

---

# **smc-python Documentation**

***Release 1.0.0***

**Forcepoint**

**Dec 15, 2023**



# CONTENTS

|          |                                                       |           |
|----------|-------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                   | <b>3</b>  |
| <b>2</b> | <b>Installation</b>                                   | <b>5</b>  |
| <b>3</b> | <b>Enable API on SMC</b>                              | <b>7</b>  |
| <b>4</b> | <b>Creating the session</b>                           | <b>9</b>  |
| 4.1      | Configuring credentials . . . . .                     | 9         |
| 4.1.1    | Method parameters . . . . .                           | 9         |
| 4.1.2    | Configuration File . . . . .                          | 10        |
| 4.1.3    | Environment Variables . . . . .                       | 11        |
| 4.2      | Handling retries on server busy . . . . .             | 11        |
| 4.3      | Handling proxies . . . . .                            | 12        |
| 4.4      | Logging helper . . . . .                              | 12        |
| <b>5</b> | <b>Resources</b>                                      | <b>13</b> |
| <b>6</b> | <b>Collections</b>                                    | <b>15</b> |
| 6.1      | ElementCollection . . . . .                           | 15        |
| 6.1.1    | Methods that return a new ElementCollection . . . . . | 16        |
| 6.1.2    | Basic rules on searching . . . . .                    | 17        |
| 6.1.3    | Additional Examples . . . . .                         | 18        |
| 6.2      | General Search . . . . .                              | 19        |
| <b>7</b> | <b>Elements</b>                                       | <b>23</b> |
| 7.1      | Create . . . . .                                      | 23        |
| 7.2      | Update . . . . .                                      | 24        |
| 7.3      | Delete . . . . .                                      | 25        |
| 7.4      | Functions or methods that modify . . . . .            | 26        |
| <b>8</b> | <b>Engines</b>                                        | <b>27</b> |
| 8.1      | Create . . . . .                                      | 27        |
| 8.1.1    | Layer3 Firewall . . . . .                             | 27        |
| 8.1.2    | Layer 2 Firewall . . . . .                            | 28        |
| 8.1.3    | IPS Engine . . . . .                                  | 28        |
| 8.1.4    | Master Engine . . . . .                               | 28        |
| 8.1.5    | Virtual Engine . . . . .                              | 29        |
| 8.1.6    | Firewall Cluster . . . . .                            | 30        |
| 8.1.7    | MasterEngine Cluster . . . . .                        | 30        |
| 8.2      | Nodes . . . . .                                       | 31        |
| 8.3      | Interfaces . . . . .                                  | 32        |

|           |                        |            |
|-----------|------------------------|------------|
| 8.3.1     | Sub-Interface and VLAN | 36         |
| 8.3.2     | Modifying Interfaces   | 36         |
| 8.3.3     | Deleting Interfaces    | 37         |
| 8.4       | Routing                | 38         |
| 8.5       | Licensing              | 38         |
| <b>9</b>  | <b>Policies</b>        | <b>39</b>  |
| <b>10</b> | <b>VPN</b>             | <b>41</b>  |
| <b>11</b> | <b>Administration</b>  | <b>43</b>  |
| 11.1      | Administrators         | 43         |
| 11.2      | Tasks                  | 43         |
| 11.3      | System                 | 44         |
| <b>12</b> | <b>Logging</b>         | <b>45</b>  |
| <b>13</b> | <b>Extensions</b>      | <b>47</b>  |
| 13.1      | smc-python-monitoring  | 47         |
| 13.1.1    | Query                  | 47         |
| 13.1.2    | Models                 | 51         |
| 13.1.2.1  | Filters                | 52         |
| 13.1.2.2  | Values                 | 56         |
| 13.1.2.3  | Formats                | 58         |
| 13.1.2.4  | Constants              | 61         |
| 13.1.2.5  | Formatters             | 81         |
| 13.1.2.6  | TimeRanges             | 82         |
| 13.1.3    | Monitors               | 85         |
| 13.1.3.1  | Blacklist              | 85         |
| 13.1.3.2  | Connections            | 88         |
| 13.1.3.3  | Logs                   | 90         |
| 13.1.3.4  | Routes                 | 93         |
| 13.1.3.5  | SSLVPN                 | 95         |
| 13.1.3.6  | Users                  | 97         |
| 13.1.3.7  | VPNs                   | 99         |
| 13.1.3.8  | Alerts                 | 102        |
| <b>14</b> | <b>API Reference</b>   | <b>105</b> |
| 14.1      | Session                | 105        |
| 14.2      | Element                | 108        |
| 14.3      | Administration         | 114        |
| 14.3.1    | Access Rights          | 114        |
| 14.3.1.1  | AccessControlList      | 115        |
| 14.3.1.2  | Administrators         | 116        |
| 14.3.1.3  | Permission             | 121        |
| 14.3.1.4  | Roles                  | 122        |
| 14.3.2    | Alert Configuration    | 124        |
| 14.3.2.1  | AlertChain             | 124        |
| 14.3.2.2  | AlertChainRule         | 124        |
| 14.3.2.3  | AlertPolicy            | 125        |
| 14.3.2.4  | AlertRule              | 125        |
| 14.3.3    | Certificates           | 126        |
| 14.3.3.1  | TLSCommon              | 126        |
| 14.3.3.2  | TLSServerCredential    | 128        |
| 14.3.3.3  | TLSProfile             | 133        |

|           |                             |     |
|-----------|-----------------------------|-----|
| 14.3.3.4  | TLSIdentity                 | 134 |
| 14.3.3.5  | TLSCryptographySuite        | 134 |
| 14.3.3.6  | TLSCertificateAuthority     | 135 |
| 14.3.3.7  | ClientProtectionCA          | 135 |
| 14.3.3.8  | VPNCertificateCA            | 137 |
| 14.3.3.9  | GatewayCertificate          | 137 |
| 14.3.4    | Domains                     | 137 |
| 14.3.5    | License                     | 139 |
| 14.3.5.1  | License                     | 139 |
| 14.3.6    | Scheduled Tasks             | 139 |
| 14.3.7    | Reports                     | 153 |
| 14.3.8    | System                      | 156 |
| 14.3.9    | Tasks                       | 165 |
| 14.3.10   | Upcoming Event              | 168 |
| 14.3.11   | Updates                     | 169 |
| 14.3.11.1 | Engine Upgrade              | 170 |
| 14.3.11.2 | Dynamic Update              | 171 |
| 14.4      | SD-WAN                      | 171 |
| 14.4.1    | Other Elements              | 171 |
| 14.4.1.1  | LinkUsageProfile            | 172 |
| 14.4.2    | Profiles                    | 173 |
| 14.4.2.1  | VPNProfile                  | 173 |
| 14.5      | Elements                    | 174 |
| 14.5.1    | Network                     | 174 |
| 14.5.1.1  | Alias                       | 174 |
| 14.5.1.2  | AddressRange                | 176 |
| 14.5.1.3  | DomainName                  | 177 |
| 14.5.1.4  | Expression                  | 178 |
| 14.5.1.5  | Host                        | 179 |
| 14.5.1.6  | IPList                      | 180 |
| 14.5.1.7  | Network                     | 182 |
| 14.5.1.8  | Router                      | 183 |
| 14.5.1.9  | URLListApplication          | 184 |
| 14.5.1.10 | Zone                        | 185 |
| 14.5.1.11 | Traffic Handlers (Netlinks) | 185 |
| 14.5.2    | Services                    | 193 |
| 14.5.2.1  | EthernetService             | 193 |
| 14.5.2.2  | RPCService                  | 194 |
| 14.5.2.3  | ICMPService                 | 195 |
| 14.5.2.4  | ICMPIPv6Service             | 195 |
| 14.5.2.5  | IPService                   | 196 |
| 14.5.2.6  | TCPService                  | 197 |
| 14.5.2.7  | UDPService                  | 198 |
| 14.5.2.8  | URLCategory                 | 199 |
| 14.5.2.9  | With Protocol               | 199 |
| 14.5.3    | Groups                      | 204 |
| 14.5.3.1  | ICMPServiceGroup            | 205 |
| 14.5.3.2  | IPServiceGroup              | 206 |
| 14.5.3.3  | RpcServiceGroup             | 206 |
| 14.5.3.4  | Group                       | 207 |
| 14.5.3.5  | ServiceGroup                | 207 |
| 14.5.3.6  | TCPServiceGroup             | 208 |
| 14.5.3.7  | UDPServiceGroup             | 209 |
| 14.5.3.8  | URLCategoryGroup            | 209 |

|           |                           |     |
|-----------|---------------------------|-----|
| 14.5.3.9  | EthernetServiceGroup      | 209 |
| 14.5.4    | Servers                   | 210 |
| 14.5.4.1  | LogServer                 | 217 |
| 14.5.4.2  | ManagementServer          | 219 |
| 14.5.4.3  | DNSServer                 | 223 |
| 14.5.4.4  | HttpProxy                 | 224 |
| 14.5.4.5  | ProxyServer               | 225 |
| 14.5.4.6  | DHCPServer                | 227 |
| 14.5.4.7  | WebPortalServer           | 228 |
| 14.5.4.8  | EpoServer                 | 229 |
| 14.5.4.9  | LDAPServer                | 230 |
| 14.5.4.10 | ActiveDirectoryServer     | 230 |
| 14.5.4.11 | SmtptServer               | 230 |
| 14.5.4.12 | IcapServer                | 231 |
| 14.5.4.13 | RadiusServer              | 232 |
| 14.5.4.14 | TacacsServer              | 232 |
| 14.5.4.15 | NTPServer                 | 233 |
| 14.5.4.16 | ElasticsearchCluster      | 234 |
| 14.5.5    | Other                     | 235 |
| 14.5.5.1  | Blacklist                 | 244 |
| 14.5.5.2  | Category                  | 245 |
| 14.5.5.3  | CategoryTag               | 247 |
| 14.5.5.4  | FilterExpression          | 248 |
| 14.5.5.5  | Location                  | 248 |
| 14.5.5.6  | LogicalInterface          | 249 |
| 14.5.5.7  | MacAddress                | 249 |
| 14.5.5.8  | HTTPSInspectionExceptions | 250 |
| 14.5.6    | Sidewinder Elements       | 250 |
| 14.5.6.1  | SSHProfile                | 253 |
| 14.5.6.2  | SSHKnownHosts             | 254 |
| 14.5.6.3  | SSHKnownHostsLists        | 255 |
| 14.5.7    | Situations                | 256 |
| 14.5.8    | Profiles                  | 262 |
| 14.5.8.1  | DNSRelayProfile           | 262 |
| 14.5.8.2  | SNMPAgent                 | 265 |
| 14.5.8.3  | LLDPProfile               | 265 |
| 14.6      | Engine                    | 266 |
| 14.6.1    | AddOn                     | 294 |
| 14.6.1.1  | AntiVirus                 | 294 |
| 14.6.1.2  | FileReputation            | 296 |
| 14.6.1.3  | SidewinderProxy           | 296 |
| 14.6.1.4  | UrlFiltering              | 297 |
| 14.6.1.5  | ZTNAConnector             | 298 |
| 14.6.1.6  | ClientInspection          | 298 |
| 14.6.1.7  | Sandbox                   | 299 |
| 14.6.1.8  | TLSInspection             | 300 |
| 14.6.2    | Dynamic Routing           | 301 |
| 14.6.2.1  | OSPF                      | 301 |
| 14.6.2.2  | BGP                       | 301 |
| 14.6.3    | General                   | 301 |
| 14.6.3.1  | DefaultNAT                | 301 |
| 14.6.3.2  | RankedDNSAddress          | 302 |
| 14.6.3.3  | DNS Relay                 | 303 |
| 14.6.3.4  | SNMP                      | 304 |

|           |                          |     |
|-----------|--------------------------|-----|
| 14.6.3.5  | Layer2Settings           | 305 |
| 14.6.3.6  | NTPSettings              | 306 |
| 14.6.4    | VPN                      | 306 |
| 14.6.4.1  | InternalEndpoint         | 308 |
| 14.6.4.2  | InternalGateway          | 309 |
| 14.6.5    | Interfaces               | 310 |
| 14.6.5.1  | InterfaceCollections     | 310 |
| 14.6.5.2  | InterfaceOptions         | 332 |
| 14.6.5.3  | QoS                      | 335 |
| 14.6.5.4  | LoopbackInterface        | 336 |
| 14.6.5.5  | LoopbackClusterInterface | 337 |
| 14.6.5.6  | PhysicalInterface        | 338 |
| 14.6.5.7  | Layer3PhysicalInterface  | 343 |
| 14.6.5.8  | Layer2PhysicalInterface  | 344 |
| 14.6.5.9  | ClusterPhysicalInterface | 345 |
| 14.6.5.10 | VirtualPhysicalInterface | 346 |
| 14.6.5.11 | SwitchPhysicalInterface  | 347 |
| 14.6.5.12 | TunnelInterface          | 348 |
| 14.6.5.13 | VPNBrokerInterface       | 349 |
| 14.6.5.14 | VlanInterface            | 349 |
| 14.6.5.15 | Sub-Interfaces           | 350 |
| 14.6.5.16 | InterfaceContactAddress  | 355 |
| 14.6.6    | Node                     | 358 |
| 14.6.6.1  | Appliance Info           | 365 |
| 14.6.6.2  | Appliance Status         | 366 |
| 14.6.6.3  | ApplianceSwitchModule    | 367 |
| 14.6.6.4  | Hardware Status          | 367 |
| 14.6.6.5  | Interface Status         | 369 |
| 14.6.6.6  | Debug                    | 369 |
| 14.6.7    | Pending Changes          | 370 |
| 14.6.8    | Routing                  | 371 |
| 14.6.8.1  | Routing                  | 376 |
| 14.6.8.2  | Antispoofing             | 381 |
| 14.6.8.3  | Route Table              | 382 |
| 14.6.8.4  | Policy Routing           | 383 |
| 14.6.8.5  | TunnelMonitoringGroup    | 384 |
| 14.6.9    | Snapshot                 | 384 |
| 14.6.10   | VirtualResource          | 385 |
| 14.6.11   | AdvanceSettings          | 386 |
| 14.7      | Engine Types             | 387 |
| 14.7.1    | IPS                      | 387 |
| 14.7.2    | Layer3Firewall           | 388 |
| 14.7.3    | Layer2Firewall           | 392 |
| 14.7.4    | Layer3VirtualEngine      | 393 |
| 14.7.5    | FirewallCluster          | 394 |
| 14.7.6    | MasterEngine             | 397 |
| 14.7.7    | MasterEngineCluster      | 398 |
| 14.7.8    | CloudSGSingleFW          | 399 |
| 14.8      | Dynamic Routing Elements | 400 |
| 14.8.1    | RouteMap                 | 400 |
| 14.8.2    | IPAccessList             | 405 |
| 14.8.3    | IPPrefixList             | 408 |
| 14.8.4    | BGP Elements             | 409 |
| 14.8.4.1  | AutonomousSystem         | 413 |

|           |                             |     |
|-----------|-----------------------------|-----|
| 14.8.4.2  | ExternalBGPPeer             | 414 |
| 14.8.4.3  | BGPPEering                  | 415 |
| 14.8.4.4  | BGPProfile                  | 418 |
| 14.8.4.5  | BGPConnectionProfile        | 420 |
| 14.8.4.6  | ASPathAccessList            | 421 |
| 14.8.4.7  | CommunityAccessList         | 422 |
| 14.8.4.8  | ExtendedCommunityAccessList | 423 |
| 14.8.5    | OSPF Elements               | 424 |
| 14.8.5.1  | OSPFArea                    | 426 |
| 14.8.5.2  | OSPFKeyChain                | 428 |
| 14.8.5.3  | OSPFProfile                 | 429 |
| 14.8.5.4  | OSPFDomainSetting           | 431 |
| 14.8.5.5  | OSPFInterfaceSetting        | 432 |
| 14.9      | Policies                    | 433 |
| 14.9.1    | InterfacePolicy             | 435 |
| 14.9.2    | FileFilteringPolicy         | 437 |
| 14.9.3    | FirewallPolicy              | 439 |
| 14.9.4    | InspectionPolicy            | 442 |
| 14.9.5    | IPSPolicy                   | 443 |
| 14.9.6    | Layer2Policy                | 446 |
| 14.9.7    | QoSPolicy                   | 449 |
| 14.10     | Sub Policies                | 449 |
| 14.10.1   | FirewallSubPolicy           | 449 |
| 14.11     | Rules                       | 450 |
| 14.11.1   | Rule                        | 450 |
| 14.11.1.1 | IPv4Rule                    | 453 |
| 14.11.1.2 | IPv4Layer2Rule              | 456 |
| 14.11.1.3 | EthernetRule                | 457 |
| 14.11.1.4 | IPv6Rule                    | 459 |
| 14.11.2   | NATRule                     | 459 |
| 14.11.2.1 | IPv4NATRule                 | 461 |
| 14.11.2.2 | IPv6NATRule                 | 464 |
| 14.11.3   | RuleElements                | 464 |
| 14.11.3.1 | Source                      | 466 |
| 14.11.3.2 | Destination                 | 466 |
| 14.11.3.3 | Service                     | 466 |
| 14.11.3.4 | Action                      | 466 |
| 14.11.3.5 | ConnectionTracking          | 469 |
| 14.11.3.6 | LogOptions                  | 470 |
| 14.11.3.7 | AuthenticationOptions       | 473 |
| 14.11.3.8 | MatchExpression             | 473 |
| 14.11.4   | NATElements                 | 475 |
| 14.11.4.1 | DynamicSourceNAT            | 476 |
| 14.11.4.2 | StaticSourceNAT             | 476 |
| 14.11.4.3 | DynamicSourceNAT            | 476 |
| 14.12     | VPN                         | 477 |
| 14.12.1   | PolicyVPN                   | 477 |
| 14.12.2   | RouteVPN                    | 480 |
| 14.12.3   | Gateways                    | 489 |
| 14.12.3.1 | ExternalGateway             | 489 |
| 14.12.3.2 | ExternalEndpoint            | 491 |
| 14.12.4   | VPNSite                     | 493 |
| 14.12.5   | Other Elements              | 495 |
| 14.12.5.1 | GatewaySettings             | 495 |

|                                         |            |
|-----------------------------------------|------------|
| 14.12.5.2 GatewayNode . . . . .         | 496        |
| 14.12.5.3 GatewayProfile . . . . .      | 496        |
| 14.12.5.4 VPNProfile . . . . .          | 497        |
| 14.12.5.5 GatewayTreeNode . . . . .     | 499        |
| 14.12.5.6 GatewayTunnel . . . . .       | 499        |
| 14.12.5.7 ConnectionType . . . . .      | 501        |
| 14.12.5.8 VPNBrokerDomain . . . . .     | 501        |
| 14.12.5.9 VPNBrokerConfigFile . . . . . | 502        |
| 14.13 Collections Reference . . . . .   | 502        |
| 14.13.1 ElementCollection . . . . .     | 503        |
| 14.13.2 SubElementCollection . . . . .  | 509        |
| 14.13.2.1 CreateCollection . . . . .    | 511        |
| 14.13.2.2 RuleCollection . . . . .      | 512        |
| 14.13.3 Search . . . . .                | 512        |
| 14.13.4 BaseIterable . . . . .          | 513        |
| 14.13.5 SerializedIterable . . . . .    | 514        |
| 14.14 Monitoring . . . . .              | 515        |
| 14.15 Advanced Usage . . . . .          | 522        |
| 14.15.1 SMCRequest . . . . .            | 522        |
| 14.15.2 SMCResult . . . . .             | 522        |
| 14.16 Waiters . . . . .                 | 523        |
| 14.17 Exceptions . . . . .              | 525        |
| <b>15 Indices and tables</b>            | <b>531</b> |
| <b>Python Module Index</b>              | <b>533</b> |
| <b>Index</b>                            | <b>535</b> |



Contents:



## **INTRODUCTION**

This is the `smc-python` library to interface with the Forcepoint Flexedge Secure SDWAN Manager.

This acts as an front-end to simplify interactions and simplify scripting when looking to integrate automated functionality.

The `smc-python` library also has a CLI that provides a command completion syntax to provide guidance on commands to be run, and can be run remotely from the Forcepoint Flexedge Secure SDWAN Engines. All actions interact with the Flexedge Secure SDWAN Manager (SMC), and commands specific to the Flexedge Secure SDWAN Engines are proxied by the Management Server component of the SMC to the individual devices.

Current versions are validated using:

- Forcepoint NGFW Management Center  $\geq 6.0$
- Python  $\geq 3.5$
- Requests  $\geq 2.31.0$
- Websocket-client  $\geq 1.5.0$



## INSTALLATION

Install the package by using a package manager such as pip.

```
pip install fp-ngfw-smc-python
```

Or optionally clone: `smc-python`:

```
python setup.py install
```

Dependencies on this library are:

- requests (REST calls)
- websocket-client (websocket calls for smc-monitoring)

If installation is required on a non-internet facing machine, you will have to download `smc-python` and dependencies manually and install by running `python setup install`.

Once the `smc-python` package has been installed, you can import the main packages into a python script:

```
import smc.elements
import smc.core.engine
import smc.core.engines
import smc.policy
import smc.elements.system
```

To remove the package, simply run:

```
pip uninstall fp-NGFW-SMC-python
```

For more information on next steps, please see creating the session



## ENABLE API ON SMC

In order to allow inbound API connections to the SMC, you must first enable the API service on the SMC management server. To do this, open the SMC and edit the properties of the Management Server/s.

Under API Client, enable the API. If SSL connections are required, import or self sign (SMC  $\geq$  6.4) a certificate for use with the API service.

|                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Warning:</b> Do not check the “Use SSL for session ID” parameter when using this library. This setting is incompatible as the sessions are tracked using client sessions (for both HTTP and HTTPS)</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



## CREATING THE SESSION

In order to interact with the SMC REST API, you must first obtain a valid login session. The session is generated by authenticating an API Client and the associated authentication key.

Once the login session has been retrieved successfully, all commands or controls will reuse the same session.

When exiting, call *smc.api.web.logout()* to remove the active session from the Management Server.

---

**Note:** Idle API sessions will still time out after a default timeout on the Management Server.

---

Steps to enable API Communication on the Management Server:

1. Enable SMC API service on the properties of the Management Server
2. Create an API Client and obtain the 'authentication key'

### 4.1 Configuring credentials

Credentials to obtain a session are obtained using the following methods (in order):

- Provide credentials in session.login() constructor
- In alternate specified file path (specified in login constructor)
- In INI configured file at users ~/.smcrc
- Environment variables

Each method is described in more detail below.

#### 4.1.1 Method parameters

Example of providing the connect information through method parameters:

```
from smc import session

session.login(url='http://1.1.1.1:8082', api_key='xxxxxxxxxxxxxxxxxx')
....do stuff....
session.logout()
```

If a specific API version is requested, you can add the following argument to the login constructor. Otherwise the latest API version available will be used.

To find supported versions using unauthenticated call to SMC API:

```
>>> from smc.api.session import available_api_versions
>>> available_api_versions('http://1.1.1.1:8082')
[5.1, 6.1, 6.2]
```

Set up connection to specific version:

```
from smc import session
session.login(url='http://1.1.1.1:8082', api_key='xxxxxxxxxxxxxxxxxx',
             api_version='6.1')
```

Logging in to a specific Admin Domain:

```
session.login(url='http://1.1.1.1:8082', api_key='xxxxxxxxxxxxxxxxxx',
             domain='mydomain')
```

---

**Note:** If an admin domain is specified but the SMC does not have domains configured, you will be placed in the 'Shared Domain'.

---

In order to use SSL connections, you must first associate a private key and certificate with the SMC API server. This is done under the Management Server properties and SMC API. Obtain the certificate for use by the client. It is recommended to ensure your certificate has the subjectAltName field set per RFC 2818.

Using SSL and specify certificate for verifying:

```
from smc import session
session.login(url='https://1.1.1.1:8082', api_key='xxxxxxxxxxxxxxxxxx',
             verify='/Users/username/home/mycacert.pem')
```

Using SSL to the SMC API without SSL validation (NOT recommended)

```
from smc import session
session.login(url='https://1.1.1.1:8082', api_key='xxxxxxxxxxxxxxxxxx',
             verify=False)
```

See also:

`smc.api.session.Session.login()` for constructor arguments.

## 4.1.2 Configuration File

It is possible to store the SMC connection information in `~/.smcrc` in order to simplify the login and eliminate the need to populate scripts with api key information. Syntax for `~/.smcrc`:

```
[smc]
smc_address=1.1.1.1
smc_apikey=xxxxxxxxxxxxxxxxxx
api_version=6.1
smc_port=8082
smc_ssl=True
verify_ssl=True
ssl_cert_file='/Users/username/home/mycacert.pem'
domain=mydomain
```

Then from launching scripts, you can do:

```
session.login()
session.logout()
```

**Note:** It is possible to override the location of `.smrc` by using the `'alt_filepath'` argument in the login constructor.

```
session.login(alt_filepath='/home/somedir/test')
```

### 4.1.3 Environment Variables

If setting environment variables, the following are supported:

```
SMC_ADDRESS=http://1.1.1.1:8082
SMC_API_KEY=123abc
SMC_CLIENT_CERT=path/to/cert
SMC_TIMEOUT = 30 (seconds)
SMC_API_VERSION = 6.1 (optional - uses latest by default)
SMC_DOMAIN = name of domain, Shared is default
```

The minimum variables that need to be present are `SMC_ADDRESS` and `SMC_API_KEY`:

```
export SMC_ADDRESS = http://1.1.1.1:8082
export SMC_API_KEY = foobarkey
```

Based on the session login constructor, you can also pass kwargs using the parameter `SMC_EXTRA_ARGS`.

Once the session has been successfully obtained, there is no reason to re-authenticate a new session unless `logout` has been called.

**Note:** The SMC API will automatically purge idle sessions after a configurable amount of time.

## 4.2 Handling retries on server busy

It is possible to override the default behavior for retrying a CRUD operation based on receiving a “Service Unavailable” (HTTP 503) response. By default, no retry is attempted. You can override this behavior and allow the API to retry an operation using a backoff algorithm.

This can be enabled through the session login constructor using the `retry_on_busy` boolean or after session login by calling `set_retry_on_busy`. If called from session login, default parameters are provided for all retry related settings. If you require more granularity, call after session login.

**Note:** By default, the following operation types are eligible for retry (GET/POST/PUT). You can override this by calling `session.set_retry_on_busy(method_whitelist=['GET', 'POST', 'DELETE'])`

Calling from session login:

```
session.login(url='https://x.x.x.x:8082', api_key='xxxxxxxxxxxxxxxx',
              verify=False, timeout=30, retry_on_busy=True)
```

Calling after session login:

```
session.login()
session.set_retry_on_busy(total=5, backoff_factor=0.1)
...
session.logout()
```

If you are using an preferences file, place the following into your `.smcrc`:

```
[smc]
retry_on_busy=True
```

You can also set this on as an environment variable using the `SMC_EXTRA_ARGS` variable:

```
os.environ['SMC_EXTRA_ARGS'] = '{"retry_on_busy": "True"}'
```

## 4.3 Handling proxies

To disable the use of an intermediate proxy and force the connection to go direct, you can add the following environment variable:

```
os.environ['no_proxy'] = 'my.smc.at.domain'
```

## 4.4 Logging helper

To enable logging from `smc-python`, you can utilize the standard python logger or use convenience methods provided. These are typically called before session login:

```
from smc import set_file_logger
set_file_logger(log_level=10, path='/Users/foo/smc-test.log')
...
```

Or use a stream logger and also optionally enable `urllib3` messages:

```
from smc import set_stream_logger
set_stream_logger(log_level=logging.DEBUG)
set_stream_logger(log_level=logging.DEBUG, logger_name='urllib3')
```

Another logging option is to add the following lines to your script:

```
import logging
logging.getLogger()
logging.basicConfig(
    level=logging.DEBUG, format='%(asctime)s %(levelname)s %(name)s.%(funcName)s:
    ↳ %(message)s')
```

The format parameter follows the standard python logging module syntax.

## RESOURCES

Resources in the SMC are typically accessed in a couple different ways.

The first would be by using the elements collection interface to search for elements of a specific type.

For example, if you are looking for Hosts by a given IP address:

```
>>> from smc.elements.network import Host
>>> list(Host.objects.filter('192.168'))
[Host(name=aws-192.168.4.254), Host(name=host-192.168.4.135), Host(name=host-192.168.4.
↳ 94), Host(name=host-192.168.4.79)]
```

See *Collections* for more information on search capabilities.

It is also possible to access resources directly:

```
>>> from smc.core.engine import Engine
>>> engine = Engine('sg_vm')
>>> print(list(engine.nodes))
[Node(name=ngf-1065), Node(name=ngf-1035)]

>>> print(list(engine.routing))
[Routing(name=Interface 0,level=interface), Routing(name=Interface 1,level=interface),
↳ Routing(name=Interface 2,level=interface), Routing(name=Tunnel Interface 2000,
↳ level=interface), Routing(name=Tunnel Interface 2001,level=interface)]
```

Retrieving a specific host element by name:

```
>>> from smc.elements.network import Host
>>> host = Host('kali')
>>> print(host.href)
http://172.18.1.150:8082/6.2/elements/host/978
```

When elements are referenced initially, they are lazy loaded until attributes or methods of the element are used that require the *data* attribute. Once an element has been ‘inflated’ due to a reference being called (property, method, etc), the resultant element data is stored in a per instance cache. The *data* attribute is the per instance cache as retrieved from the Management Server and will also include the Etag for the request.

Example of how elements are lazy loaded:

```
>>> from smc.elements.network import Host
>>> host = Host('kali')
>>> vars(host)
{'_meta': None, '_name': 'kali'}          #Base level attributes, only instance created
```

(continues on next page)

(continued from previous page)

```
>>> host.href # Call to retrieve this resource link reference loads instance meta (1 SMC query)
u'http://172.18.1.150:8082/6.2/elements/host/978'
>>> vars(host)
{'_meta': Meta(name=u'kali', href=u'http://172.18.1.150:8082/6.2/elements/host/978', type=u'host'), '_name': 'kali'}
>>> host.data # Request to a method/attribute that requires the data attribute inflates the instance (1 SMC query)
{'comment': u'this is a searchable comment', 'read_only': False, 'ipv6_address': u'2001:db8:85a3::8a2e:370:7334', 'name': u'kali', 'third_party_monitoring': {'netflow': False, 'snmp_trap': False}, 'system': False, 'link': [{'href': u'http://172.18.1.150:8082/6.2/elements/host/978', 'type': u'host', 'rel': u'self'}, {'href': u'http://172.18.1.150:8082/6.2/elements/host/978/export', 'rel': u'export'}, {'href': u'http://172.18.1.150:8082/6.2/elements/host/978/search_category_tags_from_element', 'rel': u'search_category_tags_from_element'}], 'key': 978, 'address': u'1.1.11.1', 'secondary': [u'7.7.7.7']}
>>> vars(host.data)
{'data': {'comment': u'this is a searchable comment', 'read_only': False, 'ipv6_address': u'2001:db8:85a3::8a2e:370:7334', 'name': u'kali', 'third_party_monitoring': {'netflow': False, 'snmp_trap': False}, 'system': False, 'link': [{'href': u'http://172.18.1.150:8082/6.2/elements/host/978', 'type': u'host', 'rel': u'self'}, {'href': u'http://172.18.1.150:8082/6.2/elements/host/978/export', 'rel': u'export'}, {'href': u'http://172.18.1.150:8082/6.2/elements/host/978/search_category_tags_from_element', 'rel': u'search_category_tags_from_element'}], 'key': 978, 'address': u'1.1.11.1', 'secondary': [u'7.7.7.7']}, '_meta': Meta(name=u'kali', href=u'http://172.18.1.150:8082/6.2/elements/host/978', type=u'host'), '_name': 'kali'}
```

At most 2 queries will be required to retrieve an element as a resource.

Cache contents can be viewed in their raw json format by calling the 'data' property.

---

**Note:** When modifications are made to a specific element, they are submitted back to the Management Server using the originally retrieved ETag to ensure the element has not been modified since the original retrieval.

---

## COLLECTIONS

Resource collections are designed to be similar to how Django query sets work and provide a similar API.

### 6.1 ElementCollection

ElementCollections are available on all elements that inherit from `smc.base.model.Element`, and are also available for general searching across any element with an SMC API entry point.

An `ElementCollection` can be constructed without making a single query to the Management Server database. No query will occur until you do something to evaluate the collection.

You can evaluate a collection in the following ways:

- **Iteration.** An `ElementCollection` is iterable, and it executes the Management Server query the first time you iterate over it. For example, this will retrieve all host elements:

```
>>> for host in Host.objects.all():  
...     print(host.name, host.address)
```

- **list().** Force evaluation of a collection by calling `list()` on it:

```
>>> elements = list(Host.objects.all())
```

- **first().** Helper collection method to retrieve only the first element in the search query:

```
>>> host = Host.objects.iterator()  
>>> host.first()  
Host(name=SMC)
```

If you don't need all results and only a single element, rather than getting an `ElementCollection` iterator, you can obtain this directly from the `CollectionManager`:

```
>>> Host.objects.first()  
Host(name=SMC)
```

- **last().** Helper collection method to retrieve only the last element in the search query:

```
>>> host = Host.objects.iterator()  
>>> host.last()  
Host(name=kali3)
```

- **exists().** Helper collection method to evaluate whether there are results:

```
>>> hosts = Host.objects.filter('1.1.1.1')
>>> if hosts.exists():
...     for host in list(hosts):
...         print(host.name, host.address)
...
('hax0r', '1.1.1.1')
('host', '1.1.1.1')
('hostelement', '1.1.1.1')
('abcdefghijklmnop', '1.1.1.1')
```

- `count()`. Helper collection method which returns the number of results. You can still obtain the results after:

```
>>> it = Router.objects.iterator()
>>> query1 = it.filter('10.10.10.1')
>>> query1.count()
3
>>> list(query1)
[Router(name=Router-10.10.10.10), Router(name=Router-10.10.10.10), ↵
↵Router(name=Router-10.10.10.1)]
```

- `batch()`. Iterator returning batches of results with specific by quantity. If `limit()` is also chained, it is ignored as batch and limit are mutually exclusive operations.

```
>>> for hosts in Host.objects.batch(2):
...     print(hosts)
...
[Host(name=SMC), Host(name=172.18.1.135)]
[Host(name=172.18.2.254), Host(name=host)]
[Host(name=host-54.76.110.156), Host(name=host-192.168.4.135)]
[Host(name=external primary DNS resolver), Host(name=host-192.168.4.94)]
...
```

### 6.1.1 Methods that return a new ElementCollection

There are multiple methods in an `ElementCollection` that allow you to refine how the query or results are returned. Each chained method returns a new `ElementCollection` with aggregated search parameters.

- `filter()`. Provide a filter string to narrow the search to a string value that will be used in a ‘contains’ match:

```
>>> host = Host.objects.filter('172.18.1')
>>> list(host)
[Host(name=172.18.1.135), Host(name=SMC)]
```

`filter` can also take a keyword argument to filter specifically on an attribute. The keyword argument should match a valid attribute for the element type, and value to match:

```
>>> list(Router.objects.filter(address='10.10.10.1'))
[Router(name=Router-10.10.10.1)]
```

---

**Note:** Two additional keyword arguments can be passed to `filter`, `exact_match=True` and/or `case_sensitive=False`.

---

- `limit()`. Limit the number of results to return.

```
>>> list(Host.objects.all().limit(3))
[Host(name=SMC), Host(name=172.18.1.135), Host(name=172.18.2.254)]
```

- `all()`. Return all results.

```
>>> list(Host.objects.all())
```

## 6.1.2 Basic rules on searching

- By default searches use a ‘contains’ logic. If you specify a filter string, the SMC API will return elements that contain that string. Therefore, if partial searches are performed, you may receive multiple matches:

```
>>> list(Router.objects.filter('10.10'))
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10),
 Router(name=Router-10.10.10.1)]
```

- When the search is evaluated, the elements returned contain only meta data and not the full payload for each element matching the search. The search query is built based on provided parameters to narrow the scope and only a single query is made to the Management Server.
- When using a filter, the SMC API will search the name, comment and relevant field/s for the element type selected.

Each element type will have it’s own searchable fields. For example, in addition to the name and comment field, a Host element will search the address and secondary address fields. This is automatic.

For example, the following would find Host elements with this value in any of the Host fields specified above:

```
>>> Host.objects.filter('111.111.111.111')
```

- Setting `exact_match=True` on the filter query will only match on an element’s name or comment field and is a case sensitive match. The SMC API is case sensitive, so unless you need an element by exact case, this field is not required. By default, `exact_match=False`.
- In v0.5.6, `case_sensitive=False` can be set on the filter query to change the behavior of case sensitive matches. If not set, `case_sensitive=True`.
- Using a keyword argument with ‘filter’ will provide element introspection against the attributes to perform an exact match. In general, using a kwarg is most effective when searching for network elements. Since the default search is a ‘contains’ match, a search for ‘10.10.10.1’ may return elements with values: ‘10.10.10.1’, ‘10.10.10.10’, and ‘110.10.10.1’. Using an attribute/value would override the default search behavior and attempt to only match on the specified attribute:

```
>>> list(Router.objects.filter('10.10.10.1'))
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10),
 Router(name=Router-10.10.10.1)]
```

The above query returns multiple elements contains matches. To explicitly define the attribute to make an exact match, change the filter to use a kwarg (the address attribute is the defined ipaddress for `smc.elements.network.Router`):

```
>>> list(Router.objects.filter(address='10.10.10.1'))
[Router(name=Router-10.10.10.1)]
```

**Note:** When using keyword matching with `filter`, a single query will be performed using the attribute value, returning a list of ‘contains’ matches. For each element match returned from the first query, an additional query is performed to retrieve the element attributes.

