxcontrib-applehelp	1.0.4	BSD License
sphinxcontrib-devhelp	1.0.2	BSD License
sphinxcontrib-htmlhelp	2.0.1	BSD License
sphinxcontrib-jsmath	1.0.1	BSD License
sphinxcontrib-qthelp	1.0.3	BSD License
sphinxcontrib-serializinghtml	1.1.5	BSD License
stable-baselines3	1.6.2	MIT
stack-data	0.6.2	MIT License
sympy	1.12	BSD License
tabulate	0.9.0	MIT License
tenacity	8.2.2	Apache Software License
tensorboard	2.12.3	Apache Software License
tensorboard-data-server	0.7.1	Apache Software License
tensorboardX	2.6.2	MIT License
tensorflow	2.12.0	Apache Software License
tensorflow-estimator	2.12.0	Apache Software License
tensorflow-intel	2.12.0	Apache Software License
tensorflow-io-gcs-filesystem	0.31.0	Apache Software License
termcolor	2.3.0	MIT License
terminado	0.17.1	BSD License
tifffile	2023.7.18	BSD License
tinycss2	1.2.1	BSD License
toml	0.10.2	MIT License
tomli	2.0.1	MIT License
torch	2.0.1	BSD License
tornado	6.3.2	Apache Software License
traitlets	5.9.0	BSD License
typer	0.9.0	MIT License
typing_extensions	4.7.1	Python Software Foundation License
tzdata	2023.3	Apache Software License
uri-template	1.3.0	MIT License
urllib3	1.26.16	MIT License
virtualenv	20.24.1	MIT License
webcolors	1.13	BSD License
webencodings	0.5.1	BSD License
websocket-client	1.6.1	Apache Software License

Table 13 – continued from previous

wrapt	1.14.1	BSD License
y-py	0.6.0	MIT License
ypy-websocket	0.8.4	UNKNOWN
zipp	3.16.2	MIT License

3.10 Glossary

Access Control List
PrimAITE blocks or allows certain traffic on the network by simulating firewall rules, which are defined in the Access Control List.

Action
The learning agent decides on an action to take on every step in the simulation. The action has the chance to positively or negatively impact the environment state. Over time, the agent aims to learn which actions to take when to maximise the expected reward.

Agent
An agent is a representation of a user of the network. Typically this would be a user that is using one of the computer nodes, though it could be an autonomous agent.

Blue Agent
A defensive agent that protects the network from Red Agent attacks to minimise disruption to green agents and protect data.

Episode
When an episode starts, the network simulation is reset to an initial state. The agents take actions on each step of the episode until it reaches a terminal state, which usually happens after a predetermined number of steps. After the terminal state is reached, a new episode starts and the RL agent has another opportunity to protect the network.

Evaluation
During evaluation, an RL agent acts on the simulated network but it is not allowed to update it's behaviour. Evaluation is used to assess how successful agents are at defending the network.

Green agent
Simulates typical benign activity on the network, such as real users using computers and servers.

Gym
PrimAITE uses the Gym reinforcement learning framework API to create a training environment and interface with RL agents. Gym defines a common way of creating observations, actions, and rewards.

Information Exchange Requirement (IER)
Simulates network traffic by sending data from one network node to another via links for a specified amount of time. IERs can be part of green agent behaviour or red agent behaviour. PrimAITE can be configured to apply a penalty for green agents' IERs being blocked and a reward for red agents' IERs being blocked.

Laydown
The laydown is a file which defines the training scenario. It contains the network topology, firewall rules, services, protocols, and details about green and red agent behaviours.

Link
A Link represents the connection between two Nodes. For example, a physical wire between a computer and a switch or a wireless connection.

Network
The network in primaite is a logical representation of a computer network containing *Nodes* and *Links*.

Node
A Node represents a network endpoint. For example a computer, server, switch, or an actuator.

Observation
An observation is a representation of the current state of the environment that is given to the learning agent so it can decide on which action to perform. If the environment is 'fully observable', the observation contains information about every possible aspect of the environment. More commonly, the environment is 'partially observable' which means the learning agent has to make decisions without knowing every detail of the current environment state.