---

To reduce the number of additional queries performed when using keyword matching, use a limit on the number of return elements:

```
>>> list(Router.objects.filter(address='10.10.10.1').limit(1))
[Router(name=Router-10.10.10.1)]
```

## 6.1.3 Additional Examples

Obtain an iterator from the collection manager for re-use:

```
>>> iterator = Router.objects.iterator()
>>> query1 = iterator.filter('10.10.10.1')
>>> list(query1)
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10), Router(name=Router-
↪10.10.10.1)]
>>> query2 = query1.filter(address='10.10.10.1')
>>> list(query2)
[Router(name=Router-10.10.10.1)]
```

Access a collection directly on an Element type:

```
>>> list(Host.objects.all())
[Host(name=SMC), Host(name=172.18.1.135), Host(name=172.18.2.254), Host(name=host)]
...
>>> list(TCPService.objects.filter('HTTP'))
[TCPService(name=HTTPS_No_Decryption), TCPService(name=Squid HTTP proxy),
↪TCPService(name=HTTP to Web SaaS)]
```

Limit number of return entries:

```
>>> list(Host.objects.limit(3))
[Host(name=SMC), Host(name=172.18.1.135), Host(name=172.18.2.254)]
```

Limit and filter the results using a chainable syntax:

```
>>> list(Host.objects.filter('172.18.1').limit(5))
[Host(name=172.18.1.135), Host(name=SMC), Host(name=TIE Server), Host(name=172.18.1.93)]
```

Get a host collection when partial IP address known:

```
>>> list(Host.objects.filter('192.168'))
[Host(name=aws-192.168.4.254), Host(name=host-192.168.4.135), Host(name=host-192.168.4.
↪94), Host(name=host-192.168.4.79)]
```

When filtering is performed, by default search queries will ‘wildcard’ the results. To only return an exact match of the search query, use the optional flag ‘exact\_match’:

```
>>> list(TCPService.objects.filter('8080'), exact_match=True))
[TCPService(name=TCP_8080), TCPService(name=HTTP proxy), TCPService(name=SSH),
↪ TCPService(name=SSM SSH)]
```

Additional convenience functions are provided on the collections to simplify navigating through results such as `count`, `first`, and `last`:

```
>>> query1 = iterator.filter('10.10.10.1')
>>> if query1.exists():
...     list(query1.all())
...
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10), Router(name=Router-
↪ 10.10.10.1)]

>>> list(query1)
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10), Router(name=Router-
↪ 10.10.10.1)]
>>> query1.first()
Router(name=Router-110.10.10.10)
>>> query1.last()
Router(name=Router-10.10.10.1)
>>> query1.count()
3
>>> query2 = query1.filter(address='10.10.10.1') # Add kwarg to new query
>>> list(query2)
[Router(name=Router-10.10.10.1)]
```

## 6.2 General Search

If a search is required for an element type that is not a pre-defined class of `smc.base.model.Element` type in the API, it is still possible to search any valid entry point using `smc.base.collections.Search`.

Search extends `ElementCollection` and provides additional methods:

- `entry_point()`. Entry points are top level collections available from the SMC API.
- `context_filter()`. Context filters are special filters that can return more generalized results such as all engines, etc.

Available context filters:

- `fw_clusters` - list all firewalls
- `engine_clusters` - all clusters
- `ips_clusters` - ips only clusters
- `layer2_clusters` - layer2 only clusters
- `network_elements` - all network element types
- `services` - all service types
- `services_and_applications` - all services and applications
- `tags` - element tags
- `situations` - inspection situations

- `unused()`. Search for all unused elements:

```
>>> list(Search.objects.unused())
[RouteVPN(name=myvpn), RouteVPN(name=mygre), RouteVPN(name=avpn),
↪RouteVPN(name=avpn)]
...
```

- `duplicates()`. Search for all duplicate elements:

```
>>> list(Search.objects.duplicates())
[Host(name=foohost), Router(name=router-1.1.1.1)]
...
```

Using Search is useful if there is not a direct class representation of the element you are attempting to retrieve. If there is an entry point for the target element type, you can return any element.

First, find all available searchable objects (also known as ‘entry points’):

```
>>> from smc.elements.resources import Search
>>> Search.object_types()
['elements', 'sub_ipv6_fw_policy', 'ids_alert', 'application_not_specific_tag', 'fw_alert
↪', 'virtual_ips', 'sidewinder_tag', 'os_specific_tag', 'eia_application_usage_group_tag
↪', 'external_bgp_peer', 'local_cluster_cvi_alias', 'ssl_vpn_service_profile', 'active_
↪directory_server', 'eia_golden_image_tag', 'client_gateway', 'situation_tag', 'api_
↪client', 'tls_match_situation', 'ssl_vpn_policy', 'category_group_tag', 'ip_list',
↪'vpn_profile', 'ipv6_access_list', 'appliance_information', 'single_layer2', 'ei_
↪executable', 'community_access_list']
...
```

Once the type of interest is found, the elements can be retrieved using the entry point:

```
>>> list(Search.objects.entry_point('vpn'))
[PolicyVPN(name=Amazon AWS), PolicyVPN(name=sg_vm_vpn), PolicyVPN(name=TRITON AP-WEB_
↪Cloud VPN)]
```

And subsequently add a filter as well:

```
>>> list(Search.objects.entry_point('vpn').filter('AWS'))
[PolicyVPN(name=Amazon AWS)]
```

---

Additional examples:

Searching all services for port 80:

```
>>> list(Search.objects.entry_point('services').filter('80'))
[TCPService(name=tcp80443), TCPService(name=HTTP to Web SaaS), EthernetService(name=IPX_
↪over Ethernet 802.2), UDPService(name=udp_10070-10080), Protocol(name=HTTP8080),
↪TCPService(name=tcp_10070-10080), TCPService(name=TCP_8080), TCPService(name=tcp_3478-
↪3480), EthernetService(name=IPX over Ethernet 802.3 (Novell)), TCPService(name=HTTP),
↪TCPService(name=SSM HTTP), TCPService(name=HTTP (SafeSearch)), IPService(name=ISO-IP),
↪UDPService(name=udp_3478-3480), TCPService(name=HTTP (with URL Logging))]
```

Only Network elements with ‘172.18.1’:

```
>>> list(Search.objects.context_filter('network_elements').filter('172.18.1'))
[Host(name=172.18.1.135), Host(name=SMC), Network(name=Any network),
↳ FirewallCluster(name=sg_vm), Element(name=dc-smtp), Network(name=network-172.18.1.0/
↳ 24), LogServer(name=LogServer 172.18.1.150), Layer3Firewall(name=testfw),
↳ Element(name=SecurID), Element(name=Windows 2003 DHCP), AddressRange(name=range-172.18.
↳ 1.100-172.18.1.120), ManagementServer(name=Management Server)]
```

Only firewall clusters:

```
>>> list(Search.objects.context_filter('fw_clusters'))
[FirewallCluster(name=sg_vm), Layer3VirtualEngine(name=ve-8),
↳ Layer3Firewall(name=testfw), Layer3Firewall(name=i-04eec8f019adf818e (us-east-2a)),
↳ MasterEngine(name=master)]
```

In addition to using more generic filters, with general searches, you can also specify multiple valid entry points by specifying the string filter comma separated.

For example, finding all hosts and routers:

```
>>> list(Search.objects.entry_point('router,host'))
[Host(name=172.18.2.254), Router(name=router-172.18.3.129), Host(name=All Routers (Site-
↳ Local))]
```

Filter based on hosts and routers:

```
>>> list(Search.objects.entry_point('router,host').filter('172.18.1'))
[Host(name=172.18.1.135), Host(name=SMC), Host(name=ePolicy Orchestrator),
↳ Router(name=router-172.18.1.225), Host(name=fw-internal-primary), Router(name=router-
↳ 172.18.1.209)]
```

---

**Note:** If an element of class `smc.base.model.Element` exists, it will be returned as that type to enable access to the objects instance methods. If there is no element defined, a dynamic class is produced from type `Element`.

---

For example, searching for object of type 'ids\_alert' will produce a dynamic class as type `Element` and will have access to the base class methods:

```
>>> list(Search.objects.entry_point('ids_alert'))
[IdsAlertDynamic(name=Default alert), IdsAlertDynamic(name=Test alert),
↳ IdsAlertDynamic(name=System alert)]
```

Classes deriving from `smc.base.model.Element` are found in the API reference, for example: [Administration](#)



## ELEMENTS

Elements are the building blocks for policy and include types such as Networks, Hosts, Services, Groups, Lists, Zones, etc.

### 7.1 Create

Elements within the Management Server are common object types that are referenced by other configurable areas of the system such as policy, routing, VPN, etc.

This is not an exhaustive list, all supported element types can be found in the API reference documentation: [Administration](#)

- *Hosts*
- *AddressRange*
- *Networks*
- *Routers*
- *Groups*
- *DomainName*
- *IPList* (SMC API >= 6.1)
- *URLListApplication* (SMC API >= 6.1)
- *Zone*
- *LogicalInterface*
- *TCPService*
- *UDPService*
- *IPService*
- *EthernetService*
- *ServiceGroup*
- *TCPServiceGroup*
- *UDPServiceGroup*
- *IPServiceGroup*
- *RPCService*
- *ICMPService*

- *ICMPv6Service*

Oftentimes these objects are cross referenced within the configuration, like when creating rule or NAT policy. All calls to `create()` will return the href of the new element stored in the Management Server database or will raise an exception for failure.

Examples of creating elements are as follows:

```
>> from smc.elements.network import Host, Network, AddressRange
>>> host = Host.create(name='hostelement', address='1.1.1.1')
>>> host
Host(name=hostelement)
>>> host.address
u'1.1.1.1'
>>> network = Network.create(name='networkelement', ipv4_network='1.1.1.0/24', comment=
↳ 'mynet')
>>> network
Network(name=networkelement)
>>> network.ipv4_network
u'1.1.1.0/24'
>>> network.comment
u'mynet'
>>> AddressRange.create(name='myaddrrange', ip_range='1.1.1.1-1.1.1.10')
AddressRange(name=myaddrrange)
```

Check the various reference documentation for defined elements supported.

## 7.2 Update

Updating elements can be done in multiple ways. In most cases, making modifications to an element through methods or element attributes are the preferred way. Modifications done through existing methods/attributes are done idempotent to the elements cache. In order to commit these changes to the Management Server database, calling `.update()` is required unless explicitly documented otherwise.

---

**Note:** There are some edge cases where `.update()` is called automatically like when modifying interfaces where multiple areas are updated. These will be documented on the method.

---

Another way to update an element is by providing the kwarg values in the `update()` call directly.

For example, setting the address, secondary address and comment for a host element can be done in update by providing kwargs:

```
host = Host('kali')
host.update(
    address='3.3.3.3',
    secondary=['12.12.12.12'],
    comment='something about this host')
```

A much more low-level way of modifying an element is to modify the data in cache (dict) directly. After making the modifications, you must also call `.update()` to submit the change.

Modifying a service element after reviewing the element cache:

```

>>> service = TCPService.create(name='aservice', min_dst_port=9090)
>>> service
TCPService(name=aservice)
...
>>> pprint(vars(service.data))
{'u'key': 3551,
 u'link': [{u'href': u'http://172.18.1.150:8082/6.2/elements/tcp_service/3551',
            u'rel': u'self',
            u'type': u'tcp_service'},
           {u'href': u'http://172.18.1.150:8082/6.2/elements/tcp_service/3551/export',
            u'rel': u'export'},
           {u'href': u'http://172.18.1.150:8082/6.2/elements/tcp_service/3551/search_
↪category_tags_from_element',
            u'rel': u'search_category_tags_from_element'}]],
 u'min_dst_port': 9090,
 u'name': u'aservice',
 u'read_only': False,
 u'system': False}
...
>>> service.data['min_dst_port'] = 9091
>>> service.update()      # Submit to SMC, cache is refreshed
'http://172.18.1.150:8082/6.2/elements/tcp_service/3551'
...
>>> pprint(vars(service.data))
{'u'key': 3551,
 u'link': [{u'href': u'http://172.18.1.150:8082/6.2/elements/tcp_service/3551',
            u'rel': u'self',
            u'type': u'tcp_service'},
           {u'href': u'http://172.18.1.150:8082/6.2/elements/tcp_service/3551/export',
            u'rel': u'export'},
           {u'href': u'http://172.18.1.150:8082/6.2/elements/tcp_service/3551/search_
↪category_tags_from_element',
            u'rel': u'search_category_tags_from_element'}]],
 u'min_dst_port': 9091,
 u'name': u'aservice',
 u'read_only': False,
 u'system': False}

```

Attributes supported by elements are documented in the API Reference: [Administration](#)

## 7.3 Delete

Deleting elements is done by using the base class delete method. If the element has already been fetched, the ETag of the original fetch is stored with the element cache and will be provided during the delete.

Deleting a host:

```

>>> from smc.elements.network import Host
>>> Host('kali').delete()

```

## 7.4 Functions or methods that modify

Some functions or element methods may make modifications to an element depending on the operation. These functions are documented and will also be decorated with and `autocommit` decorator. This allows you to queue changes locally before submitting them to the Management Server by calling `update`. To override this behavior, you can either pass `autocommit=True` to these functions or set `session.AUTOCOMMIT=True` on the session. Most methods will `autocommit` by default with exception of methods defined in `smc.core.properties`.

## ENGINES

Engines are the definitions for a Single Firewall, Layer 2 Firewall, IPS, Firewall Cluster, Master NGFW Engine, or Virtual NGFW Engine.

An engine defines the basic settings to make the device or virtual instance operational such as interfaces, routes, ip addresses, networks, dns servers, etc.

Creating engines are done using the Firewall specific base classes in [smc.core.engines](#)

Nodes are individual devices represented as properties of an engine element. In the case of single device deployments, there is only one node. For clusters, there will be at a minimum 2 nodes, max of 16. The [smc.core.node](#) class represents the interface to managing and sending commands individually to a node in a cluster.

By default, each constructor will have default values for the interface used for management (interface 0). This can be overridden as necessary.

Once engines are created, they can be retrieved directly by using [smc.core.engine.Engine](#) or directly by their engine type. The `__repr__` of Engine will show a descriptive view of the engine type regardless of how the context was obtained.

```
>>> Engine('sg_vm')
FirewallCluster(name=sg_vm)
...
>>> from smc.core.engines import FirewallCluster
>>> FirewallCluster('sg_vm')
FirewallCluster(name=sg_vm)
```

---

**Note:** There is no difference between the two options. Loading from the Engine class tends to be easier as you are not required to know the engine type to obtain the context.

---

## 8.1 Create

### 8.1.1 Layer3 Firewall

For Layer 3 single firewall engines, the minimum requirements are to specify a name, management IP and management network. By default, the Layer 3 firewall will use interface 0 as the management port. This can be overridden in the constructor if a different interface is required.

To create a Single Firewall:

```
>>> from smc.core.engines import Layer3Firewall
>>> Layer3Firewall.create(name='firewall', mgmt_ip='1.1.1.1', mgmt_network='1.1.1.0/24')
Layer3Firewall(name=firewall)
```

See reference for more information: [`smc.core.engines.Layer3Firewall`](#)

## 8.1.2 Layer 2 Firewall

For Layer 2 Firewall and IPS engines, an inline interface pair will automatically be created using interfaces 1-2 but can be overridden in the constructor to use different interface mappings. At least one inline pair or a capture interface is required to successfully create.

Creating a Layer2 Firewall with alternative management interface and DNS settings:

```
>>> from smc.core.engines import Layer2Firewall
>>> Layer2Firewall.create(name='myfirewall', mgmt_ip='1.1.1.1', mgmt_network='1.1.1.0/24',
↳ mgmt_interface=5, domain_server_address=['172.18.1.20'])
Layer2Firewall(name=myfirewall)
```

See reference for more information: [`smc.core.engines.Layer2Firewall`](#)

## 8.1.3 IPS Engine

Similar to Layer2Firewall, at least one inline interface pair or a capture interface is required to successfully create.

Use alternative inline interface pair configuration (mgmt on interface 0):

```
>>> from smc.core.engines import IPS
>>> IPS.create(name='myips',
...           mgmt_ip='1.1.1.1',
...           mgmt_network='1.1.1.0/24',
...           inline_interface='5-6')
IPS(name=myips)
```

See reference for more information: [`smc.core.engines.IPS`](#)

## 8.1.4 Master Engine

A Master NGFW Engine is used to manage Virtual NGFW Engine nodes and provides in system virtualization. Master NGFW Engine controls administrative aspects and specifies how resources are allocated to the Virtual NGFW Engines.

Create a Master NGFW Engine with a single management interface, then add 2 more physical interface for Virtual NGFW Engine allocation:

```
>>> from smc.core.engines import MasterEngine
>>> engine = MasterEngine.create(name='api-master',
...                             mgmt_ip='1.1.1.1',
...                             mgmt_network='1.1.1.0/24',
...                             master_type='firewall',
...                             domain_server_address=['8.8.4.4', '7.7.7.7'])
>>> print(engine)
>>> MasterEngine(name=api-master)
```

(continues on next page)

(continued from previous page)

```
>>> engine.physical_interface.add(1)    # add interfaces
>>> engine.physical_interface.add(2)
>>> for intf in engine.interface.all():
...     print(intf)
...
PhysicalInterface(name=Interface 1)
PhysicalInterface(name=Interface 0)
PhysicalInterface(name=Interface 2)
```

See `smc.core.engines.MasterEngine` for more details.

### 8.1.5 Virtual Engine

A Virtual Firewall is a host that resides on a Master NGFW Engine node used for multiple firewall contexts. The Management Server maps a ‘virtual resource’ to a Virtual NGFW Engine as a way to map the Master NGFW Engine interface to the individual instance residing within the physical device.

In order to create a Virtual NGFW Engine, you must first manually create the Master NGFW Engine, then create the interfaces that will be used for the virtual instances.

The first step in creating the Virtual NGFW Engine is to create the virtual resource and map that to a physical interface or VLAN on the Master NGFW Engine. Once that has been created, add IP addresses to the Virtual NGFW Engine interfaces as necessary.

First create the virtual resource on the already created Master NGFW Engine:

```
>>> from smc.core.engines import MasterEngine
>>> engine = MasterEngine('api-master')
>>> engine.virtual_resource.create('ve-1', vfw_id=1)
'http://1.1.1.1:8082/6.1/elements/master_engine/62629/virtual_resource/756'
```

See `smc.core.engine.VirtualResource.create()` for more information.

Creating a Virtual NGFW Engine with two single physical interfaces:

```
>>> from smc.core.engines import Layer3VirtualEngine
>>> Layer3VirtualEngine.create(name='myvirtual',
...                           master_engine='api-master',
...                           virtual_resource='ve-1',
...                           interfaces=[{'address':'5.5.5.5','network_value':'5.5.5.0/
↪24','interface_id':0},
...                                       {'address':'6.6.6.6','network_value':'6.6.6.0/
↪24','interface_id':1}])
Layer3VirtualEngine(name=myvirtual)
```

**Note:** Virtual NGFW Engine interface numbering takes into account the dedicated interface for the Master NGFW Engine. For example, if the Master NGFW Engine is using physical interface 0 for management, the Virtual NGFW Engine may be assigned physical interface 1 for use. From an indexing perspective, the naming within the Virtual NGFW Engine configuration will start at interface 0 but be using physical interface 1.

See reference for more information: `smc.core.engines.Layer3VirtualEngine`

### 8.1.6 Firewall Cluster

Creating a Firewall Cluster requires additional interface related information to bootstrap the engine properly. With NGFW clusters, a “cluster virtual interface” is required (if only one interface is used) to specify the cluster address as well as each engine specific node IP address. In addition, a macaddress is required for packetdispatch functionality (recommended HA configuration).

By default, the FirewallCluster class will allow as many nodes as needed (up to 16 per cluster) for the singular interface. The node specific interfaces are defined by passing in the ‘nodes’ argument to the constructor as follows:

Create a 3 node cluster:

```
>>> from smc.core.engines import FirewallCluster
>>> FirewallCluster.create(name='mycluster',
...                        cluster_virtual='1.1.1.1',
...                        cluster_mask='1.1.1.0/24',
...                        cluster_nic=0,
...                        macaddress='02:02:02:02:02:02',
...                        nodes=[{'address': '1.1.1.2', 'network_value': '1.1.1.0/24',
↪ 'nodeid': 1},
...                               {'address': '1.1.1.3', 'network_value': '1.1.1.0/24',
↪ 'nodeid': 2},
...                               {'address': '1.1.1.4', 'network_value': '1.1.1.0/24',
↪ 'nodeid': 3}],
...                        domain_server_address=['8.8.8.8'])
FirewallCluster(name=mycluster)
```

See `smc.core.engines.FirewallCluster` for more info

### 8.1.7 MasterEngine Cluster

Create a Master NGFW Engine cluster for redundancy. Master NGFW Engine clusters support active/standby mode.

Create the cluster and add a second interface for each cluster node:

```
>>> MasterEngineCluster.create(name='engine-cluster',
...                             master_type='firewall',
...                             macaddress='22:22:22:22:22:22',
...                             nodes=[{'address': '5.5.5.2', 'network_value': '5.5.5.0/24',
↪ 'nodeid': 1},
...                                     {'address': '5.5.5.3', 'network_value': '5.5.5.0/24',
↪ 'nodeid': 2}])
MasterEngine(name=engine-cluster)
```

Adding an interface after creation:

```
>>> from smc.core.engine import Engine
>>> engine = Engine('engine-cluster')
>>> engine.physical_interface.add_cluster_interface_on_master_engine(
...     interface_id=1,
...     macaddress='22:22:22:22:22:33',
...     nodes=[{'address': '6.6.6.2', 'network_value':
↪ '6.6.6.0/24', 'nodeid': 1},
...             {'address': '6.6.6.3', 'network_value':
↪ '6.6.6.0/24', 'nodeid': 2}])
```

See `smc.core.engines.MasterEngineCluster` for more info

## 8.2 Nodes

Managed engines have many options for controlling the behavior of the device or virtual through the SMC API. Once an engine has been created, The engine is represented with ‘nodes’ that map to the individual firewall/IPS’s. For example, a cluster will have 2 or more nodes.

Engine hierarchy resembles the following:

```
Engine
| - ---> Node1
| - ---> Node2
| - ---> Node3
\ - .... (up to 16)
```

Engine level commands allow operations like refresh policy, upload new policy, generating snapshots, export configuration, blacklisting, adding routes, route monitoring, and add or delete a physical interfaces

Some example engine level commands:

```
>>> engine = Engine('testfw')
>>> for node in engine.nodes:
>>> engine.generate_snapshot() #generate a policy snapshot
>>> engine.export(filename='/Users/username/export.xml') #generate policy export
>>> engine.refresh() #refresh policy
>>> engine.routing_monitoring() #get route table status
....
```

For all available commands for engines, see `smc.core.engine.Engine`

Node level commands are specific commands targeted at individual nodes directly. In the case of a cluster, you can control the correct node by iterating `smc.core.engine.Engine.nodes` list.

Node level commands allow actions such as fetch license, bind license, initial contact, appliance status, go online, go offline, go standby, lock online, lock offline, reset user db, diagnostics, reboot, sginfo, ssh (enable/disable/change pwd), and time sync.

View nodes and reboot a node by name:

```
>>> engine = Engine('testfw')
>>> print(engine.nodes)
[Node(name=testfw node 1)]
...
>>> for node in engine.nodes:
...     if node.name == 'testfw':
...         node.reboot()
```

Bind license, then generate initial contact for each node for a specific engine:

```
>>> for node in engine.nodes:
...     node.initial_contact(filename='/Users/username/engine.cfg')
...     node.bind_license()
```

For all available commands for node, see `smc.core.node.Node`

## 8.3 Interfaces

After your engine has been successfully created with the default interfaces, you can add and remove interfaces as needed.

From an interface perspective, there are several different interface types that have subtle differences. The supported physical interface types available are:

- Single Node Dedicated Interface (Single Layer 3 Firewall)
- Node Dedicated Interface (Used on Clusters, IPS, Layer 2 Firewall)
- Inline Interface (IPS / Layer2 Firewall)
- Capture Interface (IPS / Layer2 Firewall)
- Cluster Virtual Interface
- Virtual Physical Interface (used for Virtual NGFW Engines)
- Tunnel Interface

The distinction is subtle but straightforward. A single node interface is used on a single Firewall instance and represents a unique interface with dedicated IP Address.

A node dedicated interface is used on Layer 2 and IPS engines as management based interfaces and may also be used as a heartbeat (for example).

It is a unique IP address for each machine. It is not used for operative traffic in Firewall Clusters, IPS engines, and Layer 2 Firewalls. Firewall Clusters use a second type of interface, Cluster Virtual IP Address (CVI), for operative traffic.

IPS engines have two types of interfaces for traffic inspection: the Capture Interface and the Inline Interface. Layer 2 Firewalls only have Inline Interfaces for traffic inspection.

---

**Note:** When creating your engine instance, the correct type/s of interfaces are created automatically without having to specify the type. However, this may be relevant when adding interfaces to an existing device after creation.

---

To access interface information on existing engines, or to add to an existing engine, you must obtain the engine context object. It is not required to know the engine type (layer3, layer2, ips) as you can load by the parent class `smc.core.engines.Engine`.

For example, if I know I have an engine named 'myengine' (despite the engine 'role'), it can be obtained via:

```
>>> from smc.core.engine import Engine
>>> engine = Engine('sg_vm')
>>> print(engine.nodes)
[Node(name=ngf-1065), Node(name=ngf-1035)]
```

It is not possible to add certain interface types based on the node type. For example, it is not possible to add inline or capture interfaces to Single Firewalls. This is handled automatically and will raise an exception if needed.

Adding interfaces are handled by property methods on the engine class.

To add a single node interface to an existing engine as Interface 10:

```
>>> engine = Engine('sg_vm')
>>> engine.physical_interface.add_single_node_interface(10, '33.33.33.33', '33.33.33.0/24
↪')
```

Node Interface's are used on IPS, Layer2 Firewall, Virtual and Cluster Engines and represent either a single interface or a cluster member interface used for communication.

To add a node interface to an existing engine:

```
>>> engine = Engine('sg_vm')
>>> engine.physical_interface.add_node_interface(10, '32.32.32.32', '32.32.32.0/24')
```

Inline interfaces can only be added to Layer 2 Firewall or IPS engines. An inline interface consists of a pair of interfaces that do not necessarily have to be contiguous. Each inline interface requires that a 'logical interface' is defined. This is used to identify the interface pair and can be used to simplify policy. See [smc.elements.other.LogicalInterface](#) for more details.

To add an inline interface to an existing engine:

```
>>> from smc.core.engine import Engine
>>> engine = Engine('sg_vm')
...
>>> from smc.elements.helpers import logical_intf_helper
>>> logical_interface = logical_intf_helper('MyLogicalInterface') #get logical interface_
↪reference
>>> engine.physical_interface.add_inline_interface('5-6', logical_interface_ref=logical_
↪intf)
```

**Note:** Use `smc.elements.helpers.logical_intf_helper('name')()` to find the existing logical interface reference by name or create it automatically

Capture Interfaces are used on Layer 2 Firewall or IPS engines as SPAN interfaces.

To add a capture interface to a Layer 2 Firewall or IPS:

```
>>> logical_interface = logical_intf_helper('MyLogicalInterface')
>>> engine = Engine('myengine')
>>> engine.physical_interface.add_capture_interface(10, logical_interface_ref=logical_
↪interface)
```

Cluster Virtual Interfaces are used on clustered engines and require a defined "CVI" (sometimes called a 'VIP'), as well as node dedicated interfaces for the engine initiated communications. Each clustered interface will therefore have 3 total address for a cluster of 2 nodes.

To add a cluster virtual interface on a Firewall Cluster with a zone:

```
>>> engine = Engine('myengine')
>>> engine.physical_interface.add_cluster_virtual_interface(
...     interface_id=1,
...     cluster_virtual='5.5.5.1',
...     cluster_mask='5.5.5.0/24',
...     macaddress='02:03:03:03:03:03',
...     nodes=[{'address': '5.5.5.2', 'network_value': '5.5.5.0/24
↪', 'nodeid': 1},
...             {'address': '5.5.5.3', 'network_value': '5.5.5.0/24
↪', 'nodeid': 2},
...             {'address': '5.5.5.4', 'network_value': '5.5.5.0/24
↪', 'nodeid': 3}],
...     zone_ref=zone_helper('Heartbeat'))
```

**Warning:** Make sure the cluster virtual netmask matches the node level networks

Nodes specified are the individual node dedicated addresses for the cluster members.

VLANs can be applied to layer 3 or inline interfaces. For inline interfaces, these will not have assigned IP addresses, however layer 3 interfaces will require addressing.

To add a VLAN to a generic physical interface for single node or a node interface, independent of engine type:

```
>>> engine = Engine('myengine')
>>> engine.physical_interface.add_vlan_to_node_interface(23, 154)
>>> engine.physical_interface.add_vlan_to_node_interface(23, 155)
>>> engine.physical_interface.add_vlan_to_node_interface(23, 156)
```

This will add 3 VLANs to physical interface 23. If this is a layer 3 routed firewall, you may still need to add addressing to each VLAN.

---

**Note:** In the case of Virtual NGFW Engines, it may be advisable to create the physical interfaces with VLANs on the Master NGFW Engine and allocate the IP addressing scheme to the Virtual NGFW Engine.

---

To add layer 3 interfaces with a VLAN and IP address:

```
>>> engine = Engine('myengine')
>>> engine.physical_interface.add_ipaddress_to_vlan_interface(
...     interface_id=2,
...     address='3.3.3.3',
...     network_value='3.3.3.0/24',
...     vlan_id=3,
...     zone_ref=zone_helper('Internal'))
```

---

**Note:** The physical interface will be created if it doesn't already exist

---

When adding VLANs to a cluster interface, there are multiple options. Adding a VLAN, then adding a CVI interface, adding a VLAN and only NDI interfaces, adding VLAN with CVI and NDI or adding a simple VLAN with no interfaces.

Add a cluster interface with id 2, vlan 2, with no interfaces:

```
engine.physical_interface.add_ipaddress_and_vlan_to_cluster(
    interface_id=2, vlan_id=2)
```

Add a cluster interface with id 2, vlan 2 and a single CVI interface with no macaddress (exempts this interface from load balancing):

```
engine.physical_interface.add_ipaddress_and_vlan_to_cluster(
    interface_id=2, vlan_id=2,
    cluster_virtual='3.3.3.1',
    cluster_mask='3.3.3.0/24',
    macaddress=None)
```

Add a cluster interface with id 2, vlan 2, single CVI interface and macaddress to allow load balancing. Set cluster mode to 'packetdispatch':

```
engine.physical_interface.add_ipaddress_and_vlan_to_cluster(
    interface_id=2, vlan_id=2,
    nodes=None, cluster_virtual='22.22.22.22',
    cluster_mask='22.22.22.0/24',
    macaddress='02:02:02:02:02:02',
    cvi_mode='packetdispatch')
```

Add a cluster interface with id 2, vlan 2, a CVI, NDI interfaces along with an assigned macaddress and zone:

```
engine.physical_interface.add_ipaddress_and_vlan_to_cluster(
    interface_id=2, vlan_id=2,
    nodes=[{'address': '4.4.4.4', 'network_value': '4.4.
↪4.0/24', 'nodeid':1},
          {'address': '4.4.4.5', 'network_value': '4.4.
↪4.0/24', 'nodeid':2}]
    cluster_virtual='4.4.4.1',
    cluster_mask='4.4.4.0/24',
    macaddress='02:02:02:02:02:02',
    cvi_mode='packetdispatch',
    zone_ref=zone_helper('thiszone'))
```

To add VLANs to layer 2 or IPS inline interfaces:

```
>>> logical_interface = logical_intf_helper('default_eth') #find logical intf or create_
↪it
...
>>> engine = Engine('myengine')
>>> engine.physical_interface.add_vlan_to_inline_interface(interface_id='5-6',
...                                                         vlan_id=56,
...                                                         logical_interface_ref=logical_
↪interface)
...
>>> engine.physical_interface.add_vlan_to_inline_interface(interface_id='5-6',
...                                                         vlan_id=57,
...                                                         logical_interface_ref=logical_
↪interface)
...
>>> engine.physical_interface.add_vlan_to_inline_interface(interface_id='5-6',
...                                                         vlan_id=58,
...                                                         logical_interface_ref=logical_
↪interface)
```

---

**Note:** The physical interface will be created if it doesn't already exist

---

To see additional information on interfaces, [smc.core.interfaces](#) reference documentation

### 8.3.1 Sub-Interface and VLAN

Top level interface types hold basic settings about the interface, and sub-interfaces define the actual configuration itself, such as IP Addresses, Netmask, which node the interface is assigned to, etc. To obtain more information about a given interface such as sub-interfaces or vlans, use the interface `vlan_interfaces()` and `sub_interfaces()` resources.

To show all vlan interfaces:

```
>>> for interface in engine.interface.all():
...     if interface.has_vlan:
...         print(interface.vlan_interfaces())
[PhysicalVlanInterface(address=None,vlan_id=14), PhysicalVlanInterface(address=45.45.45.
↪50,vlan_id=13)]
```

Interfaces that have IP addresses assigned are considered ‘sub interfaces’. There may be multiple sub interfaces on a given physical interface if multiple IP’s are assigned.

Display addresses for a specific interface (showing the sub-interfaces):

```
>>> for interface in engine.interface.all():
...     if interface.name == 'Interface 0':
...         print(interface.sub_interfaces())
[SingleNodeInterface(name=172.18.1.55)]
```

It is not required to traverse the physical or sub-interface hierarchy to view properties of an interface.

Show IP addresses and networks for all interfaces:

```
>>> for interface in engine.interface.all():
...     print(interface.name, interface.addresses)
('Tunnel Interface 2001', [('169.254.9.22', '169.254.9.20/30', '2001')])
('Tunnel Interface 2000', [('169.254.11.6', '169.254.11.4/30', '2000')])
('Interface 2', [('192.168.1.252', '192.168.1.0/24', '2'), ('192.168.1.253', '192.168.1.
↪0/24', '2')])
('Interface 1', [('10.0.0.254', '10.0.0.0/24', '1'), ('10.0.0.253', '10.0.0.0/24', '1'), ↪
↪('10.0.0.252', '10.0.0.0/24', '1')])
('Interface 0', [('172.18.1.254', '172.18.1.0/24', '0'), ('172.18.1.252', '172.18.1.0/24
↪', '0'), ('172.18.1.253', '172.18.1.0/24', '0')])
```

See `smc.core.interfaces.Interface` for more info.

### 8.3.2 Modifying Interfaces

To modify an existing interface, you will first need to obtain a reference to the interface. There are some modifications that may have dependencies on other settings. For example, when an interface is configured with an IP address, the SMC will automatically create a route entry mapping that physical interface to the directly connected network. Changing the IP will leave the old network definition from the previously assigned interface and would also need to be removed.

---

**Note:** Save must be called on the interface itself or changes will only be made to a local copy of the element.

---

Example of changing the IP address of an existing single node interface:

```
>>> for interface in engine.interface.all():
...     if interface.name == 'Interface 0':
...         for intf in interface.sub_interfaces():
...             intf.address = '172.18.1.60'
...             interface.save()
...
>>> intf = engine.interface.get(0)
>>> print(intf.addresses)
[('172.18.1.60', '172.18.1.0/24', '0')]
```

Change the zone on the top level Physical Interface:

```
>>> intf = engine.interface.get(0)
>>> intf.zone_ref=zone_helper('My New Zone')
>>> intf.save()
```

Change a VLAN on a single NGFW engine node under Interface 2:

```
>>> intf = engine.interface.get(2)
>>> for vlan in intf.vlan_interfaces():
...     if vlan.vlan_id == '14':
...         vlan.vlan_id = '15'
...         intf.save()
```

### 8.3.3 Deleting Interfaces

Deleting interfaces by referencing the interface from the engine context.

Once you have loaded the engine, you can display all available interfaces by calling using the engine level property interface: `smc.core.engine.Engine.interface()` to view all interfaces for the engine.

The name of the interface is the name the NGFW gives the interface based on interface index. For example, physical interface 1 would be “Interface 1” and so on.

Viewing all interfaces and removing one by id:

```
>>> engine = Engine('testfw')
>>> for interface in engine.interface.all():
...     print(interface)
...
PhysicalInterface(name=Interface 12)
TunnelInterface(name=Tunnel Interface 2000)
PhysicalInterface(name=Interface 10)
TunnelInterface(name=Tunnel Interface 1001)
TunnelInterface(name=Tunnel Interface 1000)
PhysicalInterface(name=Interface 20)
PhysicalInterface(name=Interface 11)
PhysicalInterface(name=Interface 40)
...
>>> intf = engine.interface.get(20)      #Get interface 20
>>> print(intf.name)
Interface 20
...
>>> intf.delete()                       #Delete interface
```

To see additional information on interfaces, [smc.core.interfaces](#) reference documentation

## 8.4 Routing

Adding routes to routed interfaces is done by loading the engine and providing the next hop gateway and destination network as parameters. It is not necessary to specify the interface to place the route, the mapping will be done automatically in the SMC based on the existing IP addresses and networks configured on the engine.

Show routes, and view specific interface details:

```
>>> from smc.core.engine import Engine
>>> engine = Engine('testfw')
>>> for routes in engine.routing.all():
...     print(routes)
...
Routing(name=Interface 1,level=interface)
Routing(name=Tunnel Interface 1000,level=interface)
Routing(name=Interface 11,level=interface)
Routing(name=Tunnel Interface 2000,level=interface)
Routing(name=Interface 10,level=interface)
```

Details of interface 1 routes:

```
>>> for routes in engine.routing.all():
...     if routes.name == 'Interface 1':
...         print(routes.all())
...
[Routing(name=network-1.1.1.0/24,level=network), Routing(name=network-2.2.2.0/24,
↪level=network)]
```

Add a route. It is not required to specify the interface in which to add the route, the gateway will determine the interface as it is required to be directly connected:

```
>>> engine = Engine('master-eng')
>>> engine.add_route(gateway='172.18.1.200', network='192.168.17.0/24')
```

## 8.5 Licensing

NGFW Engine licensing for physical appliances is done by having the Management Server ‘fetch’ the license POS from the appliance and auto-assign the license. If the engine is running on a platform that doesn’t have a POS (Proof-of-Serial) such as a virtual platform, then the fetch will fail. In this case, it is possible to do an auto bind which will look for unassigned dynamic licenses available in the Management Server database.

Example of attempting an auto-fetch and falling back to auto binding a dynamic license:

```
>>> engine = Engine('testfw')
>>> for node in engine.nodes:
...     node.bind_license()
```

## POLICIES

Policies are available for all 3 NGFW Engine roles, Firewall, Layer 2 Firewall and IPS. The only initial requirement to create a policy is to reference a policy template. The policy template is a pre-configured set of best practice rules that provide connectivity and enables basic features such as stateful inspection, etc.

Obtaining available templates can be achieved through the collections interface:

```
>>> from smc.policy.layer3 import FirewallTemplatePolicy
>>> FirewallTemplatePolicy.objects.all()
>>> print(list(FirewallTemplatePolicy.objects.all()))
[FirewallTemplatePolicy(name=Firewall Inspection Template),
 FirewallTemplatePolicy(name=Firewall Template)]
```

Example of creating a basic firewall policy; reference template by name:

```
>>> from smc.policy.layer3 import FirewallPolicy
>>> FirewallPolicy.create('newpolicy', template='Firewall Template')
FirewallPolicy(name=newpolicy)
```

Loading an existing policy is similar to obtaining other elements:

```
>>> policy = FirewallPolicy('newpolicy')
>>> policy.template
FirewallTemplatePolicy(name=Firewall Template)
```

Once a policy instance has been obtained, Access or NAT rules can be added, viewed, or removed.