Pattern-of-Life (PoL)
PoLs allow agents to change the current hardware, OS, file system, or service statuses of nodes during the course of an episode. For example, a green agent may restart a server node to represent scheduled maintainance. A red agent's Pattern-of-Life can be used to attack nodes by changing their states to CORRUPTED or COMPROMISED.

Protocol
Protocols are used by links to separate different types of network traffic. Common examples would be HTTP, TCP, and UDP.

Red Agent
An agent that is aiming to attack the network in some way, for example by executing a Denial-Of-Service attack or stealing data.

Reference environment
While the network simulation is unfolding, a parallel simulation takes place which is identical to the main one except that blue and red agent actions are not applied. This reference environment essentially shows what would be happening to the network if there had been no cyberattack or defense. The reference environment is used to calculate rewards.

Reward
The reward is a single number used by the blue agent to understand whether it's performing well or poorly. RL agents change their behaviour in an attempt to increase the expected reward each episode. The reward is generated based on the current states of the environment / *reference environment* and is impacted positively by things like green IERS running successfully and negatively by things like nodes being compromised.

Service
A service represents a piece of software that is installed on a node, such as a web server or a database.

Step
The agents can only act in the environment at discrete intervals. The time step is the basic unit of time in the simulation. At each step, the RL agent has an opportunity to observe the state of the environment and decide an action. Steps are also used for updating states for time-dependent activities such as rebooting a node.

Training
During training, an RL agent is placed in the simulated network and it learns which actions to take in which scenarios to obtain maximum reward.

Transaction
PrimAITE records the decisions of the learning agent by saving its observation, action, and reward at every time step. During each session, this data is saved to disk to allow for full inspection.

User app home
PrimAITE supports upgrading software version while retaining user data. The user data directory is where configs, notebooks, and results are stored, this location is *~/primaite<version>* on linux/darwin and *C:Users<username>primaite<version>* on Windows.

3.11 v1.2 to v2.0 Migration guide

1. Installing PrimAITE

Like before, you can install primaite from the repository by running `pip install -e .`. But, there is now an additional setup step which does several things, like setting up user directories, copy default configs and notebooks, etc. Once you have installed PrimAITE to your virtual environment, run this command to finalise setup.

```
primaite setup
```

2. Running a training session

In version 1.2 of PrimAITE, the main entry point for training or evaluating agents was the `src/primaite/main.py` file. v2.0.0 introduced managed 'sessions' which are responsible for reading configuration files, performing training, and writing outputs.

`main.py` file still runs a training session but it now uses the new *PrimaiteSession*, and it now requires you to provide the path to your config files.

```
python src/primaite/main.py --tc path/to/training-config.yaml --ldc path/to/
↪laydown-config.yaml
```

Alternatively, the session can be invoked via the commandline by running:

```
primaite session --tc path/to/training-config.yaml --ldc path/to/laydown-
↪config.yaml
```

3. Location of configs

In version 1.2, training configs and laydown configs were all stored in the project repository under `src/primaite/config`. Version 2.0.0 introduced user data directories, and now when you install and setup PrimAITE, config files are stored in your user data location. On Linux/OSX, this is stored in `~/primaite/2.0.0/config`. On Windows, this is stored in `C:\Users\<your username>\primaite\configs`. Upon first setup, the configs folder is populated with some default yaml files. It is recommended that you store all your custom configuration files here.

4. Contents of configs

Some things that were previously part of the laydown config are now part of the traning config.

- Actions

If you have custom configs which use these, you will need to adapt them by moving the configuration from the laydown config to the training config.

Also, there are new configurable items in the training config:

- Observations
- Agent framework
- Agent
- Deep learning framework
- random red agents
- seed
- deterministic

- hard coded agent view

Each of these items have default values which are designed so that PrimAITE has the same behaviour as it did in 1.2.0, so you do not have to specify them.

ACL Rules in laydown configs have a new required parameter: `position`. The lower the position, the higher up in the ACL table the rule will placed. If you have custom laydowns, you will need to go through them and add a position to each ACL_RULE.

PYTHON MODULE INDEX

p

t

INDEX

Symbols

A

B

C

D

E

F

G

H

M

Q

R

S

U

V

W

X

Z