Example of creating a rule for a firewall policy:

```
>>> policy.fw_ipv4_access_rules.create(name='newrule', sources='any', destinations='any',
    ↳ services='any', action='permit')
'http://1.1.1.1:8082/6.1/elements/fw_policy/265/fw_ipv4_access_rule/2099472'

#View all rules
>>> for rule in policy.fw_ipv4_access_rules.all():
...     print(rule.name, rule.sources, rule.destinations, rule.services)
...
('newrule', <smc.policy.rule_elements.Source object at 0x1050d3b50>, <smc.policy.rule_
    ↳ elements.Destination object at 0x1050d3dd0>, <smc.policy.rule_elements.Service object
    ↳ at 0x1050d3f50>)
```

NAT can be applied as dynamic source NAT, static source NAT, or static destination NAT.

Example of creating a dynamic source NAT rule:

```
>>> from smc.policy.layer3 import FirewallPolicy
>>> from smc.elements.network import Host
>>> policy = FirewallPolicy('newpolicy')
>>> policy.fw_ipv4_nat_rules.create(name='mynat',
...                               sources=[Host('kali')],
...                               destinations='any',
...                               services='any',
...                               dynamic_src_nat='1.1.1.1',
...                               dynamic_src_nat_ports=(1024,65535))
'http://1.1.1.1:8082/6.1/elements/fw_policy/265/fw_ipv4_nat_rule/2099475'
```

Example of creating a destination NAT rule where the destination is to Host('3.3.3.3') and will be translated to '1.1.1.1':

```
>>> policy.fw_ipv4_nat_rules.create(name='mynat',
...                               sources='any',
...                               destinations=[Host('3.3.3.3')],
...                               services='any',
...                               static_dst_nat='1.1.1.1')
'http://1.1.1.1:8082/6.1/elements/fw_policy/265/fw_ipv4_nat_rule/2099476'
```

Create an any/any no NAT rule (no value for NAT field):

```
>>> policy.fw_ipv4_nat_rules.create(name='nonat', sources='any', destinations='any',
... ↪ services='any')
'http://1.1.1.1:8082/6.1/elements/fw_policy/265/fw_ipv4_nat_rule/2099477'
```

For additional NAT related options, see: *smc.policy.rule\_nat.IPv4NATRule*

## VPN

It is possible to create all gateway elements and configurations related to Policy Based VPN. Gateway's in the VPN configuration can be either managed engines or remote gateways (ExternalGateway).

There are several components or terminology required to set up a VPN.

- External Gateway: Third-party VPN gateway or an NGFW Engine managed by a different Management Server
- External Endpoint: VPN Endpoint/s defined in external gateway (IP addresses, profiles)
- Sites: sites define the protected network/s for both sides of the VPN
- Internal Gateway: NGFW Engine managed by the Management Server to which you are currently connected.

When creating a VPN to a non-managed device, an external gateway is required. This is a container object used to encapsulate the remote endpoints where the VPN will terminate:

```
>>> gateway = ExternalGateway.create('remoteside')
```

An external endpoint specifies the IP address settings and other VPN specific settings for the external gateway.

Create the external endpoint from the gateway resource:

```
>>> gateway.external_endpoint.create(name='remoteendpoint', address='2.2.2.2')
'http://1.1.1.1:8082/6.1/elements/external_gateway/22961/external_endpoint/26740'
```

Lastly, 'sites' need to be configured that identify the network/s for the external gateway side of the VPN. You can use pre-existing network elements, or create new ones as in the example below.

```
>>> network = Network('internal-network')
>>> print(network.href)
http://1.1.1.1:8082/6.1/elements/network/17911
...
>>> gateway.vpn_site.create('remote-site', [network.href])
'http://1.1.1.1:8082/6.1/elements/external_gateway/22961/vpn_site/22994'
```

Retrieve the engine internal gateway resource for the managed engine by obtaining the engine context.

```
>>> engine = Engine('testfw')
>>> print(engine.internal_gateway.href) #Internal gateway resource
http://1.1.1.1:8082/6.1/elements/single_fw/39550/internal_gateway/11476
```

Create the VPN Policy and apply the internal gateway as the 'Central Gateway' and the ExternalGateway as the 'Satellite Gateway':

```
>>> vpn = PolicyVPN.create(name='myVPN', nat=True)
>>> print(vpn.name, vpn.vpn_profile)
(u'myVPN', u'http://172.18.1.150:8082/6.1/elements/vpn_profile/2')
...
>>> vpn.open()
>>> vpn.add_central_gateway(engine.internal_gateway.href)
>>> vpn.add_satellite_gateway(external_gateway.href)
>>> vpn.save()
>>> vpn.close()
```

---

**Note:** You must call `smc.vpn.policy.PolicyVPN.open()` before modifications can be made. You also must call `smc.vpn.policy.PolicyVPN.save()` and `smc.vpn.policy.PolicyVPN.close()`

---

See API Reference documentation for more details.

## ADMINISTRATION

Administration provides an interface to system level administration tasks such as creating administrators, updating the SMC with dynamic updates, updating NGFW Engines with engine upgrades, running tasks, etc.

### 11.1 Administrators

Creating administrators and modifying settings can be done using the `smc.elements.user.AdminUser` class.

For example, to create a user called ‘administrator’ and modify after creation, do:

Create admin:

```
AdminUser.create('administrator')
```

To modify after creation by setting a password and making a superuser:

```
admin = AdminUser('administrator') # Load an admin user called administrator
admin.change_password('mynewpassword')
admin.update(superuser=True) # ad-hoc update of attribute
admin.enable_disable() #enable or disable account
```

### 11.2 Tasks

Tasks may be generated by methods within certain classes, for example, many classes support an `export()` method. This is an asynchronous task that generates a ‘follower’ link to the task.

It is possible to monitor those asynchronous operations separately from the direct method call by getting the follower href and using `smc.actions.tasks.TaskMonitor` or `smc.actions.tasks.TaskDownload` classes.

For example, fire off a policy update on an engine and get the asynchronous follower href:

```
engine = Engine('myfw')
task_follower = engine.refresh(wait_for_finish=True) #This isn't required as engine will
↳ still refresh
while not task_follower.done():
    task_follower.wait(3)
print("Did task succeed: %s" % task_follower.success)
print("Last message from task: %s" % task_follower.last_message)
```

## 11.3 System

System level tasks include operations such as checking for and downloading a new dynamic update, engine upgrades, last activated package, SMC version, SMC time, emptying the trash bin, viewing all license details, importing, exporting elements and submitting global blacklist entries.

To view any available update packages:

```
from smc.administration.system import System
system = System()
available_packages = system.update_package()
print(list(available_packages))
```

To fully download and activate a dynamic update:

```
system = System()
available_packages = system.update_package()

my_dynup = available_packages.get_contains('1097')

if my_dynup.state.lower() == 'available':
    download_task = my_dynup.download(wait_for_finish=True)
    while not download_task.done():
        download_task.wait(3)
        print(download_task.last_message())
    if download_task.success:
        print("Success!")

# We are now downloaded, so activate
activation = my_dynup.activate(wait_for_finish=True)
while not activation.done():
    activation.wait(3)
    print(activation.last_message())

if activation.success:
    print("We are now activated")
else:
    print("Something bad went wrong: %s" % activation.last_message())
```

Empty the trash bin:

```
system = System()
system.empty_trash_bin()
```

## LOGGING

The smc-python API uses python logging for INFO, ERROR and DEBUG logging levels. If needed, add the following to your classes:

```
import logging
logging.getLogger()
logging.basicConfig(level=logging.ERROR, format='%(asctime)s %(levelname)s: %(message)s')
```

---

**Note:** This is a recommended setting initially as it enables detailed logging of each call as it is processed through the API. It also includes the backend web based calls initiated by the requests module.

---

If you simply require stream logging to console for scripts, from your script import the smc module `set_stream_logger`, debug level, and optional format string conforming to the logging module:

```
from smc import set_stream_logger
set_stream_logger(level=logging.DEBUG, format_string=None)
```



## EXTENSIONS

smc-python provides additional extensions to extend the base library. Extensions are installed as separate packages and will have the dependency on the base smc-python library.

Available extensions:

- smc-python-monitoring

### 13.1 smc-python-monitoring

smc-python-monitoring API provides a monitoring interface to the SMC to perform queries for dynamic engine components such as blacklists, connections, routes, vpn's, users and logs.

Capabilities in the API implement the functionality found in the Log Server and engine level monitoring.

#### 13.1.1 Query

A Query is the top level object used to construct parameters to make queries to the SMC.

Query is the parent class for all monitors in package *smc\_monitoring.monitors*

Each monitor type will have it's own predefined set of log fields that are considered 'default' for the query type. These will correlate closely to the default fields you will see in the Management Client component of the SMC when viewing the same information (Connections, VPN SAs, Blacklist, etc).

Each query also has a specific formatter which defines how the data is returned from the query. Formatters are defined in *smc\_monitoring.models.formats*.

Each formatter type allows customization of the field\_format and allows a value of 'pretty', 'name' or 'id'. By default 'pretty' is used as the format which aligns with the column names in the Management Client monitoring views.

**class** `smc_monitoring.models.query.Query`(*definition=None, target=None, format=None, \*\*socket*)

Query is the top level structure for controlling requests over the SMC websocket protocol. Any keyword arguments are passed through from inheriting classes are passed through as socket options for `smc_monitoring.websocket.SMCWebSocketProtocol`.

#### Variables

- **request** (*dict*) – built request, eventually sent to socket
- **format** (*TextFormat*) – format settings for query

**add\_and\_filter(\*values)**

Add a filter using “AND” logic. This filter is useful when requiring multiple matches to evaluate to true. For example, searching for a specific IP address in the src field and another in the dst field.

**See also:**

[\*smc\\_monitoring.models.filters.AndFilter\*](#) for examples.

**Parameters**

**values** – optional constructor args for [\*smc\\_monitoring.models.filters.AndFilter\*](#). Typically this is a list of InFilter expressions.

**Type**

[\*list\(QueryFilter\)\*](#)

**Return type**

[\*AndFilter\*](#)

**add\_defined\_filter(\*value)**

Add a DefinedFilter expression to the query. This filter will be considered true if the `smc.monitoring.values.Value` instance has a value.

**See also:**

[\*smc\\_monitoring.models.filters.DefinedFilter\*](#) for examples.

**Parameters**

**value (Value)** – single value for the filter. Value is of type [\*smc\\_monitoring.models.values.Value\*](#).

**Type**

[\*list\(QueryFilter\)\*](#)

**Return type**

[\*DefinedFilter\*](#)

**add\_in\_filter(\*values)**

Add a filter using “IN” logic. This is typically the primary filter that will be used to find a match and generally combines other filters to get more granular. An example of usage would be searching for an IP address (or addresses) in a specific log field. Or looking for an IP address in multiple log fields.

**See also:**

[\*smc\\_monitoring.models.filters.InFilter\*](#) for examples.

**Parameters**

**values** – optional constructor args for [\*smc\\_monitoring.models.filters.InFilter\*](#)

**Return type**

[\*InFilter\*](#)

**add\_not\_filter(\*value)**

Add a filter using “NOT” logic. Typically this filter is used in conjunction with and AND or OR filters, but can be used by itself as well. This might be more useful as a standalone filter when displaying logs in real time and filtering out unwanted entry types.

**See also:**

[\*smc\\_monitoring.models.filters.NotFilter\*](#) for examples.

**Parameters**

**values** – optional constructor args for `smc_monitoring.models.filters.NotFilter`. Typically this is a list of InFilter expressions.

**Type**

`list(QueryFilter)`

**Return type**

`OrFilter`

**add\_or\_filter(\*values)**

Add a filter using “OR” logic. This filter is useful when matching on one or more criteria. For example, searching for IP 1.1.1.1 and service TCP/443, or IP 1.1.1.10 and TCP/80. Either pair would produce a positive match.

**See also:**

`smc_monitoring.models.filters.OrFilter` for examples.

**Parameters**

**values** – optional constructor args for `smc_monitoring.models.filters.OrFilter`. Typically this is a list of InFilter expressions.

**Type**

`list(QueryFilter)`

**Return type**

`OrFilter`

**add\_translated\_filter()**

Add a translated filter to the query. A translated filter syntax uses the SMC expression syntax to build the filter. The simplest way to see the syntax is to create a filter in the Logs view of the Management Client and right click->Show Expression.

Example to fetch a specific Situation from the Active Alerts:

```
t_filter.update_filter('$Situation==516')
query = ActiveAlertQuery('Shared Domain', timezone='Berlin/Europe')
query.update_filter(t_filter)

for record in query.fetch_batch():
    print(record)

.. seealso:: :class:`smc_monitoring.models.filters.TranslatedFilter` for
examples.
```

**Parameters**

**values** – optional constructor args for `smc_monitoring.models.filters.TranslatedFilter`

**Type**

`list(QueryFilter)`

**Return type**

`TranslatedFilter`

**execute()**

Execute the query with optional timeout. The response to the execute query is the raw payload received from the websocket and will contain multiple dict keys and values. It is more common to call `query.fetch_XXX` which will filter the return result based on the method. Each result set will have a max batch size of 200 records. This method will also continuously return results until terminated. To make a single bounded fetch, call `fetch_batch()` or `fetch_raw()`.

**Parameters**

**sock\_timeout** (*int*) – event loop interval

**Returns**

raw dict returned from query

**Return type**

`dict(list)`

**fetch\_as\_element()**

Each inheriting class will override this method if supported.

**fetch\_batch**(*formatter=<class 'smc\_monitoring.models.formatters.TableFormat'>, \*\*kw*)

Fetch and return in the specified format. Output format is a formatter class in `smc_monitoring.models.formatters`. This fetch type will be a single shot fetch unless providing `max_recv` keyword with a value greater than the default of 1. Keyword arguments available are kw in `fetch_raw()`.

**Parameters**

- **query\_timeout** (*int*) – length of time to wait on receiving web socket results (total query time).
- **inactivity\_timeout** (*int*) – length of time before exiting if no new entry.
- **max\_recv** (*int*) – for queries that are not ‘live’, set this to supply a max number of receive iterations.
- **formatter** – Formatter type for data representation. Any type in `smc_monitoring.models.formatters`.

**Returns**

generator returning data in specified format

---

**Note:** You can provide your own formatter class, see `smc_monitoring.models.formatters` for more info.

---

**fetch\_live**(*formatter=<class 'smc\_monitoring.models.formatters.TableFormat'>*)

Fetch a live stream query. This is the equivalent of selecting the “Play” option for monitoring fields within the Management Client. Data will be streamed back in real time.

**Parameters**

**formatter** – Formatter type for data representation. Any type in `smc_monitoring.models.formatters`.

**Returns**

generator yielding results in specified format

**fetch\_raw**(\*\*kw)

Fetch the records for this query. This fetch type will return the results in raw dict format. It is possible to limit the number of receives on the socket that return results before exiting by providing `max_recv`.

This fetch should be used if you want to return only the result records returned from the query in raw dict format. Any other dict key/values from the raw query are ignored.

**Parameters**

- **max\_recv** (*int*) – max number of socket receive calls before returning from this query. If you want to wait longer for results before returning, increase max\_iterations (default: 0)
- **query\_timeout** (*int*) – length of time to wait on receiving web socket results (total query time).
- **inactivity\_timeout** (*int*) – length of time before exiting if no new entry.

**Returns**

list of query results

**Return type**

`list(dict)`

**static resolve\_field\_ids**(*ids*, *\*\*kw*)

Retrieve the log field details based on the LogField constant IDs. This provides a helper to view the fields representation when using different field\_formats. Each query class has a default set of field IDs that can easily be looked up to examine their fields and different label options. For example:

```
Query.resolve_field_ids(ConnectionQuery.field_ids)
```

**Parameters**

**ids** (*list*) – list of log field IDs. Use LogField constants to simplify search.

**Returns**

raw dict representation of log fields

**Return type**

`list(dict)`

**update\_filter**(*filt*)

Update the query with a new filter.

**Parameters**

**filt** (`smc_monitoring.models.filters.QueryFilter`) – change query to use new filter

**update\_format**(*format*)

Update the format for this query.

**Parameters**

**format** (`smc_monitoring.models.formats`) – new format to use for this query

## 13.1.2 Models

The models package consists of the building blocks that make up a query.

Each module represents different class models that simplify adding things like filters, specifying values and formats.

### 13.1.2.1 Filters

Filters are used by queries to refine how results are returned.

QueryFilter is the top level ‘interface’ for all filter types. The `filter` attribute of a QueryFilter provides access to the compiled query string used to build the filter. Each QueryFilter also has an `update_filter` method that can be used to swap new filters in and out of an existing query.

Filters can be added to queries using the `add_XXX` methods of the query, or by building the filters and adding to the query using `query.update_filter()`. Filters can be swapped in and out of a query.

Examples:

Build a query to return all records of alert severity high or critical:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.ALERTSEVERITY), [ConstantValue(Alerts.HIGH, Alerts.CRITICAL)])
```

If you prefer building your filters individually, it is not required to call the `add_XX_filter` methods of the query. You can also insert filters by building the filter and calling the `update_filter` method on the query:

```
query = LogQuery(fetch_size=50)
query.update_filter(
    InFilter(FieldValue(LogField.SERVICE), [ServiceValue('UDP/53', 'TCP/80')]))
```

You can also replace existing query filters with new filters to re-use the base level query parameters such as `fetch_size`, `format style`, `time/date ranges`, etc.

Replace the existing query filter with a different filter:

```
new_filter = InFilter(FieldValue(LogField.SERVICE), [ServiceValue('UDP/53', 'TCP/80')]))
query.update_filter(new_filter)
```

---

**Note:** it is also possible to update a filter by calling `query.add_XX_filter` methods multiple times. Each time will replace an existing filter if it exists.

---

For example, calling `add_XX_filter` methods multiple times to refine filter results:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(    # First filter query - look for alert severity high and critical
    FieldValue(LogField.ALERTSEVERITY), [ConstantValue(Alerts.HIGH, Alerts.CRITICAL)])

query.add_and_filter([    # Change filter to AND filter for further granularity
    InFilter(FieldValue(LogField.ALERTSEVERITY), [ConstantValue(Alerts.HIGH, Alerts.
↪CRITICAL)]),
    InFilter(FieldValue(LogField.SRC), [IPValue('192.168.4.84')])])
```

```
class smc_monitoring.models.filters.AndFilter(*filters)
```

Bases: QueryFilter

An AND filter combines other filter types and requires that each filter matches. An AND filter is a collection of QueryFilter’s, typically IN or NOT filters that are AND’d together.

Example of fetching 50 records for sources matching ‘192.168.4.84’ and a service of ‘TCP/80’:

```
query = LogQuery(fetch_size=50)
query.add_and_filter([
    InFilter(FieldValue(LogField.SRC), [IPValue('192.168.4.84')]),
    InFilter(FieldValue(LogField.SERVICE), [ServiceValue('TCP/80')])])
```

**Parameters**

**filters** (*list* or *tuple*) – Any filter type in `smc.monitoring.filters`.

**class** `smc_monitoring.models.filters.CILikeFilter`

Bases: `QueryFilter`

A `CILikeFilter` is a case insensitive LIKE string match filter.

**class** `smc_monitoring.models.filters.CSLikeFilter`

Bases: `QueryFilter`

A `CSLikeFilter` is a case sensitive LIKE string match filter.

**class** `smc_monitoring.models.filters.DefinedFilter`(*value=None*)

Bases: `QueryFilter`

A `DefinedFilter` applied to a query will only match if the value specified has a value in the audit record/s.

Show only records that have a defined Action (read as ‘match if action has a value’):

```
query = LogQuery(fetch_size=50)
query.add_defined_filter(FieldValue(LogField.ACTION))
```

`DefinedFilter`’s can be used in AND, OR or NOT filter queries as well. Fetch the most recent 50 records for source 192.168.4.84 that have an application defined:

```
query = LogQuery(fetch_size=50)
query.add_and_filter([
    DefinedFilter(FieldValue(LogField.IPSAPPID)),
    InFilter(FieldValue(LogField.SRC), [IPValue('192.168.4.84')])])
```

**Parameters**

**values** (*Value*) – single value type to require on filter

**class** `smc_monitoring.models.filters.InFilter`(*left, right*)

Bases: `QueryFilter`

`InFilter`’s are made up of two parts, a left and a right. An `InFilter` is considered a match if evaluation of the left part is equivalent to one of the elements of the right part. The left part of an `InFilter` is made up of a target of type `smc.monitoring.values.Value`. The right part is made up of a list of the same type.

Search the Source field for IP addresses 192.168.4.84 or 10.0.0.252:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.SRC), [IPValue('192.168.4.84', '10.0.0.252')])
```

Reverse the logic and search for IP address 192.168.4.84 in source and dest log fields:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    IPValue('192.168.4.84'), [FieldValue(LogField.SRC, LogField.DST)])
```

InFilter's are one of the most common filters and are often added to AND, OR or NOT filters for more specific matching.

#### Parameters

- **left** (Values: any value type in `smc_monitoring.models.values`) – single value for leftmost portion of filter
- **right** (list(Values): any value type in `smc_monitoring.models.values`) – list of values for rightmost portion of filter

```
class smc_monitoring.models.filters.NotFilter(*filters)
```

Bases: QueryFilter

A NOT filter provides the ability to suppress auditing based on a specific filter. A NOT filter is typically added to an AND filter to remove unwanted entries from the response.

Use only a NOT filter to a query and to ignore DNS traffic:

```
query = LogQuery(fetch_size=50)
query.add_not_filter(
    [InFilter(FieldValue(LogField.SERVICE), [ServiceValue('UDP/53')])])
```

The above example by itself is not overly useful, however you can use NOT filters with AND filters to achieve a logic like “Find source IP 192.168.4.68 and not service UDP/53 or TCP/80”:

```
query = LogQuery(fetch_size=50)
not_dns = NotFilter(
    [InFilter(FieldValue(LogField.SERVICE), [ServiceValue('UDP/53', 'TCP/80')])])
by_ip = InFilter(
    FieldValue(LogField.SRC), [IPValue('172.18.1.20')])
query.add_and_filter([not_dns, by_ip])
```

#### Parameters

**filters** (*list* or *tuple*) – Any filter type in `smc.monitoring.filters`.

```
class smc_monitoring.models.filters.OrFilter(*filters)
```

Bases: QueryFilter

An OR filter matches if any of the combined filters match. An OR filter is a collection of QueryFilter's, typically IN or NOT filters that are OR'd together.

Example of fetching 50 records for sources matching '192.168.4.84' or a service of 'TCP/80':

```
query = LogQuery(fetch_size=50)
query.add_or_filter([
    InFilter(FieldValue(LogField.SRC), [IPValue('192.168.4.84')]),
    InFilter(FieldValue(LogField.SERVICE), [ServiceValue('TCP/80')])])
```

#### Parameters

**filters** (*list* or *tuple*) – Any filter type in `smc.monitoring.filters`.

**class** smc\_monitoring.models.filters.TranslatedFilter

Bases: QueryFilter

Translated filters use the SMC internal name alias and builds expressions to make more complex queries.

Example of using built in filter methods:

```

query = LogQuery(fetch_size=50)
query.format.timezone('CST')
query.format.field_format('name')

translated_filter = query.add_translated_filter()
translated_filter.within_ipv4_network('$Dst', ['192.168.4.0/24'])
translated_filter.within_ipv4_range('$Src', ['1.1.1.1-192.168.1.254'])
translated_filter.exact_ipv4_match('$Src', ['172.18.1.152', '192.168.4.84'])

```

**exact\_ipv4\_match**(*field*, *values*)

An exact IPv4 address match on relevant address fields.

**Parameters**

- **field** (*str*) – name of field to filter on. Taken from ‘Show Filter Expression’ within the Management Client.
- **values** (*list*) – value/s to add. If more than a single value is provided, the query is modified to use UNION vs. ==
- **complex** (*bool*) – A complex filter is one which requires AND’ing or OR’ing values. Set to return the filter before committing.

**within\_ipv4\_network**(*field*, *values*)

This filter adds specified networks to a filter to check for inclusion.

**Parameters**

- **field** (*str*) – name of field to filter on. Taken from ‘Show Filter Expression’ within Management Client.
- **values** (*list*) – network definitions, in cidr format, i.e: 1.1.1.0/24.

**within\_ipv4\_range**(*field*, *values*)

Add an IP range network filter for relevant address fields. Range (between) filters allow only one range be provided.

**Parameters**

- **field** (*str*) – name of field to filter on. Taken from ‘Show Filter Expression’ within Management Client.
- **values** (*list*) – IP range values. Values would be a list of IP’s separated by a ‘-’, i.e. ['1.1.1.1-1.1.1.254']

### 13.1.2.2 Values

Values are used to provide searchable input for filters. Each value format is specific to the data type added to the filter. For example, an `IPValue` specifies IP's or network values that can be added to a filter from `smc_monitoring.models.filters`.

Each constructor can be initialized in the following ways:

Single value:

```
IPValue('1.1.1.1')
```

Multiple values:

```
IPValue('1.1.1.1', '2.2.2.2')
```

As a list of values:

```
i = ['1.1.1.1', '3.3.3.3']
IPValue(*i)
```

The value attribute of each `Value` stores the query string as a list that is absorbed by the filter.

**class** `smc_monitoring.models.values.ConstantValue(*constants)`

Bases: `Value`

Constant values can be used for log field values. For example, specifying a filter by Action can be simplified by specifying the constant for the action value. Constant values are not used for log field names (use `FieldValue` instead).

Searching for all actions of discard and block:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.ACTION), [ConstantValue(Actions.DISCARD, Actions.BLOCK)])
```

#### Parameters

**constants** (*list or str*) – constant values

**class** `smc_monitoring.models.values.ElementValue(*elements)`

Bases: `Value`

Element Values are used when creating a filter for an element already defined in the Management Server database. The element can be referenced by its type.

Search for a host element 'kali' in the 'source' log field:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.SRC), [ElementValue(Host('kali'))])
```

#### Parameters

**elements** (*list or str*) – element definitions

---

**Note:** Using elements expands the search to potentially include a broader range of data. For example, a host can have multiple IP addresses, both ipv4 and ipv6.

---

**class** smc\_monitoring.models.values.**FieldValue**(\*fields)

Bases: [Value](#)

FieldValue specifies a log field filter by either constant ID or name. The field name field is the internal name representation for the Management Client. To find a given field name, in the Logs view of the Management Client, drag a field into the filter window, right click and select “Show Filter Expression”.

Using field value as filter for InFilter type:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.SRC), [IPValue('192.168.4.84')])
```

#### Parameters

**fields** (*list* or *str*) – fields definitions by name or int ID

---

**Note:** If using constant values, consult `smc_monitoring.constants.LogField` for valid attributes.

---

**class** smc\_monitoring.models.values.**IPValue**(\*addresses)

Bases: [Value](#)

IP Values specify IP addresses used for searching.

Search for IP address in source and dest fields:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    IPValue('192.168.4.84'), [FieldValue(LogField.SRC, LogField.DST)])
```

#### Parameters

**addresses** (*list* or *str*) – address definitions

**class** smc\_monitoring.models.values.**NumberValue**(\*values)

Bases: [Value](#)

Number value match.

Search for port in source fields:

```
query = LogQuery(fetch_size=10) query.add_in_filter(FieldValue(LogField.SPORT), [NumberValue(7000, 7001)])
```

#### Parameters

**value** (*list* or *int*) – number definitions

**class** smc\_monitoring.models.values.**ServiceValue**(\*services)

Bases: [Value](#)

Service Values allow searches on the service field. When specifying the service value, specify as <protocol/port>. For example, ‘TCP/80’, ‘UDP/53’. For ICMP, specify as ICMP/Type/Code (Code is optional).

Search for any services with TCP port 80 and UDP port 53:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.SERVICE), [ServiceValue('TCP/80', 'UDP/53')])
```

**Parameters**

**services** (*list* or *str*) – service definitions

**class** smc\_monitoring.models.values.**StringValue**(\*values)

Bases: [Value](#)

String value match. Note that string matching can only be done on log fields that are of type string (no type conversions are done on non-string types). String matches are also exact.

Find all audits accessing URL play.googleapis.com:

```
query = LogQuery(fetch_size=50)
query.add_in_filter(
    FieldValue(LogField.HTTPREQUESTHOST), [StringValue('play.googleapis.com')])
```

**Parameters**

**value** (*list*) – string to match

**class** smc\_monitoring.models.values.**TranslatedValue**(value)

Bases: [Value](#)

Internal SMC filter format value match. To use with “translated” filter

Search for port in destination fields:

```
query = LogQuery(fetch_size=10)
query_filter.update_filter(TranslatedValue("$Dport == 22 OR $Dport == 25").value)
query.update_filter(query_filter)
```

**Parameters**

**value** (*str*) – specifies a string in the internal SMC filter format.

**class** smc\_monitoring.models.values.**Value**(values)

Bases: [object](#)

Value is the topmost parent for all value types.

**Variables**

**value** – stores value formatted into dict

### 13.1.2.3 Formats

Field formats represent a way to control the format of the returned data. By modifying a field format, you can control field level settings such as wther to resolve IP’s via DNS, how to display field names and values and which fields to return in the query.

Each log format will return a different view type The most common and default for all queries is the [TextFormat](#) using a ‘pretty’ field format which is what you will see from the column data and values if using the Logs view of the Management Client.

Return only a specific set of fields by id’s:

```
query = LogQuery(fetch_size=5)
query.format.field_ids([
    LogField.TIMESTAMP, LogField.NODEID, LogField.SRC,
    LogField.DST, LogField.PROTOCOL, LogField.ACTION])
```

Return only a specific set of fields by name:

```
query = LogQuery(fetch_size=5)
query.format.field_names(['Src', 'Dst'])
```

**Note:** If both `field_ids` and `field_names` are provided, they will be merged.

**class** `smc_monitoring.models.formats.CombinedFormat(**kw)`

Bases: `object`

CombinedFormat provides a way to specify different field resolvers based on field name or ID. Keyword arguments provided will define a unique key that represents the format object and value is the format object itself.

For example, using a combined filter to resolve the `TIMESTAMP` field in text format, but source and destination fields in detailed format:

```
text = TextFormat()
text.field_ids([LogField.TIMESTAMP])

detailed = DetailedFormat()
detailed.field_ids([LogField.SRC, LogField.DST])

combined = CombinedFormat(tformat=text, dformat=detailed)

query = LogQuery(fetch_size=1, format=combined)
```

After executing the query, the raw record results will be formatted as a list of dict's, which each record having a dict key equal to keyword argument input provided:

```
[{'dformat': {'Src Addr': '10.0.0.1', 'Dst Addr': '224.0.0.1'},
  'tformat': {'Creation Time': '2017-08-05 14:12:44'}},
 {'dformat': {'Src Addr': '10.0.0.1', 'Dst Addr': '224.0.0.1'},
  'tformat': {'Creation Time': '2017-08-05 14:12:44'}}]
...
```

The results can then be parsed and used to provide custom views as necessary.

#### Parameters

**kw** – key word arguments should use an identifier key that will be present in the results, and a value which is a format object type in `smc_monitoring.formats`.

**class** `smc_monitoring.models.formats.DetailedFormat(field_format='pretty', **kw)`

Bases: `TextFormat`

Detailed format does not do a Log value conversion as the `TextFormat` would, however does provide a field map in the first payload with characteristics of the fields in the return data. This might be a useful format to obtain conversion ID's for specific fields or debugging.

**class** `smc_monitoring.models.formats.FormatFieldMixin`

Bases: `object`

Format field methods for modifying behavior of a query.

**field\_format**(*name*)

Specify how the field name are printed in the response.

**Parameters**

- **id** (*str*) – as integer IDs from constants found in `smc_monitoring.models.constants.LogField`
- **name** (*str*) – as internal SMC names
- **pretty** (*str*) – pretty printed as you would see in the Management Client

**field\_ids**(*ids*)

Add filter to show only fields with given field ID's. Field ID's can be mapped to the LogField constants in `smc_monitoring.models.constants.LogField`

---

**Note:** Set the return display mode for the Log field name by using `field_format()`. The display value name will match the name of the LogField constant.

---

**field\_names**(*names*)

Show only fields with given name. The name is the name for the log field in the Management Client. The simplest way to obtain the name for a log field is from the Logs view of the Management Client. Use the Query pane in the Logs view of the Management Client to drag a column filter and select “Show Filter Expression”

**..note::** The log field name is case sensitive and is typically using camelcase notation.

**class** `smc_monitoring.models.formats.RawFormat`(*field\_format='pretty'*)

Bases: `FormatFieldMixin`

Raw format is an abbreviated version of the detailed format. Fewer fields are provided and resolution of field values is not done.

**class** `smc_monitoring.models.formats.TextFormat`(*field\_format='pretty', \*\*kw*)

Bases: `FormatFieldMixin`

Text format with ‘pretty’ field formatting uses the same display to what you would see from the native SMC Log Viewer.

Keyword arguments can optionally be provided to set ‘resolving’ fields during instance creation, or they can be set on the instance afterwards by calling `set_resolving()`.

**set\_resolving**(*\*\*kw*)

Certain log fields can be individually resolved. Use this method to set these fields. Valid keyword arguments:

**Parameters**

- **timezone** (*str*) – string value to set timezone for audits
- **time\_show\_zone** (*bool*) – show the time zone in the audit.
- **time\_show\_millis** (*bool*) – show timezone in milliseconds
- **keys** (*bool*) – resolve log field keys

- **ip\_elements** (*bool*) – resolve IP’s to SMC elements
- **ip\_dns** (*bool*) – resolve IP addresses using DNS
- **ip\_locations** (*bool*) – resolve locations

**timezone**(*tz*)

Set timezone on the audit records. Timezone can be in formats: ‘US/Eastern’, ‘PST’, ‘Europe/Helsinki’

See SMC Log Viewer settings for more examples.

#### Parameters

**tz** (*str*) – timezone, i.e. CST

### 13.1.2.4 Constants

Constants used within *smc\_monitoring.models.values.Value* values to simplify referencing log viewer data.

**class** `smc_monitoring.models.constants.Actions`

Bases: `object`

Rule Actions

**ALLOW** = 1

Allowed

**BLOCK** = 13

Block

**DISCARD** = 0

Discard

**DISCARD\_PASSIVE** = 4

Silent discard

**PERMIT** = 11

Permit the connection

**REFUSE** = 2

Reset

**TERMINATE** = 9

Terminate

**TERMINATE\_FAILED** = 10

Failed terminating connection

**TERMINATE\_PASSIVE** = 8

Silent terminate

**TERMINATE\_RESET** = 12

Reset the connection

**class** `smc_monitoring.models.constants.Alerts`

Bases: `object`

Alert actions

**CRITICAL** = 10

Critical alert

**HIGH = 5**  
High alert

**INFO = 1**  
Info alert

**LOW = 3**  
Low alert

**class** smc\_monitoring.models.constants.DataType

Bases: `object`

Query by type of logs. This identifies which log types you are interested in filtering by, i.e. Audit, FW Logs, Third\_Party, etc. Equivalent to the Query dropdown in the Logs view of the Management Client

**class** smc\_monitoring.models.constants.LogField

Bases: `object`

Log field constants can be referenced when creating filters such as Field Values. i.e. FieldValue(LogField.SRC). Each constant name is identical to the value when using the field format type of 'name' (with exception that the constant names are in upper case).

**ACCELAPSED = 104**  
Elapsed time of connection in seconds

**ACCRXBYTES = 106**  
Number of bytes received during connection

**ACCRXPACKETS = 139**  
Number of packets received during connection

**ACCTXBYTES = 105**  
Number of bytes sent during connection

**ACCTXPACKETS = 138**  
Number of packets sent during connection

**ACK = 29**  
Acknowledged Alert

**ACTION = 14**  
Connection action

**ALERT = 25**  
Type of alert

**ALERTCOUNT = 603**  
Alert count

**ALERTERTRACE = 600**  
Alerter trace (events) information (datatype:4)

**ALERTSEVERITY = 602**  
Severity of situation

**ALERTSTATUS = 604**  
Alert status

**ALLOWEDDATATAG = 482**

Allowed data type tag

**APPLICATION = 800**

Application

**APPLICATIONCOMBINATIONFLAGS = 54**

Anomaly information of certain combination of network application and client application.

**APPLICATIONDETAIL = 801**

Application Detail

**APPLICATIONUSAGE = 52**

The type of the application that caused sending this event.

**ASPAMEMAILMESSAGEID = 155**

Email message-ID

**ASPAMEMAILSCORE = 153**

Email score value

**ASPAMEMAILSUBJECT = 152**

Email subject

**ASPAMRECEIVEREMAIL = 151**

Receiver email address

**ASPAMSENDEREMAIL = 150**

Sender email address

**ASPAMSENDERMTA = 154**

Sender Message Transfer Agent IP address

**AUTHENTICATIONCOUNTER = 850**

Authentication counters

**AUTHMETHOD = 133**

Authentication Method element

**AUTHNAME = 108**

User name of authorized user

**AUTHRULEID = 107**

The rule number of the rule that led to the log creation

**BALANCINGPROBING = 397**

BALANCING\_PROBING

**BALANCINGSELECTION = 392**

BALANCING\_SELECTION

**BLACKLISTENTRYDESTINATIONIP = 120**

Blacklist entry destination IP address

**BLACKLISTENTRYDESTINATIONIPMASK = 121**

Blacklist entry destination IP address mask

**BLACKLISTENTRYDESTINATIONPORT = 125**

Blacklist entry destination port

**BLACKLISTENTRYDESTINATIONPORTRANGE = 126**

Blacklist entry destination port range end

**BLACKLISTENTRYDURATION = 127**

Blacklist entry duration

**BLACKLISTENTRYID = 117**

None

**BLACKLISTENTRYPROTOCOL = 122**

Blacklist entry IP protocol

**BLACKLISTENTRYSOURCEIP = 118**

Blacklist entry source IP address

**BLACKLISTENTRYSOURCEIPMASK = 119**

Blacklist entry source IP address mask

**BLACKLISTENTRYSOURCEIPPREFIXLEN = 172**

Blacklist entry source IP address prefix length

**BLACKLISTENTRYSOURCEPORT = 123**

Blacklist entry source port

**BLACKLISTENTRYSOURCEPORTRANGE = 124**

Blacklist entry source port range end

**BLACKLISTER = 128**

Blacklister

**BLOCK\_LISTENTRYDESTINATIONIP = 120**

block\_list entry destination IP address

**BLOCK\_LISTENTRYDESTINATIONIPMASK = 121**

block\_list entry destination IP address mask

**BLOCK\_LISTENTRYDESTINATIONPORT = 125**

block\_list entry destination port

**BLOCK\_LISTENTRYDESTINATIONPORTRANGE = 126**

block\_list entry destination port range end

**BLOCK\_LISTENTRYDURATION = 127**

block\_list entry duration

**BLOCK\_LISTENTRYID = 117**

None

**BLOCK\_LISTENTRYPROTOCOL = 122**

block\_list entry IP protocol

**BLOCK\_LISTENTRYSOURCEIP = 118**

block\_list entry source IP address

**BLOCK\_LISTENTRYSOURCEIPMASK = 119**  
block\_list entry source IP address mask

**BLOCK\_LISTENTRYSOURCEIPPREFIXLEN = 172**  
block\_list entry source IP address prefix length

**BLOCK\_LISTENTRYSOURCEPORT = 123**  
block\_list entry source port

**BLOCK\_LISTENTRYSOURCEPORTRANGE = 124**  
block\_list entry source port range end

**BLOCK\_LISTER = 128**  
block\_lister

**CIPHERALG = 536**  
Cipher algorithm

**CLIENTIPADDRESS = 403**  
Address of client causing event

**COMPID = 3**  
The identifier of the creator of the log entry.

**CONNDIRECTION = 310**  
Connection direction

**CONNECTEDMACADDR = 447**  
Connected MAC addresses

**CONNECTIVITY = 306**  
Connectivity

**CONNSTATUS = 309**  
Connection status

**CONNTYPE = 308**  
Connection type

**CONTAINEDDATATAG = 483**  
Contained data type tag

**CONTROLCOMMANDID = 28**  
None

**DATATAG = 481**  
Data type tag

**DATATAGS = 485**  
Data tags concerning the record

**DATATYPE = 34**  
Data type

**DHCPLEASEEXPIRES = 528**  
DHCP\_LEASE\_EXPIRES

**DHCPLEASEGW = 529**  
DHCP\_LEASE\_GW

**DHCPLEASEIP = 530**  
DHCP\_LEASE\_IP

**DHCPLEASENETMASK = 531**  
DHCP\_LEASE\_NETMASK

**DHCPLEASEPREFIXLEN = 498**  
DHCP\_LEASE\_PREFIXLEN

**DHCPLEASERECEIVED = 532**  
DHCP\_LEASE\_RECEIVED

**DHCPLEASES = 527**  
DHCP\_LEASES

**DPD = 543**  
Dead Peer Detection

**DPORT = 10**  
Connection destination protocol port

**DSCPMARK = 130**  
DSCP Mark

**DST = 8**  
Connection destination IP address

**DSTADDRS = 20008**  
Destination addresses

**DSTIF = 13**  
Destination interface of firewall

**DSTIPRANGE = 526**  
Destination IP Range

**DSTVLAN = 113**  
Destination VLAN

**DSTZONE = 47**  
Connection destination interface zone

**ELEMENTDOMAIN = 415**  
Administrative Domain of Associated Element

**ENDPOINT = 504**  
Local VPN end point

**ENTERPRISEOID = 493**  
Enterprise OID

**EVENT = 6**  
Logged event

**EVENTADDRESS = 705**  
Notification destination

**EVENTINFO = 701**  
Description for event

**EVENTLOGID = 702**  
Data Identifier of the alert

**EVENTTIME = 700**  
Time stamp of the alert

**EVENTTYPE = 703**  
Type of event

**EVENTUSER = 704**  
User who executed the action

**EXPIRATIONTIME = 534**  
VPN SA expiration time

**FACILITY = 22**  
Firewall subsystem

**FILETYPECOMPAT = 56**  
The type of the file that caused sending this event.

**FLAG = 114**  
None

**FPCACHED = 57**  
Fingerprint match came from fingerprinting cache.

**FW100INTERFACE = 431**  
FW100 Interface

**FW100TRAFFICCOUNTERS = 430**  
Fw100 Traffic counters

**FWACCEPTEDBYTES = 326**  
FW\_ACCEPTED\_BYTES

**FWACCEPTEDPACKETS = 327**  
FW\_ACCEPTED\_PACKETS

**FWACCOUNTEDBYTES = 336**  
FW\_ACCOUNTED\_BYTES

**FWACCOUNTEDPACKETS = 337**  
FW\_ACCOUNTED\_PACKETS

**FWADSLRXBYTES = 417**  
FW\_ADSL\_RX\_BYTES

**FWADSLTXBYTES = 416**  
FW\_ADSL\_TX\_BYTES

**FWDECRYPTEDBYTES = 332**  
FW\_DECRYPTED\_BYTES

**FWDECRYPTEDPACKETS = 333**  
FW\_DECRYPTED\_PACKETS

**FWDROPPEDBYTES = 328**  
FW\_DROPPED\_BYTES

**FWDROPPEDPACKETS = 329**  
FW\_DROPPED\_PACKETS

**FWENCRYPTEDBYTES = 330**  
FW\_ENCRYPTED\_BYTES

**FWENCRYPTEDPACKETS = 331**  
FW\_ENCRYPTED\_PACKETS

**FWFORWARDEDBYTES = 419**  
FW\_FORWARDED\_BYTES

**FWFORWARDEDPACKETS = 418**  
FW\_FORWARDED\_PACKETS

**FWINTERFACEKEY = 340**  
FW\_INTERFACE\_KEY

**FWNATTEDBYTES = 334**  
FW\_NATTED\_BYTES

**FWNATTEDPACKETS = 335**  
FW\_NATTED\_PACKETS

**FWRECEIVEDBYTES = 322**  
FW\_RECEIVED\_BYTES

**FWRECEIVEDPACKETS = 323**  
FW\_RECEIVED\_PACKETS

**FWSENTBYTES = 324**  
FW\_SENT\_BYTES

**FWSENTPACKETS = 325**  
FW\_SENT\_PACKETS

**FWTRAFFIC = 342**  
FW Traffic

**FWTRAFFICACCOUNTEDBYTES = 352**  
Accounted Bytes

**FWTRAFFICACCOUNTEDPACKETS = 346**  
Accounted Packets

**FWTRAFFICALLOWEDBYTES = 349**  
Allowed Bytes

**FWTRAFFICALLOWEDPACKETS = 343**  
Allowed Packets

**FWTRAFFICDISCARDEDBYTES = 350**

Discarded Bytes

**FWTRAFFICDISCARDEDPACKETS = 344**

Discarded Packets

**FWTRAFFICENCRYPTEDBYTES = 354**

Encrypted Bytes

**FWTRAFFICENCRYPTEDPACKETS = 348**

Encrypted Packets

**FWTRAFFICLOGGEDBYTES = 351**

Logged Bytes

**FWTRAFFICLOGGEDPACKETS = 345**

Logged Packets

**FWTRAFFICNATTEDBYTES = 353**

Natted Bytes

**FWTRAFFICNATTEDPACKETS = 347**

Natted Packets

**GENERICTRAPTYPE = 494**

Generic Trap Type

**HASHALG = 538**

Hash Algorithm

**HITS = 48**

HITS

**HTTPREQUESTHOST = 1586**

HTTP request host

**ICMPCODE = 101**

ICMP code attribute

**ICMPID = 102**

ICMP identifier

**ICMPTYPE = 100**

ICMP type attribute

**IKEDHGROUP = 901**

Diffie-Hellman Group

**IKELOCALID = 540**

Local IKE ID

**IKEREMOTEID = 541**

Remote IKE ID

**IKEV1MODE = 542**

IKEv1 negotiation mode

**INCIDENTCASE = 411**

Incident Case

**INFMSG = 19**

Information Message

**INTERFACE = 35**

Interface

**IPCOMPRESSION = 546**

IP Compression

**IPSAPPID = 134**

Network application detected in the connection

**IPSECSSPI = 103**

Inbound IPsec SPI value (hexadecimal)

**LOGID = 2**

Data Identifier

**LOGIFTOPDESTINATIONIPADDRS = 446**

Amount of traffic flowing to the most used destination IP addresses per logical interface

**LOGIFTOPSOURCEIPADDRS = 445**

Amount of traffic originating from the most used source IP addresses per logical interface

**LOGSEVERITY = 805**

Severity

**LONGMSG = 601**

Long field description of alert

**MACALG = 537**

MAC Algorithm

**MESSAGEID = 804**

Message Id

**NATBALANCEID = 393**

NAT\_BALANCE\_ID

**NATDPORT = 18**

Translated packet destination port

**NATDST = 16**

Translated packet destination IP address

**NATMAPID = 394**

NAT\_MAP\_ID

**NATRULEID = 21**

The rule number of the rule that led to the log creation

**NATSPORT = 17**

Translated packet source protocol port

**NATSRC = 15**

Translated packet source IP address

**NATT = 544**

NAT Traversal

**NEGOTIATIONROLE = 539**  
SA Negotiation Role

**NEIGHBORINTERFACE = 733**  
Interface

**NEIGHBORL2DATA = 736**  
Mac address

**NEIGHBORL3DATA = 735**  
IP Address

**NEIGHBORPROTOCOL = 734**  
Protocol

**NEIGHBORSTATE = 737**  
State

**NODECAPACITY = 321**  
Capacity

**NODECONFIGURATION = 304**  
Current configuration

**NODECONFIGURATIONTIMESTAMP = 305**  
Configuration upload time

**NODEDYNUP = 303**  
Update package level

**NODEHWSTATUS = 315**  
Node hardware status

**NODEID = 4**  
Firewall or server node that passes this information

**NODELOAD = 320**  
Node load

**NODESTATUS = 300**  
Node status

**NODEVERSION = 301**  
Node version

**NONCONTAINEDDATATAG = 484**  
Non-contained data type tag

**NUMALERTRESPONSES = 365**  
Number of alert responses performed by this engine

**NUMBLACKLISTRESPONSES = 369**  
Number of blacklist responses performed by this engine

**NUMBLOCK\_LISTRESPONSES = 369**  
Number of block\_list responses performed by this engine

**NUMBYTESRECEIVED = 12201**

Number of bytes received, used for VPN

**NUMBYTESENT = 12200**

Number of bytes sent, used for VPN

**NUMDISCARDRESPONSES = 368**

Number of discard responses performed by this engine

**NUMLOGEVENTS = 363**

Number of log events

**NUMLOGRESPONSES = 364**

Number of log responses performed by this engine

**NUMPACKETSRECEIVED = 549**

Number of packets received

**NUMPACKETSENT = 548**

Number of packets sent

**NUMRECORDRESPONSES = 366**

Number of record responses performed by this engine

**NUMRESETRESPONSES = 367**

Number of reset responses performed by this engine

**OBJECTDN = 410**

User and Group Information

**OBJECTKEY = 409**

Element Id

**OBJECTNAME = 407**

Elements being manipulated

**OBJECTTYPE = 408**

Element Type

**ORIGINNAME = 400**

Name of component producing event

**OUTBOUNDSPI = 533**

Outbound IPsec SPI value (hexadecimal)

**PASSEDBYTES = 388**

PASSED\_BYTES

**PEERCOMPONENTID = 307**

Peer component id

**PEERENDPOINT = 506**

Peer VPN end point

**PEERSECURITYGATEWAY = 505**

Peer VPN gateway

**PFSDHGROUP = 547**  
PFS Diffie-Hellman Group

**PHASE1FAIL = 511**  
IKE\_PHASE1\_FAIL

**PHASE1SUCC = 510**  
IKE\_PHASE1\_SUCC

**PHASE2FAIL = 513**  
IKE\_PHASE2\_FAIL

**PHASE2SUCC = 512**  
IKE\_PHASE2\_SUCC

**POTENTIALLYDUPLICATERESPONSE = 170**  
Potentially duplicate correlation response

**PROBEFAIL = 500**  
PROBE\_FAIL

**PROBEOK = 399**  
PROBE\_OK

**PROTOCOL = 11**  
IP protocol

**QOSCLASS = 129**  
QoS Class

**QOSPRIORITY = 131**  
QoS Priority

**RADIUSACCOUNTINGTYPE = 851**  
Radius Accounting Type

**RECEIVEDLOGEVENTS = 361**  
RECEIVED\_LOG\_EVENTS

**RECEPTIONTIME = 24**  
Reception Time on the log Server

**RESOURCE = 806**  
Resource

**RESULT = 405**  
Result state

**RETSRCIF = 49**  
Return source interface of the connection

**ROUTEBGPPATH = 167**  
Active BGP path

**ROUTEDISTANCE = 162**  
Relative distance for route validation

**ROUTE\_GATEWAY = 164**

IP address of the gateway for the route

**ROUTE\_METRIC = 163**

Protocol specific metric value

**ROUTE\_NETMASK = 161**

Netmask address of the network

**ROUTE\_NETWORK = 160**

Network address of the network

**ROUTE\_OSPF\_LSATYPE = 166**

Type of OSPF LSA's

**ROUTE\_TYPE = 165**

Type of route

**RTT = 109**

Round trip time of connection establishing

**RULE\_COUNTERS = 412**

RULE\_COUNTERS

**RULE\_HITS = 413**

RULE\_HITS

**RULE\_ID = 20**

Rule tag value of acceptance rule

**RWP\_HTTP\_REFERER = 832**

HTTP Referrer

**RWP\_HTTP\_USER\_AGENT = 830**

HTTP User Agent

**RWP\_SERVICE\_NAME = 831**

SSL VPN Portal Service Name

**SAAUTH\_ALG = 520**

SA\_AUTH\_ALG

**SABUNDLE = 514**

SA\_BUNDLE

**SACIPHER\_ALG = 518**

SA\_CIPHER\_ALG

**SACLASS = 535**

SA Type

**SACOMPRESSIONALG = 519**

SA\_COMPRESSION\_ALG

**SAEXPIREHARDLIMIT = 524**

SA\_EXPIRE\_HARDLIMIT

**SAEXPIRESOFTLIMIT = 523**  
SA\_EXPIRE\_SOFTLIMIT

**SAINCOMING = 517**  
SA\_INCOMING

**SAKBHARDLIMIT = 522**  
SA\_KB\_HARDLIMIT

**SAKBSOFTLIMIT = 521**  
SA\_KB\_SOFTLIMIT

**SARESPONDER = 516**  
SA\_RESPONDER

**SATYPE = 515**  
SA\_TYPE

**SECURITYGATEWAY = 502**  
VPN gateway

**SELECTEDCACHE = 396**  
SELECTED\_CACHE

**SELECTEDRTT = 395**  
SELECTED\_RTT

**SENDER = 5**  
None

**SENDERDOMAIN = 38**  
Administrative Domain of Event Sender

**SENDERTYPE = 31**  
Sender type

**SENSORALLOWEDINSPECTEDTCPCONNECTIONS = 437**

**SENSORALLOWEDINSPECTEDUDPCONNECTIONS = 438**

**SENSORALLOWEDUNINSPECTEDTCPCONNECTIONS = 439**

**SENSORALLOWEDUNINSPECTEDUDPCONNECTIONS = 440**

**SENSORDISCARDEDTCPCONNECTIONS = 441**

**SENSORDISCARDEDUDPCONNECTIONS = 442**

**SENSORINSPECTEDBYTES = 357**  
Bytes inspected by sensor

**SENSORINSPECTEDPACKETS = 358**  
Packets inspected by sensor

**SENSORINTERFACEKEY = 370**  
Sensor interface key

**SENSORLOSTBYTES = 359**  
Bytes lost in sensor

**SENSORLOSTPACKETS = 360**  
Packets lost in sensor

**SENSORPROCESSEDBYTES = 355**  
Bytes processed by sensor

**SENSORPROCESSEDPACKETS = 356**  
Packets processed by sensor

**SENSORRECEIVEDBYTES = 338**  
Bytes received by sensor

**SENSORRECEIVEDPACKETS = 339**  
Packets received by sensor

**SENSORTRAFFIC = 372**  
Sensor traffic

**SENSORTRAFFICCLOSEDTCPCONNECTIONS = 383**  
Closed TCP Connections

**SENSORTRAFFICINSPECTEDPACKETS = 376**  
Inspected Packets

**SENSORTRAFFICLOSTPACKETS = 375**  
Lost Packets

**SENSORTRAFFICNEWTCPCONNECTIONS = 381**  
New TCP Connections

**SENSORTRAFFICNUMBEROFALERTS = 380**  
Number of Alerts

**SENSORTRAFFICOKCONNECTIONS = 378**  
OK Connections

**SENSORTRAFFICPROCESSEDBYTES = 374**  
Processed Bytes

**SENSORTRAFFICPROCESSEDPACKETS = 373**  
Processed Packets

**SENSORTRAFFICSTATSOFPACKETS = 377**  
Stats Of Packets

**SENSORTRAFFICSUSPICIOUSCONNECTIONS = 379**  
Suspicious Connections

**SENSORTRAFFICTCPHANDSHAKES = 382**  
TCP Handshakes

**SENSORTRAFFICTCPTIMEOUTS = 384**  
TCP Timeouts

**SENTLOGEVENTS = 362**  
SENT\_LOG\_EVENTS

**SERVICEKEY = 132**

Service primary key, used in service resolving

**SESSIONDOMAIN = 414**

Administrative Domain of Login Session

**SESSIONEVENT = 302**

Session monitoring event code (1 = new, 2 = update, 3 = remove, 4 = all sessions sent)

**SESSIONID = 802**

Id of the User Session

**SFPINGRESS = 900**

SFP\_INGRESS

**SHAPINGCLASS = 386**

SHAPING\_CLASS

**SHAPINGGUARANTEE = 389**

SHAPING\_GUARANTEE

**SHAPINGLIMIT = 390**

SHAPING\_LIMIT

**SHAPINGPRIORITY = 391**

SHAPING\_PRIORITY

**SITCATEGORY = 37**

The type of the situation that caused sending this event.

**SITUATION = 1000**

The identifier of the situation that caused sending this event.

**SNMPRETSRCIF = 51**

SNMP index of return source interface

**SNMPSRCIF = 50**

SNMP index of source interface

**SNMPTRAPMAP = 490**

SNMP Trap

**SNMPTRAPOID = 491**

SNMP Trap OID

**SNMPTRAPVALUE = 492**

SNMP Trap Value

**SPORT = 9**

Connection source protocol port

**SRC = 7**

Connection source IP address

**SRCADDRESS = 398**

SRC\_ADDRESS

**SRCADDRS = 20007**

Source addresses

**SRCIF = 12**  
Source interface of firewall

**SRCIPRANGE = 525**  
Source IP Range

**SRCVLAN = 112**  
Source VLAN

**SRCZONE = 46**  
Connection source interface zone

**SRVHELPERID = 110**  
Protocol agent identification

**SSLVPNSESSIONMONID = 811**  
Id of the User Session

**SSLVPNSESSIONMONRECEIVED = 809**  
Node's local time when the SSL VPN session was created

**SSLVPNSESSIONMONTIMEOUT = 810**  
Node's local time when the SSL VPN session will time-out

**SSLVPNSESSIONTYPETYPE = 808**  
SSL VPN session client type

**STATE = 116**  
Connection state in connection monitoring

**STATUSTYPE = 311**  
Status type

**STORAGESERVERID = 30**  
Storage Server

**SYSLOGTYPE = 111**  
Syslog message type

**TAGINFO = 480**  
Type tags

**TCPDUMPSTATUS = 318**  
TCPDump Monitoring Status

**TCPENCAPSULATION = 545**  
TCP Encapsulation

**TIMEOUT = 115**  
Connection timeout in connection monitoring

**TIMESTAMP = 1**  
Time of creating the event record.

**TLSALERTDESCRIPTION = 45**  
TLS/SSL Alert Message Description

**TLSALERTLEVEL = 44**  
TLS/SSL Alert Message Alert Level

**TLSCERTIFICATEVERIFYERRORCODE = 39**  
TLS/SSL Certificate verify error code

**TLSCIPHERSUITE = 42**  
TLS/SSL cipher suite

**TLSCOMPRESSIONMETHOD = 43**  
TLS/SSL compression method

**TLSDETECTED = 136**  
The connection uses SSL/TLS protocol.

**TLSDOMAIN = 40**  
Domain name field in SSL/TLS certificate

**TLSPROTOCOLVERSION = 41**  
TLS/SSL protocol version

**TOTALBYTES = 387**  
TOTAL\_BYTES

**TPACCEPTEDBYTES = 465**  
TP\_ACCEPTED\_BYTES

**TPACCEPTEDPACKETS = 466**  
TP\_ACCEPTED\_PACKETS

**TPDROPPEDBYTES = 467**  
TP\_DROPPED\_BYTES

**TPDROPPEDPACKETS = 468**  
TP\_DROPPED\_PACKETS

**TPMEMUSAGE = 470**  
Third party memory usage

**TPNODELOAD = 469**  
Third party device load

**TPRECEIVEDBYTES = 461**  
TP\_RECEIVED\_BYTES

**TPRECEIVEDPACKETS = 462**  
TP\_RECEIVED\_PACKETS

**TPSENTBYTES = 463**  
TP\_SENT\_BYTES

**TPSENTPACKETS = 464**  
TP\_SENT\_PACKETS

**TPTRAFFICCOUNTERS = 460**  
Third party traffic counters

**TRAFFICCOUNTERS = 319**  
Traffic counters

**TRAFFICSHAPING = 385**  
TRAFFIC\_SHAPING

**TRANSIENT = 26**  
None

**TUNNELINGLEVEL = 95**  
Number of tunneling protocol layers encapsulating this protocol layer

**TYPE = 23**  
Log event severity type

**TYPEDescription = 404**  
Description of the event

**URLCATEGORYGROUP = 53**  
The type of the URL that caused sending this event.

**URLCATEGORYRISK = 55**  
The risk of the URL that caused sending this event.

**USERNAME = 3001**  
Username if present

**USERORIGINATOR = 401**  
Administrator causing event

**USERROLE = 402**  
Roles of Administrator causing event

**VPNBYTESRECEIVED = 509**  
VPN\_BYTES\_RECEIVED

**VPNBYTESSENT = 508**  
VPN\_BYTES\_SENT

**VPNID = 501**  
Desination VPN

**VPNSRCID = 499**  
Source VPN

**VPNSTATISTICS = 507**  
VPN\_STATISTICS

**VPNSTATUS = 503**  
VPN\_STATUS

**VPNTYPE = 611**  
VPN\_TYPE

**VULNERABILITYREFERENCES = 20000**  
Generated from situation and original situation.

**WIRELESSCHANNEL = 448**  
Wireless Access Point's channel

**WIRELESSCONNECTIONS = 436**  
Number of wireless connections

**WIRELESSMONITORING = 432**

Wireless Monitoring

**WIRELESSECURITY = 435**

Wireless Security mode

**WIRELESSSSID = 433**

Wireless SSID

**WIRELESSSTATUS = 434**

Wireless Status

**ZIEXPORTFILE = 420**

Snapshot of element being manipulated

### 13.1.2.5 Formatters

Custom formats used to return data in different formats. These are used from the query itself when calling the `fetch_as_format()` method. For example, returning a `LogQuery` as a table:

```
query = LogQuery(fetch_size=200)
for log in query.fetch_batch(): # Default is TableFormat
    print(log)
```

As CSV:

```
query = LogQuery(fetch_size=200)
for log in query.fetch_batch(CSVFormat):
    print(log)
```

Each format also allows the ability to customize the fields that should be in the output. By default, each query type in `smc_monitoring.monitors` will have a class attribute `field_ids` which specify the default fields. These can be customized by modifying the `query.format.field_ids([...])` parameter.

For example, modifying a routing query to return only destination interface and the route network:

```
query = RoutingQuery('sg_vm')
query.format.field_ids([LogField.DSTIF, LogField.ROUTENETWORK])
for log in query.fetch_batch():
    ...
```

The same `field_id` customization applies to all query types.

A simple way to view results is to use a `RawDictFormat`:

```
query = LogQuery(fetch_size=3)
query.format.field_names(['Src', 'Dst'])
for record in query.fetch_batch(RawDictFormat):
    ...
```

It is also possible to provide your own formatter. At a minimum you must provide a method called `formatted` in your class. The custom class should extend `_Header` to support custom `field_ids` within the query.

**Note:** Constants are defined in `smc_monitoring.models.constants`. Although there are many field values, not all field values will return results for every query. It is sometimes useful to log in to the Management Client to verify

available fields.

---

**class** `smc_monitoring.models.formatters.CSVFormat(query)`

Bases: `_Header`

Return the results in CSV format. The first line will be a comma separated string with the field header. This is an iterable that will return results in batches of 200 (max) per iteration.

**class** `smc_monitoring.models.formatters.ElementFormat(query)`

Bases: `RawDictFormat`

Return the data as a list in Element format.

**exception** `smc_monitoring.models.formatters.InvalidFieldFormat`

Bases: `Exception`

If using a complex format type such as combined, formatters are not supported. These specialized formats must be returned in raw dict format as they've been customized to return the data in a specific way.

**class** `smc_monitoring.models.formatters.RawDictFormat(query)`

Bases: `object`

Return the data as a list in raw dict format. The results are not filtered with exception of the returned fields based on `field_id` filters. This is a convenience format for consistency, although you can also call the `smc_monitoring.models.query.Query.fetch_raw` method to get the same data.

**class** `smc_monitoring.models.formatters.TableFormat(query)`

Bases: `_Header`

Return the data in a table format. The `field_id` values will be used for the table header. Spacing will be calculated for each batch of results to align the table. The base spacing is determined by the header width, but adjusted wider if the data returned is wider. Anytime there is an adjustment to the width, a new table header will also be printed to visually realign. The query will return a max of 200 batch results per iteration.

---

**Note:** Table alignment will likely not be exact between batches as width is calculated per batch.

---

### 13.1.2.6 TimeRanges

Time formats are optionally used in a `LogQuery` to specify custom ranges for which to search 'stored' log events.

When adding a time format to a query, the `start_time` and `end_time` values need to be in milliseconds. The engine logs are stored in UTC time but in order to display the client side dates properly, you should set a `timezone` on the query.

There are helper methods to simplify retrieving for `last_XXX` period of time as well as custom range formats.

Set up a query with a time format:

```
query = LogQuery(fetch_size=50)
query.format.timezone('Europe/Helsinki')
query.time_range.last_five_minutes()
```

See also:

`custom_range()` for more examples on creating custom time range formats.

```
class smc_monitoring.models.calendar.TimeFormat(start_ms=0, end_ms=0)
```

Bases: `object`

Construct a time format to control the start and end times for a query. If unspecified, results will be limited by the fetch size quantity only. Helper methods are provided to simplify adding time based filters once the instance is constructed.

#### Parameters

- **start\_ms** (`int`) – datetime object in milliseconds. Where to start the query in time. If your search should go backwards in time, specify the oldest time/date in start\_time.
- **end\_ms** (`int`) – datetime object in milliseconds. Where to end the query in time.

```
custom_range(start_time, end_time=None)
```

Provide a custom range for the search query. Start time and end time are expected to be naive `datetime` objects converted to milliseconds. When submitting the query, it is strongly recommended to set the timezone matching the local client making the query.

Example of finding all records on 9/2/2017 from 06:25:30 to 06:26:30 in the local time zone CST:

```
dt_start = datetime(2017, 9, 2, 6, 25, 30, 0)
dt_end = datetime(2017, 9, 2, 6, 26, 30, 0)

query = LogQuery()
query.format.timezone('CST')
query.time_range.custom_range(
    datetime_to_ms(dt_start),
    datetime_to_ms(dt_end))

for record in query.fetch_batch():
    print(record)
```

Last two minutes from current (py2):

```
now = datetime.now()
start_time = int((now - timedelta(minutes=2)).strftime('%s'))*1000
```

Specific start time (py2):

```
p2time = datetime.strptime("1.8.2017 08:26:42,76",
                           "%d.%m.%Y %H:%M:%S,%f").strftime('%s')
p2time = int(s)*1000
```

Specific start time (py3):

```
p3time = datetime.strptime("1.8.2017 08:40:42,76", "%d.%m.%Y %H:%M:%S,%f")
p3time.timestamp() * 1000
```

#### Parameters

- **start\_time** (`int`) – search start time in milliseconds. Start time represents the oldest timestamp.
- **end\_time** (`int`) – search end time in milliseconds. End time represents the newest timestamp.

**property end\_time**

Return the end time in datetime format. Will return 0 if end time is not specified.

**Return type**  
datetime

**last\_day()**

Add time filter from current time back 1 day

**last\_fifteen\_minutes()**

Add time from current time back 15 minutes

**last\_five\_minutes()**

Add time from current time back 5 minutes

**last\_hour()**

Add time from current time back 1 hour

**last\_thirty\_minutes()**

Add time from current time back 30 minutes

**last\_week()**

Add time filter from current time back 7 days.

**property start\_time**

Return the start time in datetime format. Will return 0 if start time is not specified.

**Return type**  
datetime

**smc\_monitoring.models.calendar.datetime\_from\_ms(*ms*)**

Convenience to return datetime from milliseconds

**Returns**  
datetime from ms

**Return type**  
datetime

**smc\_monitoring.models.calendar.datetime\_to\_ms(*dt*)**

Convert an unaware datetime object to milliseconds. This datetime should be the time you would expect to see on the client side. The SMC will do the timestamp conversion based on the query timezone.

**Returns**  
value representing the datetime in milliseconds

**Return type**  
`int`

**smc\_monitoring.models.calendar.subtract\_from\_now(*td*)**

Subtract timedelta from current time

### 13.1.3 Monitors

The monitors package provides modules that represent individual monitoring areas within the SMC monitoring API. Each monitor type extends `smc_monitoring.models.query.Query` to provide a consistent API for adding filters and executing queries.

#### 13.1.3.1 Blacklist

Blacklist Query provides the ability to view current blacklist entries in the SMC by target. Target is defined as the cluster or engine. Retrieved results will have a reference to the entry and hence be possible to remove the entry.

```
query = BlacklistQuery('sg_vm')
query.format.timezone('CST')
```

Optionally add an “InFilter” to restrict search to a specific field:

```
query.add_in_filter(
    FieldValue(LogField.BLACKLISTENTRYSOURCEIP), [IPValue('2.2.2.2')])
```

An InFilter can also use a network based syntax:

```
query.add_in_filter(
    FieldValue(LogField.BLACKLISTENTRYSOURCEIP), [IPValue('2.2.2.0/24')])
```

Or combine filters using “AndFilter” or “OrFilter”. Find an entry with source IP 2.2.2.2 OR 2.2.2.5:

```
ip1 = InFilter(FieldValue(LogField.BLACKLISTENTRYSOURCEIP), [IPValue('2.2.2.2')])
ip2 = InFilter(FieldValue(LogField.BLACKLISTENTRYSOURCEIP), [IPValue('2.2.2.5')])
query.add_or_filter([in_filter, or_filter])
```

Get the results of the query in the default TableFormat:

```
for entry in query.fetch_batch():
    print(entry)
```

Delete any blacklist entries with a source IP within a network range of 3.3.3.0/24:

```
query = BlacklistQuery('sg_vm')
query.add_in_filter(
    FieldValue(LogField.BLACKLISTENTRYSOURCEIP), [IPValue('3.3.3.0/24')])

for record in query.fetch_as_element(): # <-- must get as element to obtain delete()
    ↪method
    record.delete()
```

See also:

`smc_monitoring.models.filters` for more information on creating filters

**class** `smc_monitoring.monitors.blacklist.BlacklistEntry` (\*\*kw)

Bases: `object`

A blacklist entry represents an entry in the engines kernel table indicating that a source/destination/port/protocol mapping is currently being blocked by the engine. To remove a blacklist entry from an engine, retrieve all entries as element and remove the entry of interest by called `delete` on the element.

The simplest way to use search filters with a blacklist entry is to examine the BlacklistQuery `field_ids` and use these constant fields as InFilter definitions on the query.

**property blacklist\_entry\_key**

Blacklist entry Key. Needed to remove the entry

**Return type**

`str`

**property blacklist\_id**

Blacklist entry ID. Useful if you want to locate the entry within the Management Client.

**Return type**

`str`

**delete()**

Delete the entry from the engine where the entry is applied.

**Raises**

`DeleteElementFailed`

**Returns**

`None`

**property dest\_ports**

Destination ports for this blacklist entry. If no ports are specified, 'ANY' is returned.

**Return type**

`str`

**property destination**

Destination network/netmask for this blacklist entry.

**Return type**

`str`

**property duration**

Duration for the blacklist entry.

**Return type**

`int`

**property engine**

The engine for this blacklist entry.

**Return type**

`str`

**property first\_fetch**

first fetch True means entry is part of initial data at first fetch

**Return type**

`bool`

**property href**

The href for this blacklist entry. This is the reference to the entry for deleting the entry.

**Return type**

`str`

**property protocol**

Specified protocol for the blacklist entry. If none is specified, 'ANY' is returned.

**Return type**

`str`

**property source**

Source address/netmask for this blacklist entry.

**Return type**

`str`

**property source\_ports**

Source ports for this blacklist entry. If no ports are specified (i.e. ALL ports), 'ANY' is returned.

**Return type**

`str`

**property timestamp**

Timestamp when this blacklist entry was added.

**Return type**

`str`

```
class smc_monitoring.monitors.blacklist.BlacklistQuery(target, timezone=None, **kw)
```

Bases: [`Query`](#)

Query existing blacklist entries for a given cluster/engine. It is generally recommended to set your local timezone when making a query to convert the timestamp into a relevant format.

**Parameters**

- **target** (`str`) – NAME of the engine or cluster
- **timezone** (`str`) – timezone for timestamps.

---

**Note:** Timezone can be in the following formats: 'US/Eastern', 'PST', 'Europe/Helsinki'. More example time zone formats are available in the Logs view of the Management Client when you select Tools -> Time Zones.

---

**fetch\_as\_element**(\*\*kw)

Fetch the blacklist and return as an instance of Element. :param int query\_timeout: length of time to wait on receiving web

socket results (total query time).

**Parameters**

- **inactivity\_timeout** (`int`) – length of time before exiting if no new entry.
- **max\_recv** (`int`) – for queries that are not 'live', set this to supply a max number of receive iterations.

**Returns**

generator returning element instances

**Return type**

[`BlacklistEntry`](#)

### 13.1.3.2 Connections

A connection query returns all currently connected sessions on the given target.

Create a query to obtain all connections for a given engine:

```
query = ConnectionQuery('sg_vm')
```

Add a timezone to the query:

```
query.format.timezone('CST')
```

Add a filter to only get connections if the source address is 172.18.1.252:

```
query.add_in_filter(FieldValue(LogField.SRC), [IPValue('172.18.1.252')])
```

Only connections that match a specific service:

```
query.add_in_filter(FieldValue(LogField.SERVICE), [ServiceValue('TCP/443', 'UDP/53')])
```

Execute query and return raw results:

```
for records in query.fetch_raw():  
    ...
```

Execute query and return as an *Connection* element:

```
for records in query.fetch_as_element():  
    ...
```

Retrieving live streaming results:

```
for records in query.fetch_live():  
    ...
```

**See also:**

*smc\_monitoring.models.filters* for more information on creating filters

**class** `smc_monitoring.monitors.connections.Connection(**data)`

Bases: `object`

Connection represents a state table entry. This is the result of making a *ConnectionQuery* and using *fetch\_as\_element()*.

**property** `dest_addr`

Destination address for this entry

**Return type**

`str`

**property** `dest_port`

Destination port for the entry.

**Return type**

`int`

**property engine**

The engine/cluster for this state table entry

**Returns**

engine or cluster for this entry

**Return type**

`str`

**property first\_fetch**

first fetch True means entry is part of initial data at first fetch

**Return type**

`bool`

**property protocol**

Protocol for this entry

**Returns**

protocol (UDP/TCP/ICMP, etc)

**Return type**

`str`

**property service**

Service for this entry

**Returns**

service (HTTP/HTTPS, etc)

**Return type**

`str`

**property source\_addr**

Source address for this entry

**Return type**

`str`

**property source\_port**

Source port for the entry.

**Return type**

`int`

**property state**

State of the connection.

**Returns**

state, i.e. UDP established, TCP established, etc.

**Return type**

`str`

**property timestamp**

Timestamp of this connection. It is recommended to set the timezone on the query to view this timestamp in the systems local time. For example:

```
query.format.timezone('CST')
```

**Returns**

timestamp in string format

**Return type**

`str`

```
class smc_monitoring.monitors.connections.ConnectionQuery(target, **kw)
```

Bases: [`Query`](#)

Show all current connections on the specified target.

**Variables**

**field\_ids** ([`list`](#)) – field IDs are the default fields for this entry type and are constants found in [`smc\_monitoring.models.constants.LogField`](#)

**Parameters**

**target** ([`str`](#)) – name of target engine/cluster

```
fetch_as_element(**kw)
```

Fetch the results and return as a Connection element. The original query is not modified.

**Parameters**

- **query\_timeout** ([`int`](#)) – length of time to wait on receiving web socket results (total query time).
- **inactivity\_timeout** ([`int`](#)) – length of time before exiting if no new entry.
- **max\_recv** ([`int`](#)) – for queries that are not ‘live’, set this to supply a max number of receive iterations.

**Returns**

generator of elements

**Return type**

[`Connection`](#)

### 13.1.3.3 Logs

LogQuery provides an interface to the SMC Log Viewer to retrieve data in real time or by batch.

There are a variety of settings you can configure on a query such as whether to execute a real time query versus a stored log fetch, time frame for the query, fetch size quantity, returned format style, specify which fields to return and adding filters to make a very specific query.

To make queries, first obtain a query object and optionally (recommended) specify a maximum number of records to fetch (for non-real time fetches). The default log query type is ‘stored’, and if a `fetch_size` is not provided, one batch of 200 records will be returned:

```
query = LogQuery(fetch_size=50)
```

If real time logs are preferred and set `fetch_type='current'` (default is fetch ‘stored’ logs):

```
query = LogQuery(fetch_type='current')
```

You can also use the shortcut `fetch_live` on the query:

```
query = LogQuery()
for result in query.fetch_live():
    ...
```

**Note:** If selecting `fetch_size='current'` log queries will be real-time and ignore the `fetch_size`, `time_range`, and `backwards` values if provided on the query.

You can also set a `time_range` on the query. There are convenience methods on a `TimeFormat` object to simplify adding a time range. When using time ranges, you should set the timezone on the query to the clients timezone:

```
query = LogQuery(fetch_size=50)
query.time_range.last_five_minutes()
query.format.timezone('CST')
```

You can also use custom time ranges to search between a specific period of time. This is done by providing a `smc_monitoring.models.calendar.TimeFormat` instance to the Query constructor, or by modifying the query `time_range` attribute. The `TimeFormat` object takes a ‘naive’ datetime object for start and end times. The start and end times must also be in milliseconds.

Example of finding all records on 9/2/2017 from 06:25:30 to 06:26:30 in the local time zone CST:

```
dt_start = datetime(2017, 9, 2, 6, 25, 30, 0)
dt_end = datetime(2017, 9, 2, 6, 26, 30, 0)

query = LogQuery()
query.format.timezone('CST')    # <--- Set the timezone on the query!
query.time_range.custom_range(
    datetime_to_ms(dt_start),
    datetime_to_ms(dt_end))
```

See also:

`smc_monitoring.models.calendar.TimeFormat` for more examples and information on using a `TimeFormat` in a query.

Adding filters to a query can be achieved by using `add_XX_filter` convenience methods or by calling `update_filter` with the filter object.

For example, customizing the fields returned using `query.format.field_ids`, and filtering for only HIGH alerts with a source address of 192.168.4.84:

```
query = LogQuery(fetch_size=10)
query.format.timezone('CST')

query.format.field_ids([LogField.TIMESTAMP, LogField.ACTION, LogField.SRC, LogField.DST])

query.add_and_filter(
    [InFilter(FieldValue(LogField.ALERTSEVERITY), [ConstantValue(Alerts.HIGH)]),
     InFilter(FieldValue(LogField.SRC), [IPValue('192.168.4.84')])])
```

See also:

`smc.monitoring.filters` for information on how to use and combine filters for a query.

**class** `smc_monitoring.monitors.logs.LogQuery`(*fetch\_type='stored', fetch\_size=None, backwards=True, format=None, time\_range=None, \*\*kw*)

Bases: `Query`

Make a Log Query to the SMC to fetch stored log data or monitor logs in real time.

**Variables**

**field\_ids** (*list*) – field IDs are the default fields for this entry type and are constants found in `smc_monitoring.models.constants.LogField`

**Parameters**

- **fetch\_type** (*str*) – ‘stored’ or ‘current’
- **fetch\_size** (*int*) – max number of logs to fetch
- **backwards** (*bool*) – by default records are returned from newest to oldest (backwards=True). To return in opposite direction, set backwards=False. Default: True
- **format** (format type from `smc_monitoring.models.formats` (default: TextFormat)) – A format object specifying format of return data
- **time\_range** (*TimeFormat*) – time filter to add to query
- **servers** (*list[str, Element]*) – A list of href or server elements for which to query

**fetch\_batch**(*formatter=<class 'smc\_monitoring.models.formatters.TableFormat'>*)

Fetch a batch of logs and return using the specified formatter. Formatter is class type defined in `smc_monitoring.models.formatters`. This fetch type will be a single shot fetch (this method forces `fetch_type='stored'`). If `fetch_size` is not already set on the query, the default `fetch_size` will be 200.

**Parameters**

**formatter** – Formatter type for data representation. Any type in `smc_monitoring.models.formatters`.

**Returns**

generator returning data in specified format

**fetch\_live**(*formatter=<class 'smc\_monitoring.models.formatters.TableFormat'>*)

View logs in real-time. If previous filters were already set on this query, they will be preserved on the original instance (this method forces `fetch_type='current'`).

**Parameters**

**formatter** – Formatter type for data representation. Any type in `smc_monitoring.models.formatters`.

**Returns**

generator of formatted results

**fetch\_raw()**

Execute the query and return by batches. Optional keyword arguments are passed to `Query.execute()`. Whether this is real-time or stored logs is dependent on the value of `fetch_type`.

**Returns**

generator of dict results

**property fetch\_size**

Return the fetch size for this query. If fetch size is set to 0, the query will be aborted after the first response message. If the `fetch_size` is None, it is considered undefined which indicates there is no fetch bound set on this query (i.e. fetch all).

**..note::** It is recommended to provide a `fetch_size` to limit the results when doing a ‘stored’ query.

**Returns**

configured fetch size for this query

**Return type**`int`**13.1.3.4 Routes**

Query the current routing table entries.

Create a query to obtain all connections for a given engine:

```
query = RoutingQuery('sg_vm')
```

Add a timezone to the query:

```
query.format.timezone('CST')
```

Add a filter to only routes for destination network 192.168.4.0/24:

```
query.add_in_filter(FieldValue(LogField.ROUTENETWORK), [IPValue('192.168.4.0')])
```

Only routes that use a specific gateway:

```
query.add_in_filter(FieldValue(LogField.ROUTE_GATEWAY), [IPValue('172.18.1.200')])
```

Execute query and return raw results:

```
for records in query.fetch_batch():
    ...
```

Execute query and return as an *RoutingView* element:

```
for records in query.fetch_as_element():
    ...
```

**See also:**

[\*smc\\_monitoring.models.filters\*](#) for more information on creating filters

**class** `smc_monitoring.monitors.routes.RoutingQuery(target, **kw)`

Bases: [\*Query\*](#)

Show all current dynamic and static routes on the specified target.

**Variables**

**field\_ids** (*list*) – field IDs are the default fields for this entry type and are constants found in [\*smc\\_monitoring.models.constants.LogField\*](#)

**Parameters**

**target** (*str*) – name of target engine/cluster

**fetch\_as\_element** (*\*\*kw*)

Fetch the results and return as a *RoutingView* element. The original query is not modified.

**Parameters**

- **query\_timeout** (*int*) – length of time to wait on receiving web socket results (total query time).
- **inactivity\_timeout** (*int*) – length of time before exiting if no new entry.

- **max\_recv** (*int*) – for queries that are not ‘live’, set this to supply a max number of receive iterations.

**Returns**

generator of elements

**Return type**

*RoutingView*

```
class smc_monitoring.monitors.routes.RoutingView(**data)
```

Bases: *object*

A Routing View represents an entry in the current routing table. This is the result of making a *RoutingQuery* and using *fetch\_as\_element()*.

**property dest\_if**

Destination interface for this route

**Return type**

*str*

**property dest\_vlan**

Destination VLAN for this route, if any.

**Return type**

*str*

**property dest\_zone**

Destination zone for this route, if any.

**Return type**

*str*

**property engine**

The engine/cluster for this route

**Return type**

*str*

**property first\_fetch**

first fetch True means entry is part of initial data at first fetch

**Return type**

*bool*

**property route\_gw**

The route gateway for this route.

**Return type**

*str*

**property route\_metric**

Metric for this route.

**Returns**

route metric

**Return type**

*int*

**property route\_network**

The route network for this route.

**Return type**

str

**property route\_type**

The type of route.

**Returns**

Static, Connection, Dynamic, etc.

**Return type**

str

**property timestamp**

Timestamp of this connection. It is recommended to set the timezone on the query to view this timestamp in the systems local time. For example:

```
query.format.timezone('CST')
```

:return timestamp in string format :rtype: str

### 13.1.3.5 SSLVPN

SSLVPN currently connected users.

Create a query to obtain all connections for a given engine:

```
query = SSLVPNQuery('sg_vm')
```

Add a timezone to the query:

```
query.format.timezone('CST')
```

Execute query and return raw results:

```
for records in query.fetch_batch():
    ...
```

Execute query and return as an *SSLVPNUser* element:

```
for records in query.fetch_as_element():
    ...
```

**See also:**

[smc\\_monitoring.models.filters](#) for more information on creating filters

**class** `smc_monitoring.monitors.sslvpn.SSLVPNQuery(target, **kw)`

Bases: [Query](#)

Show all current SSL VPN connections on the specified target.

**Variables**

**field\_ids** (*list*) – field IDs are the default fields for this entry type and are constants found in [smc\\_monitoring.models.constants.LogField](#)

**Parameters**

**target** (*str*) – name of target engine/cluster

**fetch\_as\_element** (\*\*kw)

Fetch the results and return as an SSLVPNUser element. The original query is not modified.

**Parameters**

- **query\_timeout** (*int*) – length of time to wait on receiving web socket results (total query time).
- **inactivity\_timeout** (*int*) – length of time before exiting if no new entry.
- **max\_recv** (*int*) – for queries that are not ‘live’, set this to supply a max number of receive iterations.

**Returns**

generator of elements

**Return type**

*SSLVPNUser*

```
class smc_monitoring.monitors.sslvpn.SSLVPNUser(**data)
```

Bases: *object*

Connection represents a state table entry. This is the result of making a *SSLVPNQuery* and using *fetch\_as\_element()*.

**property engine**

The engine/cluster for this state table entry

**Returns**

engine or cluster for this entry

**Return type**

*str*

**property first\_fetch**

first fetch True means entry is part of initial data at first fetch

**Return type**

*bool*

**property session\_expiration**

Time the session expires. It is recommended that you add a timezone to the query to present this in human readable format:

```
query.format.timezone('CST')
```

**Return type**

*str*

**property session\_start**

Time the session started. It is recommended that you add a timezone to the query to present this in human readable format:

```
query.format.timezone('CST')
```

**Return type**

*str*

**property source\_addr**

Source IP address for the SSL VPN user

**Return type**

str

**property username**

Username for this SSL VPN user

**Return type**

str

### 13.1.3.6 Users

Get active users on target cluster/engine.

Create a query to obtain all users for a given engine:

```
query = UserQuery('sg_vm')
```

Add a timezone to the query:

```
query.format.timezone('CST')
```

Execute query and return raw results:

```
for records in query.fetch_batch():
    ...
```

Execute query and return as a *User* element:

```
for records in query.fetch_as_element():
    ...
```

See also:

[smc\\_monitoring.models.filters](#) for more information on creating filters

**class** smc\_monitoring.monitors.users.**User**(\*\*data)

Bases: *object*

User mapping currently in user cache on specified target. This is the result of making a *UserQuery* and using *fetch\_as\_element()*.

**property domain**

SMC Domain that this user record belongs to

**Returns**

name of SMC domain, 'Shared' is default

**Return type**

str

**property engine**

The engine/cluster for this state table entry

**Returns**

engine or cluster for this entry

**Return type**`str`**property expiration**

Expiration time for this user entry. It is recommended to add a timezone to the query to display this field in the client local time.

**Returns**

expiration time for this user authentication entry

**Return type**`str`**property first\_fetch**

first fetch True means entry is part of initial data at first fetch

**Return type**`bool`**property ipaddress**

IP address for the entry

**Return type**`str`**property timestamp**

Timestamp of this connection. It is recommended to set the timezone on the query to view this timestamp in the systems local time. For example:

```
query.format.timezone('CST')
```

:return timestamp in string format :rtype: str

**property username**

Username for entry

**Returns**

username value as fully qualified domain name

**Return type**`str`

**class** `smc_monitoring.monitors.users.UserQuery(target, **kw)`

Bases: `Query`

Show all authenticated users on the specified target.

**Variables**

**field\_ids** (`list`) – field IDs are the default fields for this entry type and are constants found in `smc_monitoring.models.constants.LogField`

**Parameters**

**target** (`str`) – name of target engine/cluster

**fetch\_as\_element** (`**kw`)

Fetch the results and return as a User element. The original query is not modified.

**Parameters**

- **query\_timeout** (`int`) – length of time to wait on receiving web socket results (total query time).

- **inactivity\_timeout** (*int*) – length of time before exiting if no new entry.
- **max\_recv** (*int*) – for queries that are not ‘live’, set this to supply a max number of receive iterations.

**Returns**

generator of elements

**Return type***User*

### 13.1.3.7 VPNs

Get all active VPN SA's.

Create a query to obtain all connections for a given engine:

```
query = VPNSAQuery('sg_vm')
```

Add a timezone to the query:

```
query.format.timezone('CST')
```

Execute query and return raw results:

```
for records in query.fetch_batch():
    ...
```

Execute query and return as a *VPNSecurityAssoc* element:

```
for records in query.fetch_as_element():
    ...
```

Delete a VPN SA:

```
query = VPNSAQuery('sg_vm')
for sa in query.fetch_as_element():
    sa.delete()
```

**See also:**

*smc\_monitoring.models.filters* for more information on creating filters

**class** *smc\_monitoring.monitors.vpns.VPNSAQuery*(*target*, *\*\*kw*)

Bases: *Query*

Show all current VPN SA's on the specified target.

**Variables**

**field\_ids** (*list*) – field IDs are the default fields for this entry type and are constants found in *smc\_monitoring.models.constants.LogField*

**Parameters**

**target** (*str*) – name of target engine/cluster

**fetch\_as\_element** (*\*\*kw*)

Fetch the results and return as a *VPNSecurityAssoc* element. The original query is not modified.

**Parameters**

- **query\_timeout** (*int*) – length of time to wait on receiving web socket results (total query time).
- **inactivity\_timeout** (*int*) – length of time before exiting if no new entry.
- **max\_recv** (*int*) – for queries that are not ‘live’, set this to supply a max number of receive iterations.

**Returns**

generator of elements

**Return type**

*VPNSecurityAssoc*

```
class smc_monitoring.monitors.vpns.VPNSecurityAssoc(**data)
```

Bases: *object*

A VPN Security Association represents a currently connected VPN endpoint. This is the result of making a *VPNSAQuery* and using *fetch\_as\_element()*.

**property bytes\_received**

Number of bytes received.

**Return type**

*int*

**property bytes\_sent**

Number of bytes sent.

**Return type**

*int*

**property engine**

The engine/cluster for this VPN

**Return type**

*str*

**property expiration**

Expiration time for this tunnel Security Association

**Return type**

*str*

**property first\_fetch**

first fetch True means entry is part of initial data at first fetch

**Return type**

*bool*

**property local\_endpoint**

Local endpoint (IP address) for this VPN tunnel.

**Return type**

*str*

**property local\_gateway**

Local gateway for this VPN.

**Return type**

*str*

**property local\_networks**

Local protected networks

**Return type**

str

**property negotiation\_role**

Role for this tunnel entry.

**Returns**

Negotiation role, i.e. Initiator, Responder, etc.

**Return type**

str

**property peer\_endpoint**

Peer endpoint element and IP Address for this tunnel.

**Return type**

str

**property peer\_gateway**

Peer gateway for this VPN.

**Return type**

str

**property peer\_networks**

Remote protected networks

**Return type**

str

**property protocol**

Which protocol is associated with this tunnel entry.

**Returns**

IP protocol for tunnel, i.e. ESP/UDP

**Return type**

str

**property sa\_type**

SA Type for this VPN tunnel. Each VPN tunnel will typically have at least two entries, one for IPSEC and another for IKE.

**Return type**

str

**property timestamp**

Timestamp of this connection. It is recommended to set the timezone on the query to view this timestamp in the systems local time. For example:

```
query.format.timezone('CST')
```

**Return type**

str

### 13.1.3.8 Alerts

ActiveAlert Query provides the ability to view current alert entries from the alert log viewer. When creating the query, you must specify a target which specifies the SMC domain for which to retrieve the alerts.

A basic alert query using a local timezone example:

```
query = ActiveAlertQuery('Shared Domain')
query.format.timezone('CST')
```

You can also use standard filters to specify a more exact match, for example, showing alerts with a severity of CRITICAL:

```
query.add_in_filter(
    FieldValue(LogField.ALERTSEVERITY), [ConstantValue(Alerts.CRITICAL)])
```

**class** smc\_monitoring.monitors.alerts.ActiveAlertQuery(target='Shared Domain', timezone=None)

Bases: [Query](#)

Active Alert Query is an interface to the alert log viewer in Log Server. This query type provides the ability to fetch and filter on active alerts.

You can create a new query specifying a valid timezone abbreviation:

```
query = ActiveAlertQuery('Shared Domain', timezone='CST')
```

Or alternatively no timezone:

```
query = ActiveAlertQuery('DomainFoo')
```

#### Parameters

- **target** (*str*) – domain for which to filter alerts. Default: 'Shared Domain'
- **timezone** (*str*) – timezone for timestamps, i.e. 'CST', etc

**fetch\_as\_element**(\*\*kw)

Fetch the results and return as a User element. The original query is not modified.

#### Returns

generator returning element instances

#### Return type

[Alert](#)

**class** smc\_monitoring.monitors.alerts.Alert(\*\*data)

Bases: [object](#)

Alert definition returned from specified domain. This is the result of making a [ActiveAlertQuery](#) and using [fetch\\_as\\_element\(\)](#).

#### property action

Action performed for the alert

#### Return type

*str*

**property destination**

Destination IP for the alert

**Return type**

str

**property destination\_port**

Destination port for alert

**Return type**

int

**property engine**

The engine/cluster for this state table entry

**Returns**

engine or cluster for this entry

**Return type**

str

**property protocol**

Protocol for alert

**Return type**

str

**property service**

Service associated with alert

**Return type**

str

**property severity**

Severity for this alert

**Return type**

str

**property situation**

Situation defined for this alert

**Return type**

str

**property source**

Source IP for the alert

**Return type**

str

**property source\_port**

Source port for alert

**Return type**

int

**property timestamp**

Timestamp of this connection. It is recommended to set the timezone on the query to view this timestamp in the systems local time. For example:

```
query.format.timezone('CST')
```

:return timestamp in string format :rtype: str

**property vulnerability\_refs**

Comma seperated string listing any vulnerability references for the alert, if any.

**Return type**

str

## API REFERENCE

### 14.1 Session

**class** `smc.api.session.Session`(*manager=None*)

Session represents the clients session to the SMC. A session is obtained by calling `login()`. If sessions need to be long lived as might be the case when running under a web platform, a session is automatically refreshed when it expires. Best practice is to call `logout()` after to clear the session from the SMC. A session will be automatically closed once the python interpreter closes.

Each session will also have a single connection pool associated with it. This results in a single persistent connection to the SMC that will be re-used as needed.

**property** `api_version`

Current API Version

**Return type**

`str`

**property** `domain`

Logged in SMC domain

**Return type**

`str`

**property** `entry_points`

Entry points that are bound to this session. Entry points are exposed by the SMC API and provide links to top level resources

**Return type**

Resource

**property** `is_active`

Is this session active. Active means there is a stored session ID for the SMC using the current account. This does not specify whether the session ID has been timed out on the server but does indicate the account has not called `logout`.

**Return type**

`bool`

**property** `is_ssl`

Is this an SSL connection

**Return type**

`bool`

```
login(url=None, api_key=None, login=None, pwd=None, api_version=None, timeout=None, verify=True, alt_filepath=None, domain=None, pool_maxsize=None, max_retry=None, **kwargs)
```

Login to SMC API and retrieve a valid session. Sessions use a pool connection manager to provide dynamic scalability during times of increased load. Each session is managed by a global session manager making it possible to have more than one session per interpreter.

An example login and logout session:

```
from smc import session
session.login(url='http://1.1.1.1:8082', api_key='SomeSMCG3ener@t3dPwd')
.....do stuff.....
session.logout()
```

### Parameters

- **url** (*str*) – ip of SMC management server
- **api\_key** (*str*) – API key created for api client in SMC
- **login** (*str*) – Administrator user in SMC that has privilege to SMC API.
- **pwd** (*str*) – Password for user login.
- **api\_version** – specify api version (optional)
- **timeout** (*int*) – (optional): specify a timeout for initial connect; (default 10)
- **verify** (*str/boolean*) – verify SSL connections using cert (default: verify=True) You can pass verify the path to a CA\_BUNDLE file or directory with certificates of trusted CAs
- **alt\_filepath** (*str*) – If using .smcrc, alternate path+filename
- **domain** (*str*) – domain to log in to. If domains are not configured, this field will be ignored and api client logged in to ‘Shared Domain’.
- **retry\_on\_busy** (*bool*) – pass as kwarg with boolean if you want to add retries if the SMC returns HTTP 503 error during operation. You can also optionally customize this behavior and call [set\\_retry\\_on\\_busy\(\)](#)
- **pool\_maxsize** (*int*) – The maximum number of connections to save in the pool.
- **max\_retry** (*int*) – The maximum number of retry.

### Returns

user session name in SessionManager

### Return type

*str*

### Raises

[ConfigLoadError](#) – loading cfg from ~/.smcrc fails

For SSL connections, you can disable validation of the SMC SSL certificate by setting verify=False, however this is not a recommended practice.

If you want to use the SSL certificate generated and used by the SMC API server for validation, set verify='path\_to\_my\_dot\_pem'. It is also recommended that your certificate has subjectAltName defined per RFC 2818

If SSL warnings are thrown in debug output, see: <https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings>

Logout should be called to remove the session immediately from the SMC server.

---

**Note:** As of SMC 6.4 it is possible to give a standard Administrative user access to the SMC API. It is still possible to use an API Client by providing the `api_key` in the login call.

---

**logout()**

Logout session from SMC

**Returns**

None

**property manager**

Return the session manager for this session

**Return type**

SessionManager

**property name**

Return the administrator name for this session. Can be None if the session has not yet been established.

---

**Note:** The administrator name was introduced in SMC version 6.4. Previous versions will show the unique session identifier for this session.

---

**Return type**

str

**refresh()**

Refresh session on 401. This is called automatically if your existing session times out and resends the operation/s which returned the error.

**Raises**

**SMCConnectionError** – Problem re-authenticating using existing api credentials

**property session\_id**

The session ID in header type format. Can be inserted into a connection if necessary using:

```
{'Cookie': session.session_id}
```

**Return type**

str

**set\_retry\_on\_busy**(*total=5, backoff\_factor=0.1, status\_forcelist=None, \*\*kwargs*)

Mount a custom retry object on the current session that allows service level retries when the SMC might reply with a Service Unavailable (503) message. This can be possible in larger environments with higher database activity. You can all this on the existing session, or provide as a dict to the login constructor.

**Parameters**

- **total** (*int*) – total retries
- **backoff\_factor** (*float*) – when to retry
- **status\_forcelist** (*list*) – list of HTTP error codes to retry on
- **method\_whitelist** (*list*) – list of methods to apply retries for, GET, POST and PUT by default

**Returns**

None

**property sock**

get a secure socket from the pool if one is available else get a new connection

**Return type**

SSLSocket

**switch\_domain(domain)**

Switch from one domain to another. You can call `session.login()` with a domain key value to log directly into the domain of choice or alternatively switch from domain to domain. The user must have permissions to the domain or unauthorized will be returned. In addition, when switching domains, you will be logged out of the current domain to close the connection pool associated with the previous session. This prevents potentially excessive open connections to SMC

```
session.login() # Log in to 'Shared Domain'
...
session.switch_domain('MyDomain')
```

**Raises**

**SMCConnectionError** – Error logging in to specified domain. This typically means the domain either doesn't exist or the user does not have privileges to that domain.

**property timeout**

Session timeout in seconds

**Return type**

int

**property url**

The fully qualified SMC URL in use, includes the port number

**Return type**

str

## 14.2 Element

```
class smc.base.model.ElementBase(**meta)
```

Element base provides a meta data container and an instance cache as well as methods to retrieve aspects of an element. Meta is passed in to Element and SubElement types to provide links to resources. When a top level query is made to the SMC API, meta is returned for the element (unless a direct link query is made). The meta format include 'href', 'type', 'name'. For example:

```
"href": "http://1.1.1.1:8082/6.4/elements/host/707", "name": "foobar", "type": "host"
```

Methods of the element classes are designed to expose any links or attributes of the specific element to simplify manipulation. If a method, etc is accessed that requires the elements data, the element is fetched and the elements cache (stored in *data* attribute) is inflated. The ETag is also retained in the element and is used when updating or deleting the element to ensure we are operating on the latest version.

Meta can be passed to constructor through as key value pairs kwargs, `href=...` (only partial meta), or `meta={...}` (as dict)

If meta is not provided, the meta attribute will be None

**delete()**

Delete the element

**Raises**

*DeleteElementFailed* – possible dependencies, record locked, etc

**Returns**

None

**update(\*exception, \*\*kwargs)**

Update the existing element and clear the instance cache. Removing the cache will ensure subsequent calls requiring element attributes will force a new fetch to obtain the latest copy.

Calling update() with no args will assume the element has already been modified directly and the *data* cache will be used to update. You can also override the following attributes: href, etag, json and params. If json is sent, it is expected to be a complete payload to satisfy the update.

For kwargs, if attribute values are a list, you can pass 'append\_lists=True' to add to an existing list, otherwise overwrite the existing (default: overwrite)

**See also:**

To see different ways to utilize this method for updating, see: *Update*.

**Parameters**

- **exception** – pass a custom exception to throw if failure
- **kwargs** – optional kwargs to update request data to server.

**Raises**

- *ModificationFailed* – raised if element is tagged as System element
- *UpdateElementFailed* – failed to update element with reason

**Returns**

href of the element modified

**Return type**

str

```
class smc.base.model.Element(name=None, **meta)
```

Bases: *ElementBase*

Base element with common methods shared by inheriting classes. If stashing attributes on this class, be sure to prefix with an underscore to avoid having the attributes serialized when calling update.

**objects(self):**

Interface to element collections. All classes inheriting from *Element* can access collections through this class property:

```
for host in Host.objects.all():
    ...
```

Fetch a single entry:

```
host = Host.objects.filter('myhost')
...
```

For more information on collections, see: *smc.base.collection.CollectionManager*

**add\_category**(*category*)

Category Tags are used to characterize an element by a type identifier. They can then be searched and returned as a group of elements. If the category tag specified does not exist, it will be created. This change will take effect immediately.

**Parameters**

**tags** (*list*(*str*)) – list of category tag names to add to this element

**Raises**

*ElementNotFound* – Category tag element name not found

**Returns**

None

**See also:**

*smc.elements.other.Category*

**property categories**

Search categories assigned to this element

```
>>> from smc.elements.network import Host
>>> Host('kali').categories
[Category(name=foo), Category(name=foocategory)]
```

**Return type**

*list*(*Category*)

**property comment**

Comment for element

**duplicate**(*name*)

New in version 0.5.8: Requires SMC version >= 6.3.2

Duplicate this element. This is a shortcut method that will make a direct copy of the element under the new name and type.

**Parameters**

**name** (*str*) – name for the duplicated element

**Raises**

*ActionCommandFailed* – failed to duplicate the element

**Returns**

the newly created element

**Return type**

*Element*

**export**(*filename='element.zip'*)

Export this element.

Usage:

```
engine = Engine('myfirewall')
extask = engine.export(filename='fooexport.zip')
while not extask.done():
    extask.wait(3)
```

(continues on next page)

(continued from previous page)

```
print("Finished download task: %s" % extask.message())
print("File downloaded to: %s" % extask.filename)
```

**Parameters**

**filename** (*str*) – filename to store exported element

**Raises**

**TaskRunFailed** – invalid permissions, invalid directory, or this element is a system element and cannot be exported.

**Returns**

DownloadTask

---

**Note:** It is not possible to export system elements

---

**classmethod** `get(name, raise_exc=True)`

Get the element by name. Does an exact match by element type.

**Parameters**

- **name** (*str*) – name of element
- **raise\_exc** (*bool*) – optionally disable exception.

**Raises**

**ElementNotFound** – if element does not exist

**Return type**

*Element*

**classmethod** `get_or_create(filter_key=None, with_status=False, **kwargs)`

Convenience method to retrieve an Element or create if it does not exist. If an element does not have a `create` classmethod, then it is considered read-only and the request will be redirected to `get()`. Any keyword arguments passed except the optional `filter_key` will be used in a `create()` call. If `filter_key` is provided, this should define an attribute and value to use for an exact match on the element. Valid attributes are ones required on the elements `create` method or can be viewed by the elements class docs. If no `filter_key` is provided, the name field will be used to find the element.

```
>>> Network.get_or_create(
    filter_key={'ipv4_network': '123.123.123.0/24'},
    name='mynetwork',
    ipv4_network='123.123.123.0/24')
Network(name=mynetwork)
```

The kwargs should be used to satisfy the elements `create` classmethod parameters to create in the event it cannot be found.

**Parameters**

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements `create` method.
- **with\_status** (*bool*) – if set to True, a tuple is returned with (Element, created), where the second tuple item indicates if the element has been created or not.

**Raises**

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

**Returns**

element instance by type

**Return type**

*Element*

**property history**

New in version 0.5.7: Requires SMC version >= 6.3.2

Obtain the history of this element. This will not chronicle every modification made over time, but instead a current snapshot with historical information such as when the element was created, by whom, when it was last modified and it's current state.

**Raises**

*ResourceNotFound* – If not running SMC version >= 6.3.2

**Return type**

*History*

**is\_locked()**

Locked flag for element

**lock(reason\_for=None)**

Locks this element with an optional reason.

**Raises**

*ResourceNotFound* – If not running on supported SMC version

**property name**

Name of element

**property referenced\_by**

Show all references for this element. A reference means that this element is being used, for example, in a policy rule, as a member of a group, etc.

**Returns**

list referenced elements

**Return type**

*list(Element)*

**rename(name)**

Rename this element.

**Parameters**

**name** (*str*) – new name of element

**Raises**

*UpdateElementFailed* – update failed with reason

**Returns**

None

**unlock()**

Unlocks this element.

**Raises***ResourceNotFound* – If not running on supported SMC version**classmethod** `update_or_create`(*filter\_key=None, with\_status=False, \*\*kwargs*)

Update or create the element. If the element exists, update it using the kwargs provided if the provided kwargs after resolving differences from existing values. When comparing values, strings and ints are compared directly. If a list is provided and is a list of strings, it will be compared and updated if different. If the list contains unhashable elements, it is skipped. To handle complex comparisons, override this method on the subclass and process the comparison separately. If an element does not have a *create* classmethod, then it is considered read-only and the request will be redirected to *get()*. Provide a *filter\_key* dict key/value if you want to match the element by a specific attribute and value. If no *filter\_key* is provided, the name field will be used to find the element.

```
>>> host = Host('kali')
>>> print(host.address)
12.12.12.12
>>> host = Host.update_or_create(name='kali', address='10.10.10.10')
>>> print(host, host.address)
Host(name=kali) 10.10.10.10
```

**Parameters**

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements *create* method.
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

**Raises**

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

**Returns**

element instance by type

**Return type***Element***class** `smc.base.model.SubElement`(*\*\*meta*)Bases: *ElementBase*

SubElement is the base class for elements that do not have direct entry points in the SMC and instead are obtained through a reference. They are not ‘loaded’ directly as are classes that inherit from *Element*.

**class** `smc.base.model.UserElement`(*name, \*\*meta*)Bases: *Element*

User element mixin for LDAP of Internal Domains.

**property** `name`

Name of element

**property** `unique_id`

Fully qualified unique DN for this entry

**Return type***str*

```
class smc.core.resource.History(creation_time=None, creator=None, creator_name=None,  
                                is_locked=None, is_obsolete=None, is_trashed=None,  
                                last_modification_time=None, modifier=None, modifier_name=None,  
                                follower=None, in_progress=None, last_message=None, progress=None,  
                                success=None, waiting_inputs=None)
```

History description of this element. This will provide basic information about the element such as when it was created, last modified along with the accounts making the modifications.

**Variables**

- **is\_locked** (*bool*) – is this record currently locked
- **is\_obsolete** (*bool*) – is this record obsoleted
- **is\_trashed** (*bool*) – is the record in the trash bin

**property created\_by**

The account that created this element. Returned as an Element.

**Return type***Element***property last\_modified**

When the element was last modified as a datetime object

**Return type***datetime***property modified\_by**

The account that last modified this element.

**Return type***Element***property when\_created**

When the element was created as a datetime object

**Return type***datetime*

## 14.3 Administration

### 14.3.1 Access Rights

Access Rights provide the ability to create administrative accounts and assign or create specific access control lists and roles to these accounts.

### 14.3.1.1 AccessControlList

**class** `smc.administration.access_rights.AccessControlList`(*name=None, \*\*meta*)

Bases: *Element*

An ACL is assigned to an AdminUser to grant limited access permissions to either Engines, Policies or Domains. The access control list will have ‘granted elements’ that represent the elements that apply to this permission. The SMC provides default ACL’s that can be used or new ones can be created. Find all available ACL’s:

```
>>> AccessControlList.objects.all()
```

**add\_permission**(*elements*)

Add permission/s to this ACL. By default this change is committed after the method is called.

**Parameters**

**elements** (*list*(*str*, *Element*)) – Elements to grant access to. Can be engines, policies, or other ACLs

**Raises**

*UpdateElementFailed* – Failed updating permissions

**Returns**

None

**classmethod** **create**(*name*, *granted\_element=None*)

Create a new ACL

**Parameters**

- **name** (*str*) – Name of ACL
- **granted\_elements** (*list*(*str*, *Element*)) – Elements to grant access to. Can be engines, policies or other acl’s.

**Raises**

*CreateElementFailed* – failed creating ACL

**Returns**

instance with meta

**Return type**

*AccessControlList*

**property** **permissions**

Elements associated to this permission. Granted elements can be Engines, Policies or other Access Control Lists.

**Returns**

Element class deriving from *smc.base.model.Element*

**remove\_permission**(*elements*)

Remove permission/s to this ACL. Change is committed at end of method call.

**Parameters**

**elements** (*list*(*str*, *Element*)) – list of element/s to remove

**Raises**

*UpdateElementFailed* – Failed modifying permissions

**Returns**

None

### 14.3.1.2 Administrators

User module to hold accounts related to users (admin or local) in the SMC

You can create an Admin User, enable superuser, enable/disable the account, assign local access to engines, and change the account password for SMC or engine access.

It is possible to fully provision an Admin User with specific permissions and roles and initial password.

Create the admin:

```
admin = AdminUser.create(name='auditor', superuser=False)
```

---

**Note:** If the Admin User should have unrestricted access, set `superuser=True` and skip the below sections related to adding permissions and roles.

---

Permissions relate to elements that the user will have access to (Policies, Engines or AccessControlLists) and the domain where the privileges apply (default is 'Shared Domain').

Create a permission using the default domain of Shared, granting access to a specific engine and firewall policy:

```
permission = Permission.create(
    elements=[Engine('vm'), FirewallPolicy('VM Policy')],
    role=Role('Viewer'))
```

Create a second permission granting access to all firewalls in the domain 'mydomain':

```
domain_perm = Permission.create(
    elements=[AccessControlList('ALL Firewalls')],
    role=Role('Owner'),
    domain=AdminDomain('mydomain'))
```

Add the permissions to the Admin User:

```
admin.add_permission([permission, domain_perm])
```

Set an initial password for the Admin User:

```
admin.change_password('Newpassword1')
```

---

**Note:** Roles are used to define what granular controls will be available to the assigned user, such as read/read write/all. AccessControlLists encapsulate elements into a single container for re-use.

---

**See also:**

[`smc.administration.role.Role`](#) and [`smc.administration.access\_rights.AccessControlList`](#) for more information.

**class** `smc.elements.user.AdminUser`(*name=None, \*\*meta*)

Bases: [`UserMixin`](#), [`Element`](#)

Represents an Administrator account on the SMC Use the constructor to create the user.

Create an Admin:

```
>>> AdminUser.create(name='admin', superuser=True)
AdminUser(name=admin)
```

If modifications are required after you can access the admin and make changes:

```
admin = AdminUser('admin')
admin.change_password('mynewpassword1')
admin.enable_disable()
```

Attributes available:

#### Variables

- **allow\_sudo** (*bool*) – is this account allowed to sudo on an engine.
- **local\_admin** (*bool*) – is the admin a local admin
- **superuser** (*bool*) – is this account a superuser for SMC

#### **change\_engine\_password**(*password*)

Change Engine password for engines on allowed list.

#### Parameters

**password** (*str*) – password for engine level

#### Raises

**ModificationFailed** – failed setting password on engine

#### Returns

None

**classmethod create**(*name*, *local\_admin=False*, *allow\_sudo=False*, *superuser=False*, *enabled=True*, *engine\_target=None*, *can\_use\_api=True*, *console\_superuser=False*, *allowed\_to\_login\_in\_shared=True*, *auth\_method=None*, *comment=None*, *permissions=None*, *ldap\_user\_href=None*, *ldap\_group\_href=None*)

Create an admin user account.

New in version 0.6.2: Added *can\_use\_api*, *console\_superuser*, and *allowed\_to\_login\_in\_shared*. Requires SMC >= SMC 6.4

#### Parameters

- **name** (*str*) – name of account
- **local\_admin** (*bool*) – is a local admin only
- **allow\_sudo** (*bool*) – allow sudo on engines
- **can\_use\_api** (*bool*) – can log in to SMC API
- **console\_superuser** (*bool*) – can this user sudo via SSH/console
- **allowed\_to\_login\_in\_shared** (*bool*) – can this user log in to the shared domain
- **superuser** (*bool*) – is a super administrator
- **auth\_method** – authentication method
- **enabled** (*bool*) – is account enabled
- **engine\_target** (*list*) – engine to allow remote access to
- **comment** – object comment

:param permissions object in case SMC admin is not superuser :param ldap\_user\_href External user href to link as SMC admin :param ldap\_group\_href External user href to link as SMC admin :raises CreateElementFailed: failure creating element with reason :return: instance with meta :rtype: AdminUser

**property enabled**

Read only enabled status

**Return type**

`bool`

**property password\_meta\_data**

Provides creation\_date and expiration\_date of the password for AdminUser, ApiClient and WebPortalAdminUser. :return: PasswordMetaData : PasswordMetaData contains creation\_date and expiration\_date.

```
class smc.elements.user.ApiClient(name=None, **meta)
```

Bases: `UserMixin`, `Element`

Represents an API Client

```
classmethod create(name, enabled=True, superuser=True)
```

Create a new API Client. Once client is created, you can create a new password by:

```
>>> client = ApiClient.create('myclient')
>>> print(client)
ApiClient(name=myclient)
>>> client.change_password('mynewpassword')
```

**Parameters**

- **name** (`str`) – name of client
- **enabled** (`bool`) – enable client
- **superuser** (`bool`) – is superuser account

**Raises**

`CreateElementFailed` – failure creating element with reason

**Returns**

instance with meta

**Return type**

`ApiClient`

```
class smc.elements.user.PasswordMetaData(value)
```

Bases: `NestedDict`

Represents the password meta-data for AdminUser, ApiClient and WebPortalAdminUser. it provides creation\_date and expiration\_date of the password for AdminUser, ApiClient and WebPortalAdminUser

```
class smc.elements.user.UserMixin
```

Bases: `object`

User Mixin class providing common operations for Admin Users and API Clients.

```
add_permission(permission)
```

Add a permission to this Admin User. A role defines permissions that can be enabled or disabled. Elements define the target for permission operations and can be either Access Control Lists, Engines or Policy elements. Domain specifies where the access is granted. The Shared Domain is default unless specific domain provided. Change is committed at end of method call.

**Parameters****permission** (*list*(*Permission*)) – permission/s to add to admin user**Raises***UpdateElementFailed* – failed updating admin user**Returns**

None

**change\_password**(*password*)

Change user password. Change is committed immediately.

**Parameters****password** (*str*) – new password**Returns**

None

**enable\_disable**()

Toggle enable and disable of administrator account. Change is committed immediately.

**Raises***UpdateElementFailed* – failed with reason**Returns**

None

**generate\_password**()

Generate a random password for this user.

**Returns**

random password

**Return type***str***property permissions**

Return each permission role mapping for this Admin User. A permission role will have 3 fields:

- Domain
- Role (Viewer, Operator, etc)
- Elements (Engines, Policies, or ACLs)

**Returns**

permissions as list

**Return type***list*(*Permission*)**class** smc.elements.user.**WebPortalAdminUser**(*name=None, \*\*meta*)Bases: *UserMixin*, *Element***This represents a Web Portal User. It is an element that defines the details of a single person**

that is allowed to log on to the Web Portal, the Browser-based service that allows users to view logs, Policy Snapshots, and reports

Create a Web Portal Admin User:

```
>>> WebPortalAdminUser.create(name='admin')
```

If modifications are required after you can access the admin and make changes:

```
admin = WebPortalAdminUser('admin')
admin.change_password('mynewpassword1')
admin.enable_disable()
```

```
classmethod create(name, enabled=True, granted_engine=None, console_superuser=False,
                    log_service_enabled=True, policy_service_enabled=True,
                    report_service_enabled=True, show_inspection_policy=True,
                    show_main_policy=True, show_only_ip_addresses=True, show_sub_policy=True,
                    show_template_policy=False, show_upload_comment=True,
                    show_upload_history=True, granted_template_policy=None,
                    granted_sub_policy=None, granted_report_design=None, filter_tag=None,
                    comment=None)
```

Create a web portal admin user account.

New in version 0.6.2: Added can\_use\_api, console\_superuser, and allowed\_to\_login\_in\_shared. Requires SMC >= SMC 6.4

#### Parameters

- **name** (*str*) – name of account
- **enabled** (*bool*) – is account enabled
- **granted\_engine** (*list*) – The list of Granted Engines
- **console\_superuser** (*bool*) – can this user sudo via SSH/console.
- **log\_service\_enabled** (*bool*) – check if the log service enabled?
- **policy\_service\_enabled** (*bool*) – check if the policy service enabled.
- **report\_service\_enabled** (*bool*) – Is the report service enabled?
- **show\_inspection\_policy** (*bool*) – Should we display the inspection policy?
- **show\_main\_policy** (*bool*) – Should we display the main policy?
- **show\_only\_ip\_addresses** (*bool*) – Should we display only the IP Addresses of elements?
- **show\_sub\_policy** (*bool*) – Should we display the sub policy?
- **show\_template\_policy** (*bool*) – Should we display the template policy?
- **show\_upload\_comment** (*bool*) – Should we display the upload comment?
- **show\_upload\_history** (*bool*) – Should we display the upload history?
- **granted\_template\_policy** (*list*) – The list of Granted Template Policies. null value means ANY.
- **granted\_sub\_policy** (*list*) – The list of Granted Sub Policies. null value means ANY
- **granted\_report\_design** (*list*) – The list of Granted Report Designs. null value means ANY.
- **filter\_tag** (*list*) – The list of Filter expression tags for the log browsing. null value means ANY.
- **comment** (*str*) – comment,

#### Raises

**CreateElementFailed** – failure creating element with reason

**Returns**

instance with meta

**Return type***WebPortalAdminUser***14.3.1.3 Permission**

```
class smc.administration.access_rights.Permission(granted_elements=None, role_ref=None,  
                                                granted_domain_ref=None)
```

Permissions are added to admin users that do not have super user access rights. An Admin User can also have multiple permissions. There are three primary fields associated with a permission:

- Domain to grant access
- Elements to grant access to (Engines, Policies or AccessControlLists)
- Role

A permission might be used to grant read-only access to specific policies or firewalls (read-only vs read write). It can also be specific to the Admin Domain.

**See also:***smc.elements.user*

```
classmethod create(elements, role, domain=None)
```

Create a permission.

**Parameters**

- **granted\_elements** (*list(str, Element)*) – Elements for this permission. Can be engines, policies or ACLs
- **role** (*str, Role*) – role for this permission
- **domain** (*str, Element*) – domain to apply (default: Shared Domain)

**Return type***Permission***property domain**

Domain this permission applies to. Shared Domain if unspecified.

**Return type***AdminDomain***property granted\_elements**

List of elements this permission has rights to. Elements will be of type Engine, Policy or ACLs

**Return type***list(Element)***property role**

Specific Role assigned to this permission. A role is what allows read/write access to specific operations on the granted elements

**Return type***Role*

#### 14.3.1.4 Roles

Administrator Role elements specify a restricted set of permissions that include the right to create, edit, and delete elements.

Each administrator can have several different Administrator Roles applied to different sets of elements. There are some default Administrator Roles, but if you want to customize the permissions in any way, you must create custom Administrator Role elements.

Create a new role is done by using the create classmethod. By default the role will not have any permissions set:

```
>>> from smc.administration.role import Role
>>> role = Role.create(name='mynewrole')
```

A role has many attributes (mostly boolean) that can be enabled, therefore the simplest way to create a new role is to duplicate an existing role.

```
>>> list(Role.objects.all())
[Role(name=myeditor), Role(name=Logs Viewer), Role(name=Reports Manager),
↪ Role(name=Owner),
  Role(name=Viewer), Role(name=Operator), Role(name=Monitor), Role(name=Editor),
  Role(name=Superuser)]
...
```

Duplicate an existing role to simplify making modifications on permissions:

```
>>> role = Role('Editor')
>>> role.duplicate('customeditor')
Role(name=customeditor)
```

To enable or disable role permissions, use the enable/disable option after retrieving the Role resource.

Available and current permission settings can be found by calling permissions attribute:

```
>>> role = Role('newrole')
>>> role.permissions
[{'alert_mgmt': False}, {'send_advanced_commands': False}, {'license_mgmt': False},
{'element_edit': False}, {'view_edit_report': False}, {'view_system_alerts': False},
{'view_logs': False}, {'vpn_mgmt': False}, {'log_pruning_mgmt': False},
{'updates_and_upgrades_mgmt': False}, {'auth_server_user_mgmt': False}, {'view_audit':
↪ False},
{'element_delete': False}, {'element_create': False}, {'upload_policy': False},
{'send_commands': False}, {'backup_mgmt': False}, {'element_view_content': True},
{'log_mgmt': False}, {'bookmark_manage': True}, {'admin_mgmt': False}, {'name': 'newrole'
↪ },
{'overview_manage': True}, {'internal_user_mgmt': False}, {'refresh_policy': False}]
```

Then enable specific roles by specifying the keys to enable:

```
>>> role.enable(['element_create', 'upload_policy'])
```

Also disable specific roles:

```
>>> role.disable(['element_create', 'upload_policy'])
```

Once modification is complete, call update on the role:

```
>>> role.update()
'http://172.18.1.151:8082/6.4/elements/role/10'
```

**class** `smc.administration.role.Role(name=None, **meta)`

Bases: [Element](#)

Role class represents granular access control rights that can be applied to specific elements (Engines, Policies or Access Control Lists).

**classmethod** `create(name, comment=None)`

Create a new role. The role will not have any permissions by default so it will be required to call `enable` on the role after creation.

**Parameters**

- **name** ([str](#)) – name of role
- **comment** ([str](#)) – comment for role

**Raises**

[CreateElementFailed](#) – failed to create role

**Return type**

[Role](#)

**disable**([values](#))

Disable specific permissions on this role. Use [permissions](#) to view valid permission settings and current value/s. Change is committed immediately.

**Parameters**

**values** ([list](#)) – list of values by allowed types

**Returns**

None

**enable**([values](#))

Enable specific permissions on this role. Use [permissions](#) to view valid permission settings and current value/s. Change is committed immediately.

**Parameters**

**values** ([list](#)) – list of values by allowed types

**Returns**

None

**property permissions**

Return valid permissions and setting for this role. Permissions are returned as a list of dict items, {permission: state}. State for the permission is either True or False. Use [enable\(\)](#) and [disable\(\)](#) to toggle role settings.

**Returns**

list of permission settings

**Return type**

[list\(dict\)](#)

## 14.3.2 Alert Configuration

### 14.3.2.1 AlertChain

**class** smc.elements.alerts.AlertChain(name=None, \*\*meta)

This represents an Alert Chain.

**classmethod** create(name, final\_action=None, alert\_chain\_ref=None, comment=None)

Create the custom alert :param str name: name of alert chain :param str comment: optional comment :param str final\_action: optional final\_action

**possible values:**

1)none: stop policy processing without acknowledging. 2)acknowledge: stop policy processing and acknowledge. 3)redirect: redirect to another alert chain. 4)return: return to the next policy rule.

**Parameters**

**alert\_chain\_ref** (*obj*) – The redirect alert chain. object of AlertChain.

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta of alert chain

**Return type**

*AlertChain*

### 14.3.2.2 AlertChainRule

**class** smc.elements.alerts.AlertChainRule(\*\*meta)

This represents a Alert Chain Rule for Alert Chain Policy.

**static** create(self, name, alert\_channel=None, destination=None, delay=0, admin\_name=[], amount=None, notify\_first\_block=0, period=0, comment=None)

**Parameters**

- **self** (*object*) – object of AlertChain.
- **name** (*str*) – name of alert chain rule.
- **alert\_channel** (*str*) – The alert channel, default is Delay channel. Valid values are below smtp: SMTP channel. sms: SMS channel. snmp: SNMP channel. custom\_script: Custom script channel. delay: Delay channel. user\_notification: User notification channel.
- **destination** (*str*) – destination address
- **delay** (*int*) – The delay before the next notification, in minutes.
- **admin\_name** (*list*) – List of admin users. Used in the case of User notification channel.
- **amount** (*int*) – The maximum number of notifications to be sent before activating

moderation. :param int notify\_first\_block: Indicates whether we shall notify the first blocked notification upon moderation activation. :param int period: The period during which notifications are tracked before activating moderation. period need to be mentioned in minutes. :param str comment: descript of element. :raises CreateElementFailed: failed creating element with reason :return: instance with meta of alert chain rule :rtype: AlertChainRule

### 14.3.2.3 AlertPolicy

**class** `smc.elements.alerts.AlertPolicy(name=None, **meta)`

This represents an Alert Policy.

**add\_alert\_rule**(*name*, *alert\_chain\_ref*=None, *match\_sender\_ref*=[], *alert\_and\_situation\_ref*=[], *min\_severity*=1, *max\_severity*=10, *rule\_validity\_times*=[], *comment*=None)

creation of the element of type alert\_rule. :param object self: object of AlertPolicy. :param str name: name of alert rule. :param str(AlertChain) alert\_chain\_ref: The Alert Chain. :param str match\_sender\_ref: The senders. If empty, it is considered as ANY. :param list alert\_and\_situation\_ref: The alerts and situations. If empty, it is considered

as ANY.

#### Parameters

- **min\_severity** (*int*) – The minimum value for the severity (value between 1 and 10)
- **max\_severity** (*int*) – The maximum value for the severity (value between 1 and 10)
- **rule\_validity\_times** (*list*) – The rule's validity to a specific time period. During the specified time period, the rule matches. Outside the specified time period, the rule does not match and the matching continues to the next rule.
- **comment** (*str*) – descript of element.

#### Raises

*CreateElementFailed* – failed creating element with reason

#### Returns

instance with meta of alert rule

#### Return type

*AlertRule*

**classmethod** **create**(*name*, *comment*=None)

Create the custom alert :param str name: name of alert policy :param str comment: optional comment :raises CreateElementFailed: failed creating element with reason :return: instance with meta of alert policy :rtype: AlertPolicy

### 14.3.2.4 AlertRule

**class** `smc.elements.alerts.AlertRule(**meta)`

This represents Alert Rule for Alert Policy.

**static** **create**(*self*, *name*, *alert\_chain\_ref*=None, *match\_sender\_ref*=[], *alert\_and\_situation\_ref*=[], *min\_severity*=1, *max\_severity*=10, *rule\_validity\_times*=[], *comment*=None)

creation of the element of type alert\_rule. :param object self: object of AlertPolicy. :param str name: name of alert rule. :param str(AlertChain) alert\_chain\_ref: The Alert Chain. :param list match\_sender\_ref: The senders. If empty, it is considered as ANY. :param list alert\_and\_situation\_ref: The alerts and situations. If empty, it is considered

as ANY.

#### Parameters

- **min\_severity** (*int*) – The minimum value for the severity (value between 1 and 10):Matches the rule to only Situations with the specified Severity value(s). For example, if your rule is general and matches a wide range of Situations, you may want to create two similar rules: one for less severe Situations and one for more Severe situations. Useful in rules that contain Tags in the Situation cell.
- **max\_severity** (*int*) – The maximum value for the severity (value between 1 and 10):Matches the rule to only Situations with the specified Severity value(s). For example if your rule is general and matches a wide range of Situations, you may want to create two similar rules: one for less severe Situations and one for more Severe situations. Useful in rules that contain Tags in the Situation cell.
- **rule\_validity\_times** (*list*) – The rule’s validity to a specific time period. During the specified time period, the rule matches. Outside the specified time period, the rule does not match and the matching continues to the next rule.
- **comment** (*str*) – descript of element.

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta of alert rule

**Return type**

*AlertRule*

## 14.3.3 Certificates

### 14.3.3.1 TLSCommon

TLS Common module provides mixin methods that are common to certificate handling in SMC. Importing certificates and private keys can be done by providing a file where the certificates/keys are stored, or providing in string format.

**class** smc.administration.certificates.tls\_common.**ImportExportCertificate**

Mixin to provide certificate import and export methods to relevant classes.

**export\_certificate**(*filename=None*)

Export the certificate. Returned certificate will be in string format. If filename is provided, the certificate will also be saved to the file specified.

**Raises**

*CertificateExportError* – error exporting certificate

**Return type**

*str* or *None*

**import\_certificate**(*certificate*)

Import a valid certificate. Certificate can be either a file path or a string of the certificate. If string certificate, it must include the —BEGIN CERTIFICATE— string.

**Parameters**

**certificate\_file** (*str*) – fully qualified path to certificate file

**Raises**

- *CertificateImportError* – failure to import cert with reason
- *IOError* – file not found, permissions, etc.

**Returns**

None

**class** smc.administration.certificates.tls\_common.**ImportExportIntermediate**

Mixin to provide import and export capabilities for intermediate certificates

**export\_intermediate\_certificate**(filename=None)

Export the intermediate certificate. Returned certificate will be in string format. If filename is provided, the certificate will also be saved to the file specified.

**Raises**

**CertificateExportError** – error exporting certificate, can occur if no intermediate certificate is available.

**Return type**

str or None

**import\_intermediate\_certificate**(certificate)

Import a valid certificate. Certificate can be either a file path or a string of the certificate. If string certificate, it must include the —BEGIN CERTIFICATE— string.

**Parameters**

**certificate** (str) – fully qualified path or string

**Raises**

- **CertificateImportError** – failure to import cert with reason
- **IOError** – file not found, permissions, etc.

**Returns**

None

**class** smc.administration.certificates.tls\_common.**ImportPrivateKey**

Mixin to provide import capabilities to relevant classes that require private keys.

**import\_private\_key**(private\_key)

Import a private key. The private key can be a path to a file or the key in string format. If in string format, the key must start with —BEGIN. Key types supported are PRIVATE RSA KEY and PRIVATE KEY.

**Parameters**

**private\_key** (str) – fully qualified path to private key file

**Raises**

- **CertificateImportError** – failure to import cert with reason
- **IOError** – file not found, permissions, etc.

**Returns**

None

### 14.3.3.2 TLSServerCredential

TLS module provides interactions related to importing TLS Server Credentials for inbound SSL decryption, as well as client protection certificates used for outbound decryption.

To properly decrypt inbound TLS connections, you must provide the Stonesoft FW with a valid certificate and private key. Within SMC these certificate types are known as TLS Server Credentials.

Once you have imported these certificates, you must then assign them to the relevant engines that will perform the decryption services. Lastly you will need a rule that enables HTTPS with decryption.

First start by importing the TLS Server Credential class:

```
>>> from smc.administration.certificates.tls import TLSServerCredential
```

If you want to create a TLS Server Credential in steps, the process is as follows:

```
tls = TLSServerCredential.create(name)      # Create the certificate element
tls.import_certificate(certificate) # Import the certificate
tls.import_private_key(private_key) # Import the private key
tls.import_intermediate_certificate(intermediate) # Import intermediate certificate,
↳ (optional)
```

Otherwise, use helper methods that allow you to do this in a single step.

For example, creating the TLS credential from certificate files:

```
>>> tls = TLSServerCredential.import_signed(
    name='server.test.local',
    certificate='/path/to/server.crt',
    private_key='/path/to/server.key',
    intermediate=None) # <-- You can also include intermediate certificates
>>> tls
TLSServerCredential(name=server.test.local)
```

---

**Note:** Certificate, private key and intermediate certificates can also be specified in raw string format and must start with the BEGIN CERTIFICATE, etc common syntax.

---

You can also import certificates from a certificate chain file. When doing so, the certificates are expected to be in the order: server certificate, intermediate/s, root certificate. You can optionally also add the private key to the chain file or provide it separately:

```
tls = TLSServerCredential.import_from_chain(
    name='fromchain', certificate_file='/path/cert.chain',
    private_key='/path/priv.key')
```

---

**Note:** If multiple intermediate certificates are added, only the first one is imported into the TLS Server Credential. In addition, the root certificate is ignored and should be imported using `TLSCertificateAuthority.create()`.

---

It is also possible to create self signed certificates using the SMC CA:

```
>>> tls = TLSServerCredential.create_self_signed(
    name='server.test.local', common_name='CN=server.test.local')
```

(continues on next page)

(continued from previous page)

```
>>> tls
TLSServerCredential(name=server.test.local)
```

If you would rather use the SMC to generate the CSR and have the request signed by an external CA you can call `TLSServerCredential.create_csr()` and export the request:

```
>>> tls = TLSServerCredential.create_csr(name='public.test.local',
                                         common_name='CN=public.test.local')
>>> tls.certificate_export()
'-----BEGIN CERTIFICATE REQUEST-----
MIIEXTCCAkcCAQAwHDEaMBGGA1UEAwRcHVibGljLnRlc3QubG9jYWwwggIiMA0G
CSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQC68xcXrWQ5E25nkTfmgmPQiWVPwf
....
-----END CERTIFICATE REQUEST-----'
```

Optionally export the request to a local file:

```
>>> tls = TLSServerCredential.create_csr(
    name='public2.test.local', common_name='CN=public2.test.local')
>>> tls.certificate_export(filename='public2.test.local.csr')
```

If you use an external CA for signing your certificates, you can also import that as a TLS Certificate Authority. The link between the certificates and root CA will be made automatically:

```
TLSCertificateAuthority.create(
    name='myrootca',
    certificate='/path/to/cert/or/string')
```

Once you have the TLS Server Credentials within SMC, you can then assign them to the relevant engines:

```
>>> from smc.core.engine import Engine
>>> from smc.administration.certificates import TLSServerCredential
>>> engine = Engine('myfirewall')
>>> engine.tls_inspection.add_tls_credential([TLSServerCredential('public.test.local'),
                                              TLSServerCredential('server.test.local')])
>>> engine.tls_inspection.server_credentials
[TLSServerCredential(name=public.test.local), TLSServerCredential(name=server.test.
↪local)]
```

---

**Note:** It is possible to import and export certificates from the SMC, but it is not possible to export private keys.

---

**class** `smc.administration.certificates.tls.TLSServerCredential` (*name=None, \*\*meta*)

Bases: `ImportExportIntermediate`, `ImportPrivateKey`, `ImportExportCertificate`, `Element`

If you want to inspect TLS traffic for which an internal server is the destination, you must create a TLS Credentials element to store the private key and certificate of the server.

The private key and certificate allow the firewall to decrypt TLS traffic for which the internal server is the destination so that it can be inspected.

After a `TLSServerCredential` has been created, you must apply this to the engine performing decryption and create the requisite policy rule that uses SSL decryption.

**Variables**

**certificate\_state** (*str*) – State of the certificate. Available states are ‘request’ and ‘certificate’. If the state is ‘request’, this represents a CSR and needs to be signed.

**classmethod create**(*name*)

Create an empty certificate. This will only create the element in the SMC and will then require that you import the server certificate, intermediate (optional) and private key.

**See also:**

[\*import\\_signed\(\)\*](#) and [\*import\\_from\\_chain\(\)\*](#).

**Raises**

[\*CreateElementFailed\*](#) – failed creating element

**Return type**

[\*TLSServerCredential\*](#)

**classmethod create\_csr**(\*args, \*\*kwargs)

Create a certificate signing request.

**Parameters**

- **name** (*str*) – name of TLS Server Credential
- **rcommon\_name** (*str*) – common name for certificate. An example would be: “CN=CommonName,O=Organization,OU=Unit,C=FR,ST=PACA,L=Nice”. At minimum, a “CN” is required.
- **public\_key\_algorithm** (*str*) – public key type to use. Valid values `rsa`, `dsa`, `ecdsa`.
- **signature\_algorithm** (*str*) – signature algorithm. Valid values `dsa_sha_1`, `dsa_sha_224`, `dsa_sha_256`, `rsa_md5`, `rsa_sha_1`, `rsa_sha_256`, `rsa_sha_384`, `rsa_sha_512`, `ecdsa_sha_1`, `ecdsa_sha_256`, `ecdsa_sha_384`, `ecdsa_sha_512`. (Default: `rsa_sha_512`)
- **key\_length** (*int*) – length of key. Key length depends on the key type. For example, RSA keys can be 1024, 2048, 3072, 4096. See SMC documentation for more details.

**Raises**

[\*CreateElementFailed\*](#) – failed to create CSR

**Return type**

[\*TLSServerCredential\*](#)

**classmethod create\_self\_signed**(*name*, *common\_name*, *public\_key\_algorithm*='rsa',  
*signature\_algorithm*='rsa\_sha\_512', *key\_length*=4096)

Create a self signed certificate. This is a convenience method that first calls [\*create\\_csr\(\)\*](#), then calls [\*self\\_sign\(\)\*](#) on the returned [\*TLSServerCredential\*](#) object.

**Parameters**

- **name** (*str*) – name of TLS Server Credential
- **rcommon\_name** (*str*) – common name for certificate. An example would be: “CN=CommonName,O=Organization,OU=Unit,C=FR,ST=PACA,L=Nice”. At minimum, a “CN” is required.
- **public\_key\_algorithm** (*str*) – public key type to use. Valid values `rsa`, `dsa`, `ecdsa`.
- **signature\_algorithm** (*str*) – signature algorithm. Valid values `dsa_sha_1`, `dsa_sha_224`, `dsa_sha_256`, `rsa_md5`, `rsa_sha_1`, `rsa_sha_256`, `rsa_sha_384`, `rsa_sha_512`, `ecdsa_sha_1`, `ecdsa_sha_256`, `ecdsa_sha_384`, `ecdsa_sha_512`. (Default: `rsa_sha_512`)

- **key\_length** (*int*) – length of key. Key length depends on the key type. For example, RSA keys can be 1024, 2048, 3072, 4096. See SMC documentation for more details.

#### Raises

- **CreateElementFailed** – failed to create CSR
- **ActionCommandFailed** – Failure to self sign the certificate

#### Return type

*TLSServerCredential*

**classmethod** **import\_from\_chain**(*name, certificate\_file, private\_key=None*)

Import the server certificate, intermediate and optionally private key from a certificate chain file. The expected format of the chain file follows RFC 4346. In short, the server certificate should come first, followed by any intermediate certificates, optionally followed by the root trusted authority. The private key can be anywhere in this order. See <https://tools.ietf.org/html/rfc4346#section-7.4.2>.

---

**Note:** There is no validation done on the certificates, therefore the order is assumed to be true. In addition, the root certificate will not be imported and should be separately imported as a trusted root CA using *create*

---

If the certificate chain file has only two entries, it is assumed to be the server certificate and root certificate (no intermediates). In which case only the certificate is imported. If the chain file has 3 or more entries (all certificates), it will import the first as the server certificate, 2nd as the intermediate and ignore the root cert.

You can optionally provide a separate location for a private key file if this is not within the chain file contents.

**Warning:** A private key is required to create a valid TLS Server Credential.

#### Parameters

- **name** (*str*) – name of TLS Server Credential
- **certificate\_file** (*str*) – fully qualified path to chain file or file object
- **private\_key** (*str*) – fully qualified path to chain file or file object

#### Raises

- **IOError** – error occurred reading or finding specified file
- **ValueError** – Format issues with chain file or empty

#### Return type

*TLSServerCredential*

**classmethod** **import\_signed**(*name, certificate, private\_key, intermediate=None*)

Import a signed certificate and private key to SMC, and optionally an intermediate certificate. The certificate and the associated private key must be compatible with OpenSSL and be in PEM format. The certificate and private key can be imported as a raw string, file path or file object. If importing as a string, be sure the string has carriage returns after each line and the final *END CERTIFICATE* line.

Import a certificate and private key:

```
>>> tls = TLSServerCredential.import_signed(
    name='server2.test.local',
    certificate='mydir/server.crt',
    private_key='mydir/server.key')
>>> tls
TLSServerCredential(name=server2.test.local)
```

#### Parameters

- **name** (*str*) – name of TLSServerCredential
- **certificate** (*str*) – fully qualified to the certificate file, string or file object
- **private\_key** (*str*) – fully qualified to the private key file, string or file object
- **intermediate** (*str*) – fully qualified to the intermediate file, string or file object

#### Raises

- *CertificateImportError* – failure during import
- *CreateElementFailed* – failed to create credential
- *IOError* – failure to find certificate files specified

#### Return type

*TLSServerCredential*

#### **self\_sign()**

Self sign the certificate in ‘request’ state.

#### Raises

*ActionCommandFailed* – failed to sign with reason

#### **property valid\_from**

New in version 0.6.0: Requires SMC version >= 6.3.4

The valid from datetime for this TLS Server Credential.

#### Return type

*datetime.datetime*

#### **property valid\_to**

New in version 0.6.0: Requires SMC version >= 6.3.4

The expiration (valid to) datetime for this TLS Server Credential.

#### Return type

*datetime.datetime*

### 14.3.3.3 TLSProfile

**class** `smc.administration.certificates.tls.TLSProfile(name=None, **meta)`

Bases: *Element*

New in version 0.6.2: Requires SMC >= 6.4

Represents a TLS Profile. Contains common parameters for establishing TLS based connections. TLS Profiles are used in various configuration areas such as SSL VPN portal and Active Directory (when using TLS) connections.

**classmethod** `create(name, tls_version, use_only_subject_alt_name=False, accept_wildcard=False, check_revocation=True, tls_cryptography_suites=None, crl_delay=0, ocsf_delay=0, ignore_network_issues=False, tls_trusted_ca_ref=None, comment=None)`

Create a TLS Profile. By default the SMC will have a default NIST TLS Profile but it is also possible to create a custom profile to provide special TLS handling.

#### Parameters

- **name** (*str*) – name of TLS Profile
- **tls\_version** (*str*) – supported tls version, valid options are TLSv1.1, TLSv1.2, TLSv1.3
- **use\_only\_subject\_alt\_name** (*bool*) – Use Only Subject Alt Name when the TLS identity is a DNS name
- **accept\_wildcard** (*bool*) – Does server identity check accept wildcards
- **check\_revocation** (*bool*) – Is certificate revocation checked
- **tls\_cryptography\_suites** (*str*, *TLSCryptographySuite*) – allowed cryptography suites for this profile. Uses NIST profile if not specified
- **crl\_delay** (*int*) – Delay time (hours) for fetching CRL
- **ocsp\_delay** (*int*) – Ignore OCSP failure for (hours)
- **ignore\_network\_issues** (*bool*) – Ignore revocation check failures due to network issues
- **tls\_trusted\_ca\_ref** (*list*) – Trusted Certificate Authorities, empty list means trust any
- **comment** (*str*) – optional comment

#### Raises

- *CreateElementFailed* – failed to create element with reason
- *ElementNotFound* – specified element reference was not found

#### Return type

*TLSProfile*

#### 14.3.3.4 TLSIdentity

**class** smc.administration.certificates.tls.TLSIdentity(*tls\_field, tls\_value*)

Bases: `NestedDict`

New in version 0.6.2: Requires SMC >= 6.4

A TLS Identity represents a field and value pair that will be used to validate a TLS certificate. This can be used in various areas where TLS is used such as VPN.

Valid tls field types are:

DNSName IPAddress CommonName DistinguishedName SHA-1 SHA-256 SHA-512 MD5 Email  
user\_principal\_name

#### 14.3.3.5 TLSCryptographySuite

**class** smc.administration.certificates.tls.TLSCryptographySuite(*name=None, \*\*meta*)

Bases: `Element`

This represents a TLS Cryptography Suite Set used in various configurations that require a TLS Profile such as SSL VPN Tunneling, Reverse Web Proxy, ActiveDirectory TLS, etc.

**static** `ciphers`(*from\_suite=None*)

This is a helper method that will return all of the cipher strings used in a specified TLSCryptographySuite or returns the system default NIST profile list of ciphers. This can be used as a helper to identify the ciphers to specify/add when creating a new TLSCryptographySuite.

**Return type**  
`dict`

**classmethod** `create`(*name, comment=None, \*\*ciphers*)

Create a new TLSCryptographySuite. The ciphers kwargs should be a dict with the cipher suite string as key and boolean value to indicate if this cipher should be enabled. To obtain the valid cipher suite string name, use the following method:

```
cipher_strings = TLSCryptographySuite.ciphers()
```

Then to create a custom cipher suite, provide the ciphers as a dict of kwargs. In this example, create a TLS Crypto Suite that only enables AES 256 bit ciphers:

```
only256 = dict(((cipher, True) for cipher in TLSCryptographySuite.ciphers()
                if 'aes_256' in cipher))

mytls = TLSCryptographySuite.create(name='mytls', **only256)
```

##### Parameters

- **name** (*str*) – name of this TLS Crypto suite
- **ciphers** (*dict*) – dict of ciphers with cipher string as key and bool as value, True enables the cipher

##### Raises

`CreateElementFailed` – failed to create element with reason

##### Return type

`TLSCryptographySuite`

### 14.3.3.6 TLSCertificateAuthority

**class** smc.administration.certificates.tls.TLSCertificateAuthority(*name=None, \*\*meta*)

Bases: *ImportExportCertificate, Element*

TLS Certificate authorities. When using TLS Server Credentials for decryption, import the root CA for the any TLS Server certificates the leverage the root CA.

#### Variables

- **certificate** (*str*) – base64 encoded certificate for this CA
- **crl\_checking\_enabled** (*bool*) – whether CRL checking is turned on
- **internal\_ca** (*bool*) – is this an internal CA (default: false)
- **oscp\_checking\_enabled** (*bool*) – is OSCP validation enabled

**classmethod** create(*name, certificate*)

Create a TLS CA. The certificate must be compatible with OpenSSL and be in PEM format. The certificate can be either a file with the Root CA, or a raw string starting with BEGIN CERTIFICATE, etc. When creating a TLS CA, you must also import the CA certificate. Once the CA is created, it is possible to import a different certificate to map to the CA if necessary.

#### Parameters

- **name** (*str*) – name of root CA
- **certificate** (*str, file*) – The root CA contents

#### Raises

- **CreateElementFailed** – failed to create the root CA
- **ValueError** – if loading from file and no certificates present
- **IOError** – cannot find specified file for certificate

#### Return type

*TLSCertificateAuthority*

### 14.3.3.7 ClientProtectionCA

**class** smc.administration.certificates.tls.ClientProtectionCA(*name=None, \*\*meta*)

Bases: *ImportPrivateKey, ImportExportCertificate, Element*

Changed in version 0.7.0: Deprecated *create* method

Client Protection Certificate Authority elements are used to inspect TLS traffic between an internal client and an external server for outbound decryption.

When an internal client makes a connection to an external server that uses TLS, the engine generates a substitute certificate that allows it to establish a secure connection with the internal client. The Client Protection Certificate Authority element contains the credentials the engine uses to sign the substitute certificate it generates.

#### Variables

- **certificate** (*str*) – base64 encoded certificate for this CA
- **crl\_checking\_enabled** (*bool*) – whether CRL checking is turned on
- **internal\_ca** (*bool*) – is this an internal CA (default: false)
- **oscp\_checking\_enabled** (*bool*) – is OSCP validation enabled

**Note:** If the engine does not use a signing certificate that is already trusted by users web browsers when it signs the substitute certificates it generates, users receive warnings about invalid certificates. To avoid these warnings, you must either import a signing certificate that is already trusted, or configure users web browsers to trust the engine signing certificate.

---

**classmethod** `create_self_signed(name, public_key_algorithm='rsa', life_time=365, key_length=2048, **kwargs)`

Changed in version 0.7.0: *prefix* and *password* argument deprecated in SMC > 6.5.1.

Create a self signed client protection CA. To prevent browser warnings during decryption, you must trust the signing certificate in the client browsers.

#### Parameters

- **name** (*str*) – Name of this ex: “SG Root CA” Used as Key. Real common name will be derivated at creation time with a `uniqueId`.
- **public\_key\_algorithm** – public key algorithm, either `rsa`, `dsa` or `ecdsa`
- **life\_time** (*str*, *int*) – lifetime in days for CA
- **key\_length** (*int*) – length in bits, either 1024 or 2048

#### Raises

- `CreateElementFailed` – creating element failed
- `ActionCommandFailed` – failed to self sign the certificate

#### Return type

`ClientProtectionCA`

**classmethod** `import_signed(name, certificate, private_key)`

Import a signed certificate and private key as a client protection CA.

This is a shortcut method to the 3 step process:

- Create CA with name
- Import certificate
- Import private key

Create the CA:

```
ClientProtectionCA.import_signed(  
    name='myclientca',  
    certificate_file='/path/to/server.crt'  
    private_key_file='/path/to/server.key')
```

#### Parameters

- **name** (*str*) – name of client protection CA
- **certificate\_file** (*str*) – fully qualified path or string of certificate
- **private\_key\_file** (*str*) – fully qualified path or string of private key

#### Raises

- `CertificateImportError` – failure during import
- `IOError` – failure to find certificate files specified

**Return type**  
*ClientProtectionCA*

#### 14.3.3.8 VPNCertificateCA

#### 14.3.3.9 GatewayCertificate

### 14.3.4 Domains

**class** smc.administration.system.**AdminDomain**(name=None, \*\*meta)

Bases: *Element*

Administrative domain element. Domains are used to provide object based segmentation within SMC. If domains are in use, you can log in directly to a domain to modify contents within that domain.

Find all available domains:

```
>>> list(AdminDomain.objects.all())
[AdminDomain(name=Shared Domain)]
>>> admindomain_obj = AdminDomain(name=mydomain)
>>> admindomain_obj.announcement_enabled
True
>>> admindomain_obj.announcement_message
test
>>> admindomain_obj.update(announcement_enabled=False)
>>> admindomain_obj.announcement_enabled
False
```

---

**Note:** Admin Domains require and SMC license.

---

#### **property announcement\_enabled**

Display flag of announcement message :rtype: bool

#### **property announcement\_message**

Announcement message to be displayed before the login window. :rtype: str

#### **property category\_filter\_system**

Flag to know if we need to show system elements :@rtype: bool

#### **property contact\_number**

Contact Number :rtype: str

**classmethod create**(name, announcement\_enabled=False, announcement\_message=None, contact\_email=None, contact\_number=None, category\_filter\_system=True, show\_not\_categorized=True, user\_alert\_check=[], comment=None)

Create a new Admin Domain element for SMC objects.

Example:

```
>>> admindomain_obj=AdminDomain.create(name='mydomain',announcement_
↪ enabled=True, announcement_message='test', comment='mycomment
↪ ')
>>> AdminDomain(name=mydomain)
```

**Parameters**

- **name** (*str*) – name of domain
- **announcement\_enabled** (*bool*) – Enable or disable display of announcement message
- **announcement\_message** (*str*) – Announcement message to be displayed before the login window
- **contact\_email** (*str*) – contact email
- **contact\_number** (*str*) – contact phone number
- **category\_filter\_system** (*bool*) – Flag to know if we need to show system elements. By default, true.
- **show\_not\_categorized** (*bool*) – Flag to know if we need to show not categorized. By default, true.
- **user\_alert\_check** (*list*) – The list of User alert checks.
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta

**Return type**

*AdminDomain*

**get\_active\_alerts**(*full=True*)

Available for all SMC API Versions but only for SMC Version above 7.1 (7.1 included)

Return active alerts for the requested domain

**:optional param full ( default value is true ). When set to false, juste retrieve the log key of each entry ( timestamp, component id, event id ).**

**Returns**

list of alert monitoring entries : session\_monitoring.SessionMonitoringResult

**Return type**

*SerializedIterable(Route)*

**property show\_not\_categorized**

Flag to know if we need to show not categorized. :rtype: bool

**property user\_alert\_check**

The list of User alert checks. :rtype: list(UserAlertCheck)

## 14.3.5 License

### 14.3.5.1 License

Module representing read-only licenses in SMC

```
class smc.administration.license.License(**data)
```

Valid attributes (read-only) are:

#### Variables

- **binding** – master license binding serial number
- **binding\_state** – state of license, unassigned, bound, etc
- **bindings** – which node is the license bound to
- **customer\_name** – customer name, if any
- **enabled\_feature\_packs** – additional feature licenses
- **expiration\_date** – when license expires
- **features** – features enabled on this license
- **granted\_date** – when license date began
- **license\_id** – license ID (unique for each license)
- **license\_version** – max version for this license
- **maintenance\_contract\_expires\_date** – date/time support ends
- **management\_server\_binding** – management server binding POS
- **proof\_of\_license** – proof of license key
- **type** – type of license (SECNODE, Mgmt, etc)

```
class smc.administration.license.Licenses(licenses)
```

List of all available licenses for this Management Server.

## 14.3.6 Scheduled Tasks

New in version 0.5.7: Requires SMC version >= 6.3.2

Scheduled tasks are administrative processes that can run either immediately after being defined, or scheduled to run on a regular basis. Scheduled tasks in the SMC are defined under Administration->Tasks->Definition.

Some tasks are read-only, meaning they are system elements and cannot be modified or copied and can therefore only be scheduled (these task related classes will not have a *create* method). Other tasks can be created and custom settings can be defined. Check the documentation for each task to determine the capabilities.

All tasks inherit the ScheduledTaskMixin which provides a *start* method and access to a TaskSchedule instance through the *task\_schedule* property. The associated TaskSchedule defines whether to run the task ongoing and details specifying when the task should be run and how often.

An example follows that shows how to use a refresh policy task. Other tasks use the same API syntax.

Finding existing tasks for a specific task type:

```
for task in RefreshPolicyTask.objects.all():
    print(task, task.task_schedule)
```

Review an existing task and it's task schedule:

```
task = RefreshPolicyTask(name='mytask')
for schedule in task.task_schedule:
    print(schedule.activation_date, schedule.activated)
```

Create a refresh policy refresh task:

```
task = RefreshPolicyTask.create(
    name='mytask',
    engines=[Engine('engine1'), Engine('engine2')],
    comment='some comment')
```

A created task can always be run at any time without having to set a schedule for the task by calling *start* on the task:

```
task = RefreshPolicyTask('mytask')
task.start()
```

A task can also be scheduled for a future time. Adding a scheduled run to the task requires that we first obtain the task and add the schedule to it. This can be done when creating the task, or the retrieved after:

```
task = RefreshPolicyTask.create(
    name='mytask',
    engines=[Engine('engine1'), Engine('engine2')],
    comment='refresh policy on specified engines')

task.add_schedule(
    name='refresh_policy_on_saturday',
    activation_date=1512325716000, # 12/04/2017 00:00:00
    day_period='weekly',
    day_mask=128,
    comment='tun this task weekly')
```

You can also specify tasks that run on a regular interval, such as monthly:

```
task = RefreshPolicyTask(name='mytask')
task.add_schedule(
    name='run_monthly',
    activation_date=1512367200000, # Start 12/4/2017 at 00:00:00
    day_period='monthly')
```

Repeat a task for a period of time, then disable task on specified date:

```
task = DeleteLogTask.create(
    name='Delete SMC Server logs',
    servers='all',
    time_range='last_full_month',
    all_logs=True)

task.add_schedule(
    name='Run for 6 months',
    activation_date=1512367200000, # Start 12/04/2017
    day_period='monthly',
    repeat_until_date=1528088400000, # End 06/04/2018
    comment='purge log task')
```

---

**Note:** You can use the helper method `smc.base.util.datetime_to_ms()` for obtaining millisecond times for scheduled tasks.

---

**class** `smc.administration.scheduled_tasks.ArchiveLogTask`(*name=None, \*\*meta*)

Bases: [ScheduledTaskMixin](#), [Element](#)

An archive log task defines a way to archive log data from the SMC. When defining the task, you specify which servers to archive (typically management AND log server/s), and which log types to archive.

---

**Note:** Log tasks currently support pre-defined time ranges such as ‘yesterday’, ‘last\_week’, etc. If creating custom time ranges for tasks, use the SMC.

---

**classmethod** `create`(*name, servers=None, time\_range='yesterday', filter\_for\_export=None, filter\_for\_delete=None, delete\_source\_data=False, all\_logs=False, server\_directory\_lst=None, is\_local\_location=False, comment=None, \*\*kwargs*)

Create a new archive log task. Provide True to `all_logs` to archive all log types. Otherwise provide kwargs to specify each log by type of interest.

#### Parameters

- **name** (*str*) – name for this task
- **servers** (*list*([ManagementServer](#) or [LogServer](#))) – servers to back up. Servers must be instances of management servers or log servers. If no value is provided, all servers are backed up.
- **time\_range** (*str*) – specify a time range for the archive. Valid options are ‘yesterday’, ‘last\_full\_week\_sun\_sat’, ‘last\_full\_week\_mon\_sun’, ‘last\_full\_month’ (default ‘yesterday’)
- **filter\_for\_export** ([FilterExpression](#)) – The Filter expression for the archive/export task to be able to filter logs to consider. (default: `FilterExpression('Match All')`)
- **filter\_for\_delete** ([FilterExpression](#)) – The Filter expression for the archive/delete task to be able to filter logs for deletion. (default: `FilterExpression('Match None')`)
- **delete\_source\_data** (*bool*) – Flag to know if after the archive operation, logs need to be deleted.
- **all\_logs** (*bool*) – if True, all log types will be archived. If this is True, kwargs are ignored (default: False)
- **server\_directory\_lst** (*list*) – Server directories used to retrieve logs. see [server\\_directory\(\)](#) for keyword arguments and default values.
- **is\_local\_location** (*bool*) – Flag to know if the archive/export file will be stored on the log server or locally.
- **kwargs** – see [log\\_target\\_types\(\)](#) for keyword arguments and default values.

#### Raises

- [ElementNotFound](#) – specified servers were not found
- [CreateElementFailed](#) – failure to create the task

#### Returns

the task

**Return type***ArchiveLogTask***class** smc.administration.scheduled\_tasks.DeleteLogTask(*name=None, \*\*meta*)Bases: *ScheduledTaskMixin, Element*

A delete log task defines a way to purge log data from the SMC. When defining the task, you specify which servers to delete from (typically management AND log server/s), and which log types to delete.

---

**Note:** Log tasks currently support pre-defined time ranges such as ‘yesterday’, ‘last\_week’, etc. If creating custom time ranges for tasks, use the SMC.

---

**classmethod** create(*name, servers=None, time\_range='yesterday', all\_logs=False, filter\_for\_delete=None, comment=None, \*\*kwargs*)

Create a new delete log task. Provide True to all\_logs to delete all log types. Otherwise provide kwargs to specify each log by type of interest.

**Parameters**

- **name** (*str*) – name for this task
- **servers** (*list*(*ManagementServer* or *LogServer*)) – servers to back up. Servers must be instances of management servers or log servers. If no value is provided, all servers are backed up.
- **time\_range** (*str*) – specify a time range for the deletion. Valid options are ‘yesterday’, ‘last\_full\_week\_sun\_sat’, ‘last\_full\_week\_mon\_sun’, ‘last\_full\_month’ (default ‘yesterday’)
- **filter\_for\_delete** (*FilterExpression*) – optional filter for deleting. (default: *FilterExpression*(‘Match All’))
- **all\_logs** (*bool*) – if True, all log types will be deleted. If this is True, kwargs are ignored (default: False)
- **kwargs** – see *log\_target\_types()* for keyword arguments and default values.

**Raises**

- *ElementNotFound* – specified servers were not found
- *CreateElementFailed* – failure to create the task

**Returns**

the task

**Return type***DeleteLogTask***class** smc.administration.scheduled\_tasks.DeleteOldRunTask(*name=None, \*\*meta*)Bases: *ScheduledTaskMixin, Element*

A read-only task to delete the task history from already run tasks. This is generally a recommended task to run on a monthly basis to purge the old task data.

**class** smc.administration.scheduled\_tasks.DeleteOldSnapshotsTask(*name=None, \*\*meta*)Bases: *ScheduledTaskMixin, Element*

A read-only management server task to delete snapshots since the last scheduled run. For example, if this task is configured to run once per month, snapshots older than 1 month will be deleted.

**class** `smc.administration.scheduled_tasks.DisableUnusedAdminTask(name=None, **meta)`

Bases: [ScheduledTaskMixin](#), [Element](#)

A read-only task to disable any administrator account that has not been used within the time set in the Administrator password policy.

**class** `smc.administration.scheduled_tasks.ExportLogTask(name=None, **meta)`

Bases: [ScheduledTaskMixin](#), [Element](#)

An export log task defines a way to export log data from the SMC. When defining the task, you specify which servers to export from (typically management AND log server/s), and which log types to export.

---

**Note:** Log tasks currently support pre-defined time ranges such as ‘yesterday’, ‘last\_week’, etc. If creating custom time ranges for tasks, use the SMC.

---

**classmethod** `create(name, servers=None, time_range='yesterday', file_name=None, file_format='xml', is_local_location=True, filter_for_export=None, all_logs=False, overwrite_file_flag='overwrite', server_directory_lst=None, comment=None, **kwargs)`

Create a new export log task. Provide True to all\_logs to export all log types. Otherwise provide kwargs to specify each log by type of interest.

#### Parameters

- **name** (*str*) – name for this task
- **servers** (*list*([ManagementServer](#) or [LogServer](#))) – servers to back up. Servers must be instances of management servers or log servers. If no value is provided, all servers are backed up.
- **time\_range** (*str*) – specify a time range for the export. Valid options are ‘yesterday’, ‘last\_full\_week\_sun\_sat’, ‘last\_full\_week\_mon\_sun’, ‘last\_full\_month’ (default ‘yesterday’)
- **file\_name** (*str*) – name of the file to export
- **file\_format** (*str*) – format of the file to export options are:
  - \***csv**\* export in CSV format. \***xml**\* export in XML format. \***zip**\* export in ZIP format. \***cef**\* export in CEF format. \***leef**\* export in LEEF format. \***esm**\* export in ESM format. \***snoop**\* export IPS recordings as SNOOP. \***pcap**\* export IPS recordings as PCAP.
- **filter\_for\_export** ([FilterExpression](#)) – The Filter expression for the archive/export task to be able to filter logs to consider. (default: [FilterExpression](#)(‘Match All’))
- **all\_logs** (*bool*) – if True, all log types will be deleted. If this is True, kwargs are ignored (default: False)
- **server\_directory\_lst** (*list*) – see [server\\_directory\(\)](#) for keyword arguments and default values.
- **is\_local\_location** (*bool*) – Flag to know if the archive/export file will be stored on the log server or locally.
- **overwrite\_file\_flag** (*str*) – The overwrite options: \***append**\*: append option. \***overwrite**\*: overwrite option. \***use\_number\_in\_file\_name**\* create an unique file with number as name. \***fail\_task**\*: fail the task.

- **kwargs** – see `log_target_types()` for keyword arguments and default values.

**Raises**

- **`ElementNotFound`** – specified servers were not found
- **`CreateElementFailed`** – failure to create the task

**Returns**

the task

**Return type**

*ExportLogTask*

```
class smc.administration.scheduled_tasks.FetchCertificateRevocationTask(name=None, **meta)
```

Bases: *ScheduledTaskMixin*, *Element*

A read-only management server task to download updated certificate revocation lists.

```
class smc.administration.scheduled_tasks.RefreshMasterEnginePolicyTask(name=None, **meta)
```

Bases: *ScheduledTaskMixin*, *Element*

Refresh a Master Engine and virtual policy task.

---

**Note:** This task is only relevant for Master Engines types.

---

```
classmethod create(name, master_engines, comment=None, **kwargs)
```

Create a refresh task for master engines.

**Parameters**

- **name** (*str*) – name of task
- **master\_engines** (*list*(*MasterEngine*)) – list of master engines for this task
- **comment** (*str*) – optional comment
- **kwargs** – to support new attributes like `preserve_connections`, `snapshot_creation`.

**Raises**

**`CreateElementFailed`** – failed to create the task, i.e. no valid engines provided

**Returns**

the task

**Return type**

*RefreshMasterEnginePolicyTask*

```
class smc.administration.scheduled_tasks.RefreshPolicyTask(name=None, **meta)
```

Bases: *ScheduledTaskMixin*, *Element*

A scheduled task associated with refreshing policy on engine/s. A refresh will push an existing policy that is already mapped to the engine/s. Use *UploadPolicyTask* to create a task that will assign a policy to an engine/s and upload.

---

**Note:** Any engine can force a policy refresh on the engine node directly by calling `engine.refresh()`, or from the engines assigned policy by calling `policy.refresh(engine)` also.

---

**classmethod create**(*name*, *engines*, *comment=None*, *validate\_policy=True*, *\*\*kwargs*)

Create a refresh policy task associated with specific engines. A policy refresh task does not require a policy be specified. The policy used in the refresh will be the policy already assigned to the engine.

#### Parameters

- **name** (*str*) – name of this task
- **engines** (*list*(*Engine*)) – list of Engines for the task
- **comment** (*str*) – optional comment
- **validate\_policy** (*bool*) – validate the policy before upload. If set to true, validation kwargs can also be provided if customization is required, otherwise default validation settings are used.
- **kwargs** – see *policy\_validation\_settings()* for keyword arguments and default values.

#### Raises

- *ElementNotFound* – engine specified does not exist
- *CreateElementFailed* – failure to create the task

#### Returns

the task

#### Return type

*RefreshPolicyTask*

**class** `smc.administration.scheduled_tasks.RemoteUpgradeTask`(*name=None*, *\*\*meta*)

Bases: *ScheduledTaskMixin*, *Element*

A Remote Upgrade task will Upgrade Firewall to a specified version. If an engine specified

**classmethod create**(*name*, *engines*, *package*, *comment=None*, *\*\*kwargs*)

Create an upload policy task associated with specific engines. A policy reassigns any policies that might be assigned to a specified engine.

#### Parameters

- **name** (*str*) – name of this task
- **engines** (*list*(*Engine*)) – list of Engines for the task
- **package** (*str*) – Package to assign to the engine/s
- **comment** (*str*) – optional comment

#### Raises

- *ElementNotFound* – engine specified does not exist
- *CreateElementFailed* – failure to create the task

#### Returns

the task

#### Return type

*RemoteUpgradeTask*

```
class smc.administration.scheduled_tasks.RenewGatewayCertificatesTask(name=None, **meta)
```

Bases: [ScheduledTaskMixin](#), [Element](#)

A read-only management server task that renews certificates on internal gateways which have automatic certificate renewal enabled.

```
class smc.administration.scheduled_tasks.RenewInternalCATask(name=None, **meta)
```

Bases: [ScheduledTaskMixin](#), [Element](#)

A read-only management server task that renews certificate authorities used in system communications and send alerts about expiring certificate authorities.

```
class smc.administration.scheduled_tasks.RenewInternalCertificatesTask(name=None, **meta)
```

Bases: [ScheduledTaskMixin](#), [Element](#)

A read-only management server task that renews certificates used in systems communications and send alerts about expiring certificates.

```
class smc.administration.scheduled_tasks.SGInfoTask(name=None, **meta)
```

Bases: [ScheduledTaskMixin](#), [Element](#)

An SGInfo task is used for obtaining support data from the engine/s.

---

**Note:** An sginfo can be executed directly on an engine node by calling the `node.sginfo()` method directly.

---

#### Variables

- **include\_core\_files** (*bool*) – whether to include core files in output
- **include\_slapcat\_output** (*bool*) – include slapcat in output

**Warning:** For an sginfo to be readable, the engine must not have the ‘encrypt\_configuration’ field enabled on the engine or the data will be unreadable.

```
classmethod create(name, engines, include_core_files=False, include_slapcat_output=False,
                  comment=None)
```

Create an sginfo task.

#### Parameters

- **name** (*str*) – name of task
- **engines** (*list* ([Engine](#))) – list of engines to apply the sginfo task
- **include\_core\_files** (*bool*) – include core files in the sginfo backup (default: False)
- **include\_slapcat\_output** (*bool*) – include output from a slapcat command in output (default: False)

#### Raises

- [ElementNotFound](#) – engine not found
- [CreateElementFailed](#) – create the task failed

#### Returns

the task

**Return type***SGInfoTask***class** `smc.administration.scheduled_tasks.ScheduledTaskMixin`Bases: `object`

Actions common to all scheduled tasks.

**add\_schedule**(*name*, *activation\_date*, *day\_period*='one\_time', *final\_action*='ALERT\_FAILURE', *activated*=True, *minute\_period*='one\_time', *day\_mask*=None, *repeat\_until\_date*=None, *comment*=None)

Add a schedule to an existing task.

**Parameters**

- **name** (*str*) – name for this schedule
- **activation\_date** (*int*) – when to start this task. Activation date should be a UTC time represented in milliseconds.
- **day\_period** (*str*) – when this task should be run. Valid options: 'one\_time', 'daily', 'weekly', 'monthly', 'yearly'. If 'daily' is selected, you can also provide a value for 'minute\_period'. (default: 'one\_time')
- **minute\_period** (*str*) – only required if day\_period is set to 'daily'. Valid options: 'each\_quarter' (15 min), 'each\_half' (30 minutes), or 'hourly', 'one\_time' (default: 'one\_time')
- **day\_mask** (*int*) – If the task day\_period=weekly, then specify the day or days for repeating. Day masks are: sun=1, mon=2, tue=4, wed=8, thu=16, fri=32, sat=64. To repeat for instance every Monday, Wednesday and Friday, the value must be 2 + 8 + 32 = 42
- **final\_action** (*str*) – what type of action to perform after the scheduled task runs. Options are: 'ALERT\_FAILURE', 'ALERT', or 'NO\_ACTION' (default: ALERT\_FAILURE)
- **activated** (*bool*) – whether to activate the schedule (default: True)
- **repeat\_until\_date** (*str*) – if this is anything but a one time task run, you can specify the date when this task should end. The format is the same as the *activation\_date* param.
- **comment** (*str*) – optional comment

**Raises***ActionCommandFailed* – failed adding schedule**Returns**

None

**property resources**

Resources associated with this task. Depending on the task, this may be engines, policies, servers, etc.

**Returns**

list of Elements

**Return type***list***start()**

Start the scheduled task now. Task can then be tracked by using common Task methods.

**Raises***ActionCommandFailed* – failed starting task

**Returns**

return as a generic Task

**Return type**

*Task*

**property task\_schedule**

Return any task schedules associated with this scheduled task.

**Raises**

*ActionCommandFailed* – failure to retrieve task schedule

**Returns**

list of task schedules

**Return type**

*TaskSchedule*

**class** smc.administration.scheduled\_tasks.**ServerBackupTask**(name=None, \*\*meta)

Bases: *ScheduledTaskMixin*, *Element*

A task that will back up the Management Server/s, Log Server/s and optionally the Log Server data.

**Variables**

**log\_data\_must\_be\_saved** (*bool*) – whether to back up logs

**classmethod create**(name, servers, backup\_log\_data=False, encrypt\_password=None, comment=None, path=None, script=None)

Create a new server backup task. This task provides the ability to backup individual or all management and log servers under SMC management.

**Parameters**

- **name** (*str*) – name of task
- **servers** (*list*(*ManagementServer* or *LogServer*)) – servers to back up. Servers must be instances of management servers or log servers. If no value is provided all servers are backed up.
- **backup\_log\_data** (*bool*) – Should the log files be backed up. This field is only relevant if a Log Server is backed up.
- **encrypt\_password** (*str*) – Provide an encrypt password if you want this backup to be encrypted.
- **comment** (*str*) – optional comment
- **path** – The path to store the backup
- **script** – Script to execute after backup has been generated

**Raises**

- *ElementNotFound* – specified servers were not found
- *CreateElementFailed* – failure to create the task

**Returns**

the task

**Return type**

*ServerBackupTask*

**class** `smc.administration.scheduled_tasks.SystemSnapshotTask(name=None, **meta)`

Bases: [ScheduledTaskMixin](#), [Element](#)

A read-only task that will make a snapshot of all system elements after a updating a dynamic package on SMC.

**class** `smc.administration.scheduled_tasks.TaskSchedule(**meta)`

Bases: [SubElement](#)

A task schedule is associated with a given task type that defines when the scheduled task should run.

#### Variables

- **day\_period** ([str](#)) – how often to run the task
- **final\_action** ([str](#)) – what to do when the task is complete
- **minute\_period** ([str](#)) – if day\_period is set to hourly, when to run within the hour.

#### **activate()**

If a task is suspended, this will re-activate the task. Usually it's best to check for activated before running this:

```
task = RefreshPolicyTask('mytask')
for scheduler in task.task_schedule:
    if scheduler.activated:
        scheduler.suspend()
    else:
        scheduler.activate()
```

#### **property activated**

Whether this schedule is active for this task.

##### **Return type**

[bool](#)

#### **property activation\_date**

Return the UTC time when the task is set to first run. The activation date is returned as a python datetime object.

##### **Returns**

datetime object in format '%Y-%m-%d %H:%M:%S.%f'

##### **Return type**

[datetime.datetime](#)

#### **suspend()**

Suspend this scheduled task.

##### **Raises**

[ActionCommandFailed](#) – failed to suspend, already suspended. Call activate on this task to reactivate.

##### **Returns**

None

**class** `smc.administration.scheduled_tasks.TrafficCaptureInterfaceSettings(data)`

Bases: [NestedDict](#)

**classmethod** `create(node_ref=None, pint_ref=None, filter=None)`

#### **Parameters**

- **node\_ref** ([Node](#)) – Individual engine node on which traffic is being captured. Required.
- **pint\_ref** ([PhysicalInterface](#)) – Physical Interface Link on which traffic is being captured. Required.
- **filter** ([str](#)) – IP Address for filtering. By default, no ip address filtering.

**Returns**

[TrafficCaptureInterfaceSettings](#)

**classmethod** `create_interface_setting_for_all_node(engine, pint_ref=None, filter=None)`

This method is used to create interface settings for all nodes of the engine with the same filter for the given interface.

**Parameters**

- **engine** ([Engine](#)) – Engine on which traffic is being captured. Required.
- **pint\_ref** ([PhysicalInterface](#)) – Physical Interface Link on which traffic is being captured. Required.
- **filter** ([str](#)) – IP Address for filtering. By default, no ip address filtering.

**Returns**

`list(TrafficCaptureInterfaceSettings)`

**class** `smc.administration.scheduled_tasks.TrafficCaptureTask(name=None, **meta)`

Bases: [ScheduledTaskMixin](#), [Element](#)

**classmethod** `create(name, interface_settings=None, max_file_size_in_mb=500, capture_headers_only=False, description=None, duration_in_sec=3600, sg_info_option=False, destination_file=None, comment=None)`

This creates the Traffic Capture task element. It captures traffic on specific engine's physical interface with some parameters given below. :param str name: name of the traffic capture task. :param list([TrafficCaptureInterfaceSettings](#)) interface\_settings: The traffic capture interface settings to capture traffic.

**Parameters**

- **max\_file\_size\_in\_mb** ([int](#)) – The Maximum size of file the Traffic to be captured in MB.
- **capture\_headers\_only** ([bool](#)) – Decide whether to capture only header or not. By default: False
- **description** ([str](#)) – The Description about traffic capture task.
- **duration\_in\_sec** ([int](#)) – The duration in second traffic capture task will run.
- **sg\_info\_option** ([int](#)) – This flag will decide to include sginfo. By default: False.
- **destination\_file** ([str](#)) – The destination file path of traffic capture file to store.
- **comment** ([str](#)) – optional comment.

@return:

---

```
class smc.administration.scheduled_tasks.UploadPolicyTask(name=None, **meta)
```

Bases: [ScheduledTaskMixin](#), [Element](#)

An upload policy task will assign a specified policy to an engine or group of engines and upload. If an engine specified has an existing policy assigned, the engine will be reassigned the specified policy. If the intent is to create a policy task to push an existing assigned policy, use [RefreshPolicyTask](#) instead.

---

**Note:** Policy upload on an engine can be done from the engine node itself by calling `engine.upload('policy_name')` or from a policy directly by `policy.upload('engine_name')`.

---

```
classmethod create(name, engines, policy, comment=None, validate_policy=False, **kwargs)
```

Create an upload policy task associated with specific engines. A policy reassigns any policies that might be assigned to a specified engine.

#### Parameters

- **name** (*str*) – name of this task
- **engines** (*list*([Engine](#))) – list of Engines for the task
- **policy** ([Policy](#)) – Policy to assign to the engine/s
- **comment** (*str*) – optional comment
- **validate\_policy** (*bool*) – validate the policy before upload. If set to true, validation kwargs can also be provided if customization is required, otherwise default validation settings are used.
- **kwargs** – see [policy\\_validation\\_settings\(\)](#) for keyword arguments and default values.

#### Raises

- [ElementNotFound](#) – engine or policy specified does not exist
- [CreateElementFailed](#) – failure to create the task

#### Returns

the task

#### Return type

[UploadPolicyTask](#)

```
class smc.administration.scheduled_tasks.ValidatePolicyTask(name=None, **meta)
```

Bases: [ScheduledTaskMixin](#), [Element](#)

Run a policy validation task. This does not perform a policy push. This may be useful if you want to validate any pending changes before a future policy push.

#### Variables

**policy** ([Element](#)) – The policy associated with this task

```
classmethod create(name, engines, policy=None, comment=None, **kwargs)
```

Create a new validate policy task. If a policy is not specified, the engines existing policy will be validated. Override default validation settings as kwargs.

#### Parameters

- **name** (*str*) – name of task
- **engines** (*list*([Engine](#))) – list of engines to validate

- **policy** (*Policy*) – policy to validate. Uses the engines assigned policy if none specified.
- **kwargs** – see *policy\_validation\_settings()* for keyword arguments and default values.

**Raises**

- *ElementNotFound* – engine or policy specified does not exist
- *CreateElementFailed* – failure to create the task

**Returns**

the task

**Return type**

*ValidatePolicyTask*

`smc.administration.scheduled_tasks.log_target_types(all_logs=False, **kwargs)`

Log targets for log tasks. A log target defines the log types that will be affected by the operation. For example, when creating a DeleteLogTask, you can specify which log types are deleted.

**Parameters**

- **for\_alert\_event\_log** (*bool*) – alert events traces (default: False)
- **for\_alert\_log** (*bool*) – alerts (default: False)
- **for\_fw\_log** (*bool*) – NGFW Engine logs (default: False)
- **for\_ips\_log** (*bool*) – IPS logs (default: False)
- **for\_ips\_recording** (*bool*) – any IPS pcaps (default: False)
- **for\_l2fw\_log** (*bool*) – layer 2 Engine logs (default: False)
- **for\_third\_party\_log** (*bool*) – any 3rd party logs (default: False)

**Returns**

dict of log targets

`smc.administration.scheduled_tasks.policy_validation_settings(**kwargs)`

Set policy validation settings. This is used when policy based tasks are created and *validate\_policy* is set to True. The following kwargs can be overridden in the create constructor.

**Parameters**

- **configuration\_validation\_for\_alert\_chain** (*bool*) – default False
- **duplicate\_rule\_check\_settings** (*bool*) – default False
- **empty\_rule\_check\_settings** (*bool*) – default True
- **empty\_rule\_check\_settings\_for\_alert** (*bool*) – default False
- **general\_check\_settings** (*bool*) – default True
- **nat\_modification\_check\_settings** (*bool*) – default True
- **non\_supported\_feature** (*bool*) – default True
- **routing\_modification\_check** (*bool*) – default False
- **unreachable\_rule\_check\_settings** (*bool*) – default False
- **vpn\_validation\_check\_settings** (*bool*) – default True

**Returns**

dict of validation settings

```
smc.administration.scheduled_tasks.server_directories_settings(srv_directories)
```

Server directories used to retrieve logs.. This is needed for export or archive logs tasks.

#### Parameters

**server\_directories** (*list*) – list of server\_directory

#### Returns

dict of server\_directories settings

```
smc.administration.scheduled_tasks.server_directory(**kwargs)
```

create a server directory dict. This is used for export/archive logs tasks. The following kwargs can be overridden in the create constructor.

#### Parameters

**archive\_mask** (*bool*) – The bit mask of selected archive directories, -1 for all. default -1

:param bool only\_archive: Indicates whether only the active directory is excluded. :param LogServer / Server server: The server used for log retrieving. :return: dict of server\_directory

## 14.3.7 Reports

New in version 0.6.0: Requires SMC version >= 6.3

Reports generated from the SMC. Provides an interface to running existing report designs and exporting their contents.

Example usage:

```
>>> from smc.administration.reports import ReportDesign, ReportTemplate, Report
```

List all available report templates:

```
>>> list(ReportTemplate.objects.all())
[ReportTemplate(name=Firewall Weekly Summary),
 ReportTemplate(name=Firewall Daily Summary from Specific Firewall),
 ReportTemplate(name=Firewall Multi-Link Usage)
 ...
```

Create a report design using an existing report template:

```
>>> template = ReportTemplate('Firewall Weekly Summary')
>>> template.create_design('myfirewallreport')
ReportDesign(name=myfirewallreport)
```

Generate a report based on an existing or created report design:

```
>>> list(ReportDesign.objects.all())
[ReportDesign(name=Application and Web Security), ReportDesign(name=myfirewallreport)]
...
>>> design = ReportDesign('Application and Web Security')
>>> poller = design.generate(wait_for_finish=True)
>>> while not poller.done():
...     poller.wait(3)
...
>>> poller.task.resource
>>> Report(name=Application and Web Security #1515295820751)
...

```

(continues on next page)

(continued from previous page)

```
>>> design.report_files
[Report(name=Application and Web Security #1515295820751),
 Report(name=Application and Web Security #1515360776422)]
>>> report = Report('Application and Web Security #1515360776422')
>>> print(report.creation_time)
2018-01-07 15:32:56.422000
>>> report.export_pdf(filename='/foo/bar/a.pdf')
```

```
class smc.administration.reports.Report(name=None, **meta)
```

Bases: *Element*

Report represent a report that has been generated and that is currently stored on the SMC. These reports can be exported in multiple formats.

**property creation\_time**

When this report was generated. Using local time.

**Return type**

*datetime.datetime*

**export\_pdf(filename)**

Export the report in PDF format. Specify a path for which to save the file, including the trailing filename.

**Parameters**

**filename** (*str*) – path including filename

**Returns**

None

**export\_text(filename=None)**

Export in text format. Optionally provide a filename to save to.

**Parameters**

**filename** (*str*) – path including filename (optional)

**Returns**

None

**property period\_begin**

Period when this report was specified to start.

**Return type**

*datetime.datetime*

**property period\_end**

Period when this report was specified to end.

**Return type**

*datetime.datetime*

```
class smc.administration.reports.ReportDesign(name=None, **meta)
```

Bases: *Element*

A ReportDesign defines a report available in the SMC. This class provides access to generating these reports and exporting into a format supported by the SMC. Example of generating a report, and providing a callback once the report is complete which exports the report:

```
>>> def export_my_report(task):
...     if task.resource:
...         report = task.resource[0]
...         print("My report reference: %s" % report)
...         report.export_pdf('/Users/foo/myfile.pdf')
...
...
>>>
>>> report = ReportDesign('Application and Web Security')
>>> poller = report.generate(wait_for_finish=True)
>>> poller.add_done_callback(export_my_report)
>>> while not poller.done():
...     poller.wait(3)
...
...
My report reference: Report(name=Application and Web Security #1515375369483)
```

**generate**(*start\_time=None, end\_time=None, launch\_time=None, overriding\_duration=None, use\_elasticsearch=False, senders=None, wait\_for\_finish=False, timeout=5, \*\*kw*)

Generate the report and optionally wait for results. You can optionally add filters to the report by providing the senders argument as a list of type Element:

```
report = ReportDesign('Firewall Weekly Summary')
begin = datetime_to_ms(datetime.strptime("2018-02-03T00:00:00", "%Y-%m-%dT%H:%M:%S"))
end = datetime_to_ms(datetime.strptime("2018-02-04T00:00:00", "%Y-%m-%dT%H:%M:%S"))
report.generate(start_time=begin, end_time=end, senders=[Engine('vm')])
```

#### Parameters

- **start\_time** (*int*) – milliseconds time defining start time for report (legacy)
- **end\_time** (*int*) – milliseconds time defining end time for report(legacy)

:param long launch\_time : End of timerange in milliseconds (new) :param int overriding\_duration: period duration in seconds (new) :param bool use\_elasticsearch: if true will use elasticsearch storage :param senders: filter targets to use when generating report :type senders: list(Element) :param bool wait\_for\_finish: enable polling for results :param int timeout: timeout between polling :raises TaskRunFailed: refresh failed, possibly locked policy :rtype: TaskOperationPoller

#### property report\_files

Retrieve all reports that are currently available on the SMC.

#### Return type

*list(Report)*

**class** smc.administration.reports.**ReportOperation**(*name=None, \*\*meta*)

Bases: *Element*

A report operation is a way to launch a task to generate reports.

**classmethod** **create**(*launch\_time=None, overriding\_duration=None, report\_design=None, report\_per\_sender=None, use\_elasticsearch=False, export\_in\_txt=False, export\_in\_pdf=False, export\_in\_html=False, stored=False, email=None*)

Create a report design based on an existing template :param int launch\_time: report launch time in seconds :param int overriding\_duration: period duration in seconds (new) :param ReportDesign report\_design:

report design to base report operation on :param bool report\_per\_sender: if true to activate report per sender :param bool use\_elasticsearch: if true will use elasticsearch storage :param bool export\_in\_txt: if true will generate txt :param bool export\_in\_pdf: if true will generate pdf :param bool export\_in\_html: if true will generate html :param bool stored: if true will be stored :param str email: email address :param str name: Name of new report operation :raises CreateElementFailed: cannot create element :return: Report operation

**class** `smc.administration.reports.ReportTemplate`(*name=None*, *\*\*meta*)

Bases: *Element*

A report template represents an existing template in the SMC. Templates can be retrieved through the normal collections:

```
>>> list(ReportTemplate.objects.all())
[ReportTemplate(name=Firewall Weekly Summary),
 ReportTemplate(name=Firewall Daily Summary from Specific Firewall),
 ReportTemplate(name=Firewall Multi-Link Usage)
...]
```

Once a report template of interest is identified, you can create a ReportDesign using that template:

```
>>> template = ReportTemplate('Firewall Weekly Summary')
>>> template.create_design('myfirewallreport')
ReportDesign(name=myfirewallreport)
```

**create\_design**(*name*)

Create a report design based on an existing template.

**Parameters**

**name** (*str*) – Name of new report design

**Raises**

*CreateElementFailed* – failed to create template

**Return type**

*ReportDesign*

## 14.3.8 System

Module that controls aspects of the System itself, such as updating dynamic packages, updating engines, applying global block lists, etc.

To load the configuration for system, do:

```
>>> from smc.administration.system import System
>>> system = System()
>>> system.smc_version
'6.2.0 [10318]'
>>> system.last_activated_package
'881'
>>> for pkg in system.update_package():
...     print(pkg)
...
UpdatePackage(name=Update Package 889)
UpdatePackage(name=Update Package 888)
UpdatePackage(name=Update Package 887)
```

**class** smc.administration.system.**AdminDomain**(name=None, \*\*meta)

Administrative domain element. Domains are used to provide object based segmentation within SMC. If domains are in use, you can log in directly to a domain to modify contents within that domain.

Find all available domains:

```
>>> list(AdminDomain.objects.all())
[AdminDomain(name=Shared Domain)]
>>> admindomain_obj = AdminDomain(name=mydomain)
>>> admindomain_obj.announcement_enabled
True
>>> admindomain_obj.announcement_message
test
>>> admindomain_obj.update(announcement_enabled=False)
>>> admindomain_obj.announcement_enabled
False
```

---

**Note:** Admin Domains require and SMC license.

---

**property announcement\_enabled**

Display flag of announcement message :rtype: bool

**property announcement\_message**

Announcement message to be displayed before the login window. :rtype: str

**property category\_filter\_system**

Flag to know if we need to show system elements :@rtype: bool

**property contact\_number**

Contact Number :rtype: str

**classmethod create**(name, announcement\_enabled=False, announcement\_message=None, contact\_email=None, contact\_number=None, category\_filter\_system=True, show\_not\_categorized=True, user\_alert\_check=[], comment=None)

Create a new Admin Domain element for SMC objects.

Example:

```
>>> admindomain_obj=AdminDomain.create(name='mydomain',announcement_
↪enabled=True, announcement_message='test', comment='mycomment
↪')
>>> AdminDomain(name=mydomain)
```

**Parameters**

- **name** (*str*) – name of domain
- **announcement\_enabled** (*bool*) – Enable or disable display of announcement message
- **announcement\_message** (*str*) – Announcement message to be displayed before the login window
- **contact\_email** (*str*) – contact email
- **contact\_number** (*str*) – contact phone number

- **category\_filter\_system** (*bool*) – Flag to know if we need to show system elements. By default, true.
- **show\_not\_categorized** (*bool*) – Flag to know if we need to show not categorized. By default, true.
- **user\_alert\_check** (*list*) – The list of User alert checks.
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta

**Return type**

*AdminDomain*

**get\_active\_alerts**(*full=True*)

Available for all SMC API Versions but only for SMC Version above 7.1 (7.1 included)

Return active alerts for the requested domain

**:optional param full ( default value is true ). When set to false, juste retrieve the log key of each entry ( timestamp, component id, event id ).**

**Returns**

list of alert monitoring entries : session\_monitoring.SessionMonitoringResult

**Return type**

*SerializedIterable(Route)*

**property show\_not\_categorized**

Flag to know if we need to show not categorized. :rtype: bool

**property user\_alert\_check**

The list of User alert checks. :rtype: list(UserAlertCheck)

**class** smc.administration.system.**System**

System level operations such as SMC version, time, update packages, and updating engines

**active\_alerts\_ack\_all()**

Acknowledge all active alerts in the SMC. Only valid for SMC version >= 6.2.

**Raises**

*ActionCommandFailed* – Failure during acknowledge with reason

**Returns**

None

**bind\_license**(*element\_href*, *license\_id=None*, *\*\*kwargs*)

Bind license on element.

When license\_id is not set then automatic bind is done based on element type and license available.

Example:

```
system = System() system.bind_license(LogServer.objects.first()) # bind automatically license
system.bind_license(MgtServer.objects.first(), "123456787") # bind license based on id
```

**blacklist**(*src*, *dst*, *duration*=3600, *\*\*kw*)

Add blacklist to all defined engines. Use the cidr netmask at the end of *src* and *dst*, such as: 1.1.1.1/32, etc.

**Parameters**

- **src** – source of the entry
- **dst** – destination of blacklist entry

**Raises**

*ActionCommandFailed* – blacklist apply failed with reason

**Returns**

None

**See also:**

*smc.core.engine.Engine.blacklist*. Applying a blacklist at the system level will be a global blacklist entry versus an engine specific entry.

---

**Note:** If more advanced blacklist is required using source/destination ports and protocols (udp/tcp), use *kw* to provide these arguments. See *smc.elements.other.prepare\_blacklist()* for more details.

---

---

**Note:** This method requires SMC version < 7.0

---

since this version, “blacklist” is renamed “block\_list”

**block\_list**(*src*, *dst*, *duration*=3600, *\*\*kw*)

Add block\_list to all defined engines. Use the cidr netmask at the end of *src* and *dst*, such as: 1.1.1.1/32, etc.

**Parameters**

- **src** – source of the entry
- **dst** – destination of block list entry

**Raises**

*ActionCommandFailed* – block list apply failed with reason

**Returns**

None

**See also:**

*smc.core.engine.Engine.block\_list*. Applying a blacklist at the system level will be a global black-list entry versus an engine specific entry.

---

**Note:** If more advanced blacklist is required using source/destination ports and protocols (udp/tcp), use *kw* to provide these arguments. See *smc.elements.other.prepare\_blacklist()* for more details.

---

---

**Note:** This method requires SMC version >= 7.0

---

**block\_list\_bulk**(*block\_list*)

Add *block\_list* entries to all defined engines in bulk. For *block\_list* to work, you must also create a rule with action “Apply Blocklist”. First create your *block\_list* entries using *smc.elements.other.Blocklist* then provide the *block\_list* to this method.

:param Blocklist *block\_list* : pre-configured *block\_list* entries

---

**Note:** This method requires SMC version >= 7.0

---

**clean\_invalid\_filters**()

Remove all invalid filters when there are not referenced.

**Raises**

*ActionCommandFailed* – failed removing invalid filters

**Returns**

None

**delete\_license**(*license\_id*)

Delete a license based on license id.

**empty\_trash\_bin**()

Empty system level trash bin

**Raises**

*ActionCommandFailed* – failed removing trash

**Returns**

None

**engine\_upgrade**()

List all engine upgrade packages available

To find specific upgrades available from the returned collection, use convenience methods:

```
system = System()
upgrades = system.engine_upgrade()
upgrades.get_contains('6.2')
upgrades.get_all_contains('6.2')
```

**Parameters**

**engine\_version** – Version of engine to retrieve

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**Return type**

*SubElementCollection(EngineUpgrade)*

**engine\_upgrade\_import**(*import\_engine\_upgrade\_file*, *force\_import=False*)

Import upgrade package into SMC. Specify the fully qualified path to the upgrade package file.

**Parameters**

- **import\_engine\_upgrade\_file** (*str*) – system level path to upgrade package file
- **force\_import** (*boolean*) – force import when certificate that signed

the zip file has expired. :return: list imported EngineUpgrade

**export\_elements**(filename='export\_elements.zip', typeof='all', timeout=5, max\_tries=36, exclude\_trashed=None)

Export elements from SMC.

Valid types are: all (All Elements)|nw (Network Elements)|lips (IPS Elements)|sv (Services)|rb (Security Policies)|al (Alerts)|vpn (VPN Elements)

**Parameters**

- **type** – type of element
- **filename** – Name of file for export

**Raises**

*TaskRunFailed* – failure during export with reason

**Return type**

*DownloadTask*

**export\_ldif\_elements**(filename='export\_ldif\_elements.zip', timeout=5, max\_tries=36)

Export internal LDAP elements in LDIF format from SMC.

**Parameters**

**filename** – Name of file for export

**Raises**

*TaskRunFailed* – failure during export with reason

**Return type**

*DownloadTask*

**find\_system\_element**(element\_type, system\_key)

Search an element from its system key and its type

:param element\_type is the element type :param system\_key is the system key of the element :raises ElementNotFound if the system element cannot be found

**import\_elements**(import\_file)

Import elements into SMC. Specify the fully qualified path to the import file.

**Parameters**

**import\_file** (*str*) – system level path to file

**Raises**

ActionCommandFailed

**Returns**

None

**import\_ldif\_elements**(filename)

Import LDIF elements into SMC. Specify the fully qualified path to the import ldif file.

**Parameters**

**filename** (*str*) – LDIF file containing internal LDAP entries

**Raises**

ActionCommandFailed

**Returns**

None

**import\_new\_certificate\_authority\_certificate**(certificate)

Create new SMC CA from certificate.

**property last\_activated\_package**

Return the last activated package by id

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**license\_check\_for\_new()**

Launch the check and download of licenses on the Management Server. This task can be long so call returns immediately.

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**license\_details()**

This represents the license details for the SMC. This will include information with regards to the POL/POS, features, type, etc

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**Returns**

dictionary of key/values

**license\_fetch(*proof\_of\_serial*)**

Request a license download for the specified POS (proof of serial).

**Parameters**

**proof\_of\_serial** (*str*) – proof of serial number of license to fetch

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**license\_install(*license\_file*)**

Install a new license.

**Parameters**

**license\_file** (*str*) – fully qualified path to the license jar file.

**Raises**

ActionCommandFailed

**Returns**

None

**property licenses**

List of all engine related licenses This will provide details related to whether the license is bound, granted date, expiration date, etc.

```
>>> for license in system.licenses:
...     if license.bound_to.startswith('Management'):
...         print(license.proof_of_license)
abcd-efgh-ijkl-mnop
```

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**Return type**

*list(Licenses)*

**massive\_delete(*elements\_list*)**

Massive delete of several elements *elements\_list*: List of element as object to delete

**Example:**

```
host_1 = Host("My host 1") host_2 = Host("My host 2") host_3 = Host("My host 3") system = System()
system.massive_delete([host_1, host_2, host_3])
```

**property massive\_license\_bind**

Bind licenses on all unlicensed nodes

**property mgt\_integration\_configuration**

Retrieve the management API configuration for 3rd party integration devices.

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**references\_by\_element(*element\_href*)**

Return all references to element specified.

**Parameters**

**element\_href** (*str*) – element reference

**Returns**

list of references where element is used

**Return type**

*list(dict)*

**smc\_certificate\_authority()**

Show all Certificate Authorities on SMC. :rtype: SubElementCollection(CertificateAuthority).

**property smc\_time**

Return the SMC time as datetime object in UTC

:rtype datetime

**property smc\_version**

Return the SMC version

**system\_properties()**

List all global system properties available.

To find specific system property available from the returned collection, use convenience methods:

```
system = System() system_properties = system.system_properties() sys-
tem_properties.get_contains('enable_pci')
```

**Return type**

*SubElementCollection(SystemProperty)*

**system\_property(*system\_key*)**

Retrieve the global system property from its system key (unique id). Otherwise BaseException.

```
system = System() system_property = system.system_property(system_key=8)
```

**Return type**

SystemProperty

**unlicensed\_components()**

Return list of unlicensed element (if any)

**upcoming\_event()**

Allows to retrieve the upcoming events.

**Returns**

UpcomingEvents

**upcoming\_event\_ignore\_settings()**

Allows to retrieve the upcoming event ignore settings.

**Returns**

UpcomingEventIgnoreSettings

**upcoming\_event\_policy()**

Allows to retrieve the upcoming events.

**Returns**

UpcomingEventPolicy

**update\_package()**

Show all update packages on SMC.

To find specific updates available from the returned collection, use convenience methods:

```
system = System()
updates = system.update_package()
updates.get_contains('1027')
```

**Raises**

*ActionCommandFailed* – failure to retrieve resource

**Return type**

*SubElementCollection(UpdatePackage)*

**update\_package\_import(import\_update\_package\_file)**

Import update package into SMC. Specify the fully qualified path to the update package file.

**Parameters**

**import\_update\_package\_file** (*str*) – system level path to update package file

**Returns**

list imported UpdatePackage

**update\_system\_property(system\_key, new\_value)**

Update the global system property from its system key (unique id) with the specified value (str). If the system property does not exist a BaseException is thrown.

```
system = System() system.update_system_property(system_key=8, value="0")
```

**update\_upcoming\_event\_ignore\_settings(situations\_to\_ignore)**

Allows to change the upcoming event ignore settings for the current administrator. As a note, all upcoming events linked to the situation will be filtered.

**Parameters**

**situations\_to\_ignore** (*list*) – list of Situations to ignore

:return None

**update\_upcoming\_event\_policy**(*upcoming\_event\_policy*)

Allows to change the upcoming event policy. As a note, only super users are able to perform such operation.

**Parameters**

**upcoming\_event\_policy** – UpcomingEventsPolicy to update

:return None

**visible\_security\_group\_mapping**(*filter=None*)

Return all security groups assigned to VSS container types. This is only available on SMC >= 6.5.

**Parameters**

**filter** (*str*) – filter for searching by name

**Raises**

- **ActionCommandFailed** – element not found on this version of SMC
- **ResourceNotFound** – unsupported method on SMC < 6.5

**Returns**

dict

**visible\_virtual\_engine\_mapping**(*filter=None*)

Mappings for master engines and virtual engines

**Parameters**

**filter** (*str*) – filter to search by engine name

**Raises**

**ActionCommandFailed** – failure to retrieve resource

**Returns**

list of dict items related to master engines and virtual engine mappings

### 14.3.9 Tasks

Tasks will be fired when executing specific actions such as a policy upload, refresh, or making backups.

This module provides that ability to access task specific attributes and optionally poll for status of an operation.

An example of using a task poller when uploading an engine policy (use *wait\_for\_finish=True*):

```
engine = Engine('myfirewall')
poller = engine.upload(policy=fwpolicy, wait_for_finish=True)
while not poller.done():
    poller.wait(5)
    print("Task Progress {}".format(poller.task.progress))
print(poller.last_message())
```

**class** smc.administration.tasks.DownloadTask(*filename, task, timeout=5, max\_tries=36, \*\*kw*)

Bases: *TaskOperationPoller*

A download task handles tasks that have files associated, for example exporting an element to a specified file.

**class** smc.administration.tasks.Task(*task*)

Bases: *SubElement*

Task representation. This is generic and the format is used for any calls to SMC that return an asynchronous follower link to check the status of the task.

**Parameters**

- **last\_message** (*str*) – Last message received on this task
- **in\_progress** (*bool*) – Whether the task is in progress or finished
- **success** (*bool*) – Whether the task succeeded or not
- **follower** (*str*) – Fully qualified path to the follower link to track this task.

**abort()**

Abort existing task.

**Raises**

*ActionCommandFailed* – aborting task failed with reason

**Returns**

None

**download\_only**(*filename, timeout=5, max\_tries=36*)

This does not start task only return a Download Task. :rtype: DownloadTask(TaskOperationPoller)

**property end\_time**

Task end time in UTC datetime format

**Return type**

datetime

**get\_result()**

Get result of task. :rtype SMCResult.

**get\_task\_poller**(*\*\*kw*)

return a TaskOperationPoller for the Task.

**Return type**

*TaskOperationPoller*

**property last\_message**

the last message returned by the task

**Return type**

string

**property progress**

Percentage of completion

**Return type**

int

**property resource**

The resource/s associated with this task

**Return type**

*list(Element)*

**property result\_url**

Link to result (this task)

**Return type**

str

**property start\_time**

Task start time in UTC datetime format

**Return type**

datetime

**property success**

the task has succeed

**Return type**

boolean

**update\_status()**

Gets the current status of this task and returns a new task object.

**Raises**

*TaskRunFailed* – fail to update task status

**smc.administration.tasks.TaskHistory()**

Task history retrieves a list of tasks in an event queue.

**Returns**

list of task events

**Return type**

*list(TaskProgress)*

```
class smc.administration.tasks.TaskOperationPoller(task, timeout=5, max_tries=36,
                                                    wait_for_finish=False)
```

Bases: *object*

Task Operation Poller provides a way to poll the SMC for the status of the task operation. This is returned by functions that return a task. Typically these will be operations like refreshing policy, uploading policy, etc.

**add\_done\_callback(callback)**

Add a callback to run after the task completes. The callable must take 1 argument which will be the completed Task.

**Parameters**

**callback** – a callable that takes a single argument which will be the completed Task.

**done()**

Is the task done yet

**Return type**

*bool*

**last\_message(timeout=5)**

Wait a specified amount of time and return the last message from the task

**Return type**

*str*

**result(timeout=None)**

Return the current Task after waiting for timeout

**Return type**

*Task*

**stop()**

Stop the running task

**property task**

Access to task

**Return type**

*Task*

**wait**(*timeout=None*)

Blocking wait for task status.

**class** smc.administration.tasks.**TaskProgress**(*name=None, \*\*meta*)

Bases: *Element*

Task Progress represents a task event queue. These tasks may be completed or still running. The task event queue events can be retrieved by calling *TaskHistory()*.

**property task**

Return the task associated with this event

**Return type**

*Task*

### 14.3.10 Upcoming Event

Module representing read-only upcoming events in SMC

**class** smc.administration.upcoming\_event.**UpcomingEvent**(*data*)

Bases: *object*

Represents an upcoming event: - the date when the event will occur - the situation of the event - the impacted element - the possible resources an SMC event which will occur soon like certificate expiration, license expiration, scheduled task failure, ...

**property event\_date**

The upcoming event date: either the date when the event will occur or the upcoming event creation date.

**Return type**

*int*

**property impacted\_element**

The direct impacted element for this upcoming event.

**Return type**

*Element*

**property impacted\_resources**

The optional impacted resources for this upcoming event.

**Return type**

*list(Element)*

**property info**

Possible more info for the upcoming event.

**Return type**

*str*

**property situation**

The upcoming event situation which describes the context of the event.

**Return type***Situation***class** smc.administration.upcoming\_event.UpcomingEventIgnoreSettings(*data*)Bases: `object`

Represents the upcoming event ignore settings for the current administrator. List of inspection situation uri with possible resource uris. Note: all upcoming events linked to the situation uri will be filtered.

**property entries**

The list of inspection situation to filter.

**Return type**List(*Situation*)**class** smc.administration.upcoming\_event.UpcomingEvents(*upcoming\_events*)Bases: `object`

List of all upcoming events for this Management Server.

**class** smc.administration.upcoming\_event.UpcomingEventsPolicy(*upcoming\_event\_policy*)Bases: `object`

Represents the upcoming event policy. List of inspection situation uri with its threshold in days. Note: to be able to update it, you need to be superuser.

**class** smc.administration.upcoming\_event.UpcomingEventsPolicyEntry(*data*)Bases: `object`

Represents an entry for the upcoming event policy: inspection policy uri with its threshold in days.

**property enabled**

Flag to determinate if this situation must be considered as upcoming event. This flag will be global for the whole SMC.

**property situation**

The linked upcoming situation.

**Return type***Situation***property threshold\_in\_days**

The global threshold in days for the upcoming event. If the situation is confirmed, the event will be raised during this amount of time.

### 14.3.11 Updates

Functionality related to updating dynamic update packages and engine upgrades

**class** smc.administration.updates.PackageMixin

Manages downloads and activations of update packages and software upgrades

**activate**(*resource=None, force\_upgrade=False, timeout=3, wait\_for\_finish=False*)

Activate this package on the SMC

**Parameters**

**resource** (*list*) – node href's to activate on. Resource is only required for software upgrades

**:param query parameter 'force\_upgrade' flag to know if we need to force**  
the upgrade (for instance trusted certificate has expired)

**Parameters**

**timeout** (*int*) – timeout between queries

**Raises**

*TaskRunFailed* – failure during activation (downloading, etc)

**Return type**

*TaskOperationPoller*

**download**(*timeout=5, wait\_for\_finish=False*)

Download Package or Engine Update

**Parameters**

**timeout** (*int*) – timeout between queries

**Raises**

*TaskRunFailed* – failure during task status

**Return type**

*TaskOperationPoller*

**property release\_notes**

HTTP location of the release notes

#### 14.3.11.1 Engine Upgrade

**class** `smc.administration.updates.EngineUpgrade(**meta)`

Bases: *PackageMixin*, *SubElement*

Engine Upgrade package management

For example, to check engine upgrades and find a specific one, then download for installation:

```
system = System()
upgrades = system.engine_upgrade()
package = upgrades.get_contains('6.2')

poller = package.download(wait_for_finish=True)
while not poller.done():
    print(poller.result(3))
print("Finished download: %s" % poller.result())
package.activate()
```

**property platform**

Platform for this engine upgrade

**property release\_date**

Release date for this engine upgrade

**property version**

Engine upgrade version

### 14.3.11.2 Dynamic Update

**class** `smc.administration.updates.UpdatePackage(**meta)`

Bases: *PackageMixin*, *SubElement*

Container for managing update packages on SMC

Download and activate a package:

```
system = System()
packages = system.update_package()
dynup = packages.get_contains('1007')

poller = dynup.download(wait_for_finish=True)
while not poller.done():
    print(poller.result(3))
print("Finished download: %s" % poller.result())
package.activate()
```

**property** `activation_date`

Date this update was activated, if any

**Return type**

`str`

**property** `package_id`

ID of the package. These will increment as new versions are released.

**Return type**

`str`

**property** `release_date`

Date of release

**Return type**

`str`

**property** `state`

State of this package as string. Valid states are available, imported, active. If the package is available, you can execute a download. If the package is imported, you can activate.

**Return type**

`str`

## 14.4 SD-WAN

### 14.4.1 Other Elements

VPN Elements are used in conjunction with Policy or Route Based VPN configurations. VPN elements consist of external gateway and VPN site settings that identify 3rd party gateways to be used as a VPN termination endpoint.

There are several ways to create an external gateway configuration. A step by step process which first creates a network element to be used in a VPN site, then creates the ExternalGateway, an ExternalEndpoint for the gateway, and inserts the VPN site into the configuration:

```
Network.create(name='mynetwork', ipv4_network='172.18.1.0/24')
gw = ExternalGateway.create(name='mygw')
gw.external_endpoint.create(name='myendpoint', address='10.10.10.10')
gw.vpn_site.create(name='mysite', site_element=[Network('mynetwork')])
```

You can also use the convenience method *update\_or\_create* on the ExternalGateway to fully provision in a single step. Note that the ExternalEndpoint and VPNSite also have an *update\_or\_create* method to limit the update to those respective configurations (SMC version 6.4.x):

```
>>> from smc.elements.network import Network
>>> from smc.vpn.elements import ExternalGateway
>>> network = Network.get_or_create(name='network-172.18.1.0/24', ipv4_network='172.18.1.
↳0/24')
>>>
>>> g = ExternalGateway.update_or_create(name='newgw',
external_endpoint=[
    {'name': 'endpoint1', 'address': '1.1.1.1', 'enabled': True},
    {'name': 'endpoint2', 'address': '2.2.2.2', 'enabled': True}],
vpn_site=[{'name': 'sitea', 'site_element': [network]}])
>>> g
ExternalGateway(name=newgw)
>>> for endpoint in g.external_endpoint:
...     endpoint
...
ExternalEndpoint(name=endpoint1 (1.1.1.1))
ExternalEndpoint(name=endpoint2 (2.2.2.2))
>>> for site in g.vpn_site:
...     site, site.site_element
...
(VPNSite(name=sitea), [Network(name=network-172.18.1.0/24)])
```

In SMC version >= 6.5, you can also provide the *connection\_type\_ref* parameter when defining the external gateway:

```
.. note:: When calling `update_or_create` from the ExternalGateway, providing the
parameters for external_endpoints and vpn_site is optional.
```

#### 14.4.1.1 LinkUsageProfile

```
class smc.vpn.elements.LinkUsageProfile(name=None, **meta)
```

Bases: *Element*

```
classmethod create(name, default_link_balancing_preference=0, link_usage_exception=None)
```

Create a LinkUsageProfile. :param str name: name of Link usage profile :param int default\_link\_balancing\_preference: The link balancing preference from 0 equal balancing to 4 best link :param list link\_usage\_exception: list of link\_usage\_exception :raises CreateElementFailed: failed to create element with reason :raises ElementNotFound: specified element reference was not found :rtype: LinkUsageProfile

**property default\_link\_balancing\_preference**

default link balancing preference 0 to 4 equal balancing to best link

**property link\_usage\_exception**

List of exceptions

## 14.4.2 Profiles

### 14.4.2.1 VPNProfile

**class** `smc.vpn.elements.VPNProfile(name=None, **meta)`

Bases: *Element*

A VPN Profile is used to specify cryptography and other Policy VPN specific features. Every PolicyVPN requires a VPNProfile. The system will provide common profiles labeled as System Element that can be used without modification.

#### property capabilities

Capabilities are all boolean values that specify features or cryptography features to enable or disable on this VPN profile. To update or change these values, you can use the built in *update* with a key of 'capabilities' and dict value of attributes, i.e:

```
vpn = VPNProfile('mySecureVPN')
pprint(vpn.capabilities) # <-- show all options
vpn.update(capabilities={'sha2_for_ipsec': True, 'sha2_for_ike': True})
```

#### Return type

*dict*

**classmethod** `create(name, capabilities=None, cn_authentication_for_mobile_vpn=False, disable_anti_replay=False, disable_path_discovery=False, hybrid_authentication_for_mobile_vpn=False, keep_alive=False, preshared_key_for_mobile_vpn=False, sa_life_time=86400, sa_to_any_network_allowed=False, trust_all_cas=True, trusted_certificate_authority=[], tunnel_life_time_kbytes=0, tunnel_life_time_seconds=28800, comment=None, **kw)`

Create a VPN Profile. There are a variety of kwargs that can be and also retrieved about a VPN Profile. Keyword parameters are specified below. To access a valid keyword, use the standard dot notation.

To validate supported keyword attributes for a VPN Profile, consult the native SMC API docs for your version of SMC. You can also optionally print the element contents after retrieving the element and use *update* to modify the element.

For example:

```
vpn = VPNProfile.create(name='mySecureVPN', comment='test comment')
pprint(vars(vpn.data))
```

Then once identifying the attribute, update the relevant top level attribute values:

```
vpn.update(sa_life_time=128000, tunnel_life_time_seconds=57600)
```

#### Parameters

- **name** (*str*) – Name of profile
- **capabilities** (*dict*) – Capabilities are all boolean values that specify features or cryptography features to enable or disable on this VPN profile. To update or change these values, you can use the built in *update* with a key of 'capabilities' and dict value of attributes
- **cn\_authentication\_for\_mobile\_vpn** (*bool*) – Indicates whether CN authentication is allowed or not, for a client using a certificate authentication.

- **disable\_anti\_replay** (*bool*) – IPSEC\_DISABLE\_ANTI\_REPLAY flag value
- **disable\_path\_discovery** (*bool*) – NO\_PATH\_MTU\_DISCOVERY flag value.
- **hybrid\_authentication\_for\_mobile\_vpn** (*bool*) – Indicates whether Hybrid authentication is allowed or not, for a client using a certificate authentication.
- **keep\_alive** (*bool*) – Keep-alive option
- **preshared\_key\_for\_mobile\_vpn** (*bool*) – Indicates whether preshared key authentication is allowed or not, together with a client certificate authentication.,
- **sa\_life\_time** (*int*) – IKE Lifetime in seconds
- **sa\_to\_any\_network\_allowed** (*bool*) – IPSEC\_SA\_To\_ANY flag value.
- **trust\_all\_cas** (*bool*) – Whether to trust all certificate authorities or not
- **trusted\_certificate\_authority** (*list*) – List of trusted certificate authorities for this profile (if the trust all CAs attribute has been set to false)
- **tunnel\_life\_time\_kbytes** (*int*) – IPsec Lifetime in KBytes.
- **tunnel\_life\_time\_seconds** (*int*) – IPsec Lifetime in seconds.
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating element with reason

**Return type**

*VPNProfile*

## 14.5 Elements

Elements used for various configuration areas within SMC. Element types are made up of network, service groups and other.

### 14.5.1 Network

Module representing network elements used within the SMC

#### 14.5.1.1 Alias

```
class smc.elements.network.Alias(name=None, **meta)
```

Bases: *Element*

Aliases are adaptive objects that represent a single element having different values based on the engine applied on. There are many default aliases in SMC and new ones can also be created.

Finding aliases can be achieved by using collections or loading directly if you know the alias name:

```
>>> from smc.elements.network import Alias
>>> list(Alias.objects.all())
[Alias(name=$$ Interface ID 46.net), Alias(name=$$ Interface ID 45.net), etc]
```

Resolve an alias to a specific engine:

```
>>> alias = Alias('$ Interface ID 0.ip')
>>> alias.resolve('myfirewall')
[u'10.10.0.1']
```

Create an alias and assign values specific to an engine:

```
>>> alias = Alias.update_or_create(
    name='fooalias', engine=Layer3Firewall('vm'), translation_values=[Host('foo')])
>>> alias
Alias(name=fooalias)
```

**classmethod** `create(name, comment=None, default_values=None, alias_values=None)`

Create an alias.

#### Parameters

- **name** (*str*) – name of alias
- **comment** (*str*) – comment for this alias
- **default\_values** (*list* (*Element*)) – the default translated values
- **alias\_values** (*dict*) – the dedicated translated values for specified engines {Engine1: [Element1, Element2], Engine2: [Element3]}

#### Raises

*CreateElementFailed* – create failed with reason

#### Return type

*Alias*

**full\_resolve(engine)**

Fully resolve this Alias to a specific value (both translated\_element and resolved\_value). Specify the engine by name to find it's value.

```
alias = Alias('$ Interface ID 0.ip')
alias.full_resolve('smcpython-fw')
```

#### Parameters

**engine** (*str*) – name of engine to resolve value

#### Raises

*ElementNotFound* – if alias not found on engine

#### Returns

alias resolving values:

```
{“resolved_value”: [resolved type:str.],
 “translated_element”: [translated type:Element..]}
```

#### Return type

{resolved\_value:list str, translated\_element: list Element}

**resolve(engine)**

Resolve this Alias to a specific value. Specify the engine by name to find it's value.

```
alias = Alias('$ Interface ID 0.ip')
alias.resolve('smcpython-fw')
```

**Parameters**

**engine** (*str*) – name of engine to resolve value

**Raises**

*ElementNotFound* – if alias not found on engine

**Returns**

alias resolving values

**Return type**

*list*

**classmethod** **update\_or\_create**(*name*, *engine=None*, *translation\_values=None*, *with\_status=False*, *default\_values=None*, *alias\_values=None*)

Update or create an Alias and it's mappings.

**Parameters**

- **name** (*str*) – name of alias
- **engine** (*Engine*) – engine to modify alias translation values
- **translation\_values** (*list(str, Element)*) – translation values as elements. Can be None if you want to unset any existing values
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status
- **default\_values** (*list(Element)*) – the default translated values
- **alias\_values** (*dict*) – the dedicated translated values for specified engines {Engine1: [Element1, Element2], Engine2: [Element3]}

**Raises**

- *ElementNotFound* – specified engine or translation values are not found in the SMC
- *UpdateElementFailed* – update failed with reason
- *CreateElementFailed* – create failed with reason

**Return type**

*Element*

### 14.5.1.2 AddressRange

**class** `smc.elements.network.AddressRange`(*name=None*, *\*\*meta*)

Bases: *Element*

Class representing a IpRange object used in access rules

Create an address range element:

```
IpRange.create('myrange', '1.1.1.1-1.1.1.5')
```

Available attributes:

**Variables**

**ip\_range** (*str*) – IP range for element. In format: ‘10.10.10.1-10.10.10.10’

**classmethod create**(*name, ip\_range, comment=None*)

Create an AddressRange element

**Parameters**

- **name** (*str*) – Name of element
- **iprange** (*str*) – iprange of element
- **comment** (*str*) – comment (optional)

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*AddressRange*

**14.5.1.3 DomainName**

**class** smc.elements.network.DomainName(*name=None, \*\*meta*)

Bases: *Element*

Represents a domain name used as FQDN in policy Use this object to reference a DNS resolvable FQDN or partial domain name to be used in policy.

Create a domain based network element:

```
DomainName.create('mydomain.net')
```

**classmethod create**(*name, domain\_name\_entry=None, comment=None*)

Create domain name element

**Parameters**

- **name** (*str*) – name of domain, i.e. example.net, www.example.net
- **domain\_name\_entry** (*list(str)*) – The additional domain names.
- **comment** (*str*) – Optional comment.

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*DomainName*

**property domain\_name\_entry**

The additional domain names. :rtype: list(str)

#### 14.5.1.4 Expression

**class** `smc.elements.network.Expression`(*name=None, \*\*meta*)

Bases: [\*Element\*](#)

Expressions are used to build boolean like objects used in policy. For example, if you wanted to create an expression that negates a specific set of network elements to use in a “NOT” rule, an expression would be the element type.

For example, adding a rule that negates (network A or network B):

```
sub_expression = Expression.build_sub_expression(
    name='mytestexpression',
    ne_ref=['http://172.18.1.150:8082/6.0/elements/host/3999',
           'http://172.18.1.150:8082/6.0/elements/host/4325'],
    operator='union')

Expression.create(name='apiexpression',
                 ne_ref=[],
                 sub_expression=sub_expression)
```

---

**Note:** The sub-expression creates the json for the expression (network A or network B) and is then used as an parameter to create.

---

**static** `build_sub_expression`(*name, ne\_ref=None, operator='union'*)

Static method to build and return the proper json for a sub-expression. A sub-expression would be the grouping of network elements used as a target match. For example, (network A or network B) would be considered a sub-expression. This can be used to compound sub-expressions before calling create.

**Parameters**

- **name** (*str*) – name of sub-expression
- **ne\_ref** (*list*) – network elements references
- **operator** (*str*) – exclusion (negation), union, intersection (default: union)

**Returns**

JSON of subexpression. Use in [`create\(\)`](#) constructor

**classmethod** `create`(*name, ne\_ref=None, operator='exclusion', sub\_expression=None, comment=None*)

Create the expression

**Parameters**

- **name** (*str*) – name of expression
- **ne\_ref** (*list*) – network element references for expression
- **operator** (*str*) – ‘exclusion’ (negation), ‘union’, ‘intersection’ (default: exclusion)
- **sub\_expression** (*dict*) – sub expression used
- **comment** (*str*) – optional comment

**Raises**

[\*CreateElementFailed\*](#) – element creation failed with reason

**Returns**

instance with meta

**Return type**  
*Expression*

### 14.5.1.5 Host

**class** `smc.elements.network.Host`(*name=None, \*\*meta*)

Bases: *Element*, *IPv6Node*

Class representing a Host object used in access rules

Create a host element with ipv4:

```
Host.create(name='myhost', address='1.1.1.1',
            secondary=['1.1.1.2'],
            comment='some comment for my host')
```

Create a host element with ipv6 and secondary ipv4 address:

```
Host.create(name='mixedhost',
            ipv6_address='2001:cdba::3257:9652',
            secondary=['1.1.1.1'])
```

Available attributes:

#### Variables

- **address** (*str*) – IPv4 address for this element
- **ipv6\_address** (*str*) – IPv6 address for this host element
- **secondary** (*list*) – secondary IP addresses for this host
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.

**classmethod** `create`(*name, address=None, ipv6\_address=None, secondary=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None*)

Create the host element

#### Parameters

- **name** (*str*) – Name of element
- **address** (*str*) – ipv4 address of host object (optional if ipv6)
- **ipv6\_address** (*str*) – ipv6 address (optional if ipv4)
- **secondary** (*list*) – secondary ip addresses (optional)
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – comment (optional)

#### Raises

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*Host*

---

**Note:** Either ipv4 or ipv6 address is required

---

### 14.5.1.6 IPList

**class** `smc.elements.network.IPList`(*name=None, \*\*meta*)

Bases: *Element*

IPList represent a custom list of IP addresses, networks or ip ranges (IPv4 or IPv6). These are used in source/destination fields of a rule for policy enforcement.

---

**Note:** IPList requires SMC API version >= 6.1

---

Create an empty IPList:

```
IPList.create(name='mylist')
```

Create an IPList with initial content:

```
IPList.create(name='mylist', iplist=['1.1.1.1', '1.1.1.2', '1.2.3.4'])
```

Example of downloading the IPList in text format:

```
>>> iplist = list(IPList.objects.filter('mylist'))
>>> print(iplist)
[IPList(name=mylist)]
>>> iplist[0].download(filename='iplist.txt', as_type='txt')
```

Example of uploading an IPList as a zip file:

```
>>> iplist = list(IPList.objects.filter('mylist'))
>>> print(iplist)
[IPList(name=mylist)]
iplist[0].upload(filename='/path/to/iplist.zip')
```

Upload an IPList using json format:

```
>>> iplist = IPList('mylist')
>>> iplist.upload(json={'ip': ['4.4.4.4']}, as_type='json')
```

**classmethod** `create`(*name, iplist=None, comment=None*)

Create an IP List. It is also possible to add entries by supplying a list of IPs/networks, although this is optional. You can also use upload/download to add to the iplist.

**Parameters**

- **name** (*str*) – name of ip list
- **iplist** (*list*) – list of ipaddress

- **comment** (*str*) – optional comment

**Raises**

**CreateElementFailed** – element creation failed with reason

**Returns**

instance with meta

**Return type**

*IPList*

**download**(*filename=None, as\_type='zip'*)

Download the IPList. List format can be either zip, text or json. For large lists, it is recommended to use zip encoding. Filename is required for zip downloads.

**Parameters**

- **filename** (*str*) – Name of file to save to (required for zip)
- **as\_type** (*str*) – type of format to download in: txt,json,zip (default: zip)

**Raises**

**IOError** – problem writing to destination filename

**Returns**

None

**property iplist**

Return a list representation of this IPList. This is not a recommended function if the list is extremely large. In that case use the download function in zip format.

**Raises**

**FetchElementFailed** – Reason for retrieval failure

**Return type**

list

**classmethod update\_or\_create**(*append\_lists=True, with\_status=False, \*\*kwargs*)

Update or create an IPList.

**Parameters**

- **append\_lists** (*bool*) – append to existing IP List
- **kwargs** (*dict*) – provide at minimum the name attribute and optionally match the create constructor values

**Raises**

**FetchElementFailed** – Reason for retrieval failure

**upload**(*filename=None, json=None, as\_type='zip'*)

Upload an IPList to the SMC. The contents of the upload are not incremental to what is in the existing IPList. So if the intent is to add new entries, you should first retrieve the existing and append to the content, then upload. The only upload type that can be done without loading a file as the source is *as\_type='json'*.

**Parameters**

- **filename** (*str*) – required for zip/txt uploads
- **json** (*str*) – required for json uploads
- **as\_type** (*str*) – type of format to upload in: txt|json|zip (default)

**Raises**

- **IOError** – filename specified cannot be loaded
- **CreateElementFailed** – element creation failed with reason

**Returns**

None

### 14.5.1.7 Network

**class** `smc.elements.network.Network`(*name=None, \*\*meta*)

Bases: *Element*

Class representing a Network object used in access rules Network format should be CIDR based. It is recommended that when creating the network element, you use a naming convention that includes the network cidr in the name, such as 'network-1.1.1.0/24'. This will simplify searches later and workaroud the restriction that searches with '/' and '-' only match on the name field and not an actual attribute value.

Create an ipv4 network element:

```
Network.create('mynetwork', '2.2.2.0/24')
```

Create an ipv6 network element:

```
Network.create(name='mixednetwork', ipv6_network='fc00::/7')
```

Available attributes:

**Variables**

- **ipv4\_network** (*str*) – IPv4 network, in format: 10.10.10.0/24
- **ipv6\_network** (*str*) – IPv6 network

**classmethod** `create`(*name, ipv4\_network=None, ipv6\_network=None, broadcast=True, location\_ref=None, comment=None*)

Create the network element

**Parameters**

- **name** (*str*) – Name of element
- **ipv4\_network** (*str*) – network cidr (optional if ipv6)
- **ipv6\_network** (*str*) – network cidr (optional if ipv4)
- **broadcast** (*bool*) – Includes the broadcast address and the network address in the definition. This option is only used in the Source and Destination cells in rules. In other uses, the broadcast and network addresses are always included in the definition regardless of this option.
- **location\_ref** (*Location*) – Location of the network.
- **comment** (*str*) – comment (optional)

**Raises**

**CreateElementFailed** – element creation failed with reason

**Returns**

instance with meta

**Return type**

*Network*

---

**Note:** Either an `ipv4_network` or `ipv6_network` must be specified

---

### 14.5.1.8 Router

**class** `smc.elements.network.Router`(*name=None, \*\*meta*)

Bases: [Element](#), [IPv6Node](#)

Class representing a Router object used in access rules

Create a router element with ipv4 address:

```
Router.create('myrouter', '1.2.3.4', comment='my router comment')
```

Create a router element with ipv6 address:

```
Router.create(name='mixedhost',
              ipv6_address='2001:cdba::3257:9652')
```

Available attributes:

#### Variables

- **address** (*str*) – IPv4 address for this router
- **ipv6\_address** (*str*) – IPv6 address for this router
- **secondary** (*list*) – list of additional IP's for this router
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.

**classmethod** `create`(*name, address=None, ipv6\_address=None, secondary=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None*)

Create the router element

#### Parameters

- **name** (*str*) – Name of element
- **address** (*str*) – ip address of host object (optional if ipv6)
- **ipv6\_address** (*str*) – ipv6 address (optional if ipv4)
- **secondary** (*list*) – secondary ip address (optional)
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – comment (optional)

#### Raises

[CreateElementFailed](#) – element creation failed with reason

#### Returns

instance with meta

**Return type***Router*

---

**Note:** either ipv4 or ipv6 address is required

---

### 14.5.1.9 URLListApplication

**class** `smc.elements.network.URLListApplication(name=None, **meta)`Bases: *Element*

URL List Application represents a list of URL's (typically by domain) that allow for easy grouping for performing whitelist and blacklisting

Creating a URL List:

```
URLListApplication.create(  
    name='whitelist',  
    url_entry=['www.google.com', 'www.cnn.com'])
```

---

**Note:** URLListApplication requires SMC API version >= 6.1

---

Available attributes:

**Variables****url\_entry** (*list*) – URL entries as strings**add\_application\_port**(*application\_ports*)

Add application\_ports to this url list application.

**Parameters****application\_ports** (*list*(*ApplicationPort*)) – application ports to add**Raises***UpdateElementFailed* – failed updating the url list application**Returns**

None

**property application\_port**

A collection of ApplicationPort

**Return type***list*(*ApplicationPort*)**classmethod create**(*name*, *url\_entry*, *application\_ports=None*, *comment=None*)

Create the custom URL list

**Parameters**

- **name** (*str*) – name of url list
- **url\_entry** (*list*) – list of url's
- **application\_ports** (*list*) – list of ApplicationPort
- **comment** (*str*) – optional comment

**Raises***CreateElementFailed* – element creation failed with reason**Returns**

instance with meta

**Return type***URLListApplication***property url\_entry**

URL list :rtype: list(str)

**14.5.1.10 Zone****class** smc.elements.network.Zone(*name=None, \*\*meta*)Bases: *Element*

Class representing a zone used on physical interfaces and used in access control policy rules, typically in source and destination fields. Zones can be applied on multiple interfaces which would allow logical grouping in policy.

Create a zone:

```
Zone.create('myzone')
```

**classmethod create**(*name, comment=None*)

Create the zone element

**Parameters**

- **zone** (*str*) – name of zone
- **comment** (*str*) – optional comment

**Raises***CreateElementFailed* – element creation failed with reason**Returns**

instance with meta

**Return type***Zone***14.5.1.11 Traffic Handlers (Netlinks)**

NetLink elements are used to represent alternative routes that lead to the same destination IP addresses.

NetLinks usually represent Internet connections, but can be used for other communications links as well.

You can use a single Router if a single route is enough for routing traffic to a network through an interface or an aggregated link. If you want to create separate routes for traffic to a network through two or more interfaces, you must use NetLinks.

To use traffic handlers, you must first create the netlink type required, then add this to the engine routing node.

Creating a static netlink element:

```
StaticNetlink.create(
    name='netlink',
    gateway=Router('routerfoo'),
    network=[Network('mynetwork')],
    domain_server_address=['8.8.8.8', '8.8.4.4'],
    probe_address=['1.1.1.254'],
    comment='foobar')
```

Add the netlink to the desired routing interface:

```
engine = Engine('vm')
rnode = engine.routing.get(0) #interface 0
rnode.add_traffic_handler(
    netlink=StaticNetlink('mynetlink'),
    netlink_gw=[Router('myrtr')])
```

See also:

[`smc.core.route.Routing.add\_traffic\_handler`](#)

Creating Multilink's require that you first have StaticNetlink or DynamicNetlink elements. Once you have this created, you can create a multilink in a two step process.

First create the multilink members specifying the created netlinks. A multilink member encapsulates the creation process and collects the required information for each netlink such as ip\_range to use for source NAT (static netlink only) and the network role:

```
member = MultilinkMember.create(
    StaticNetlink('netlink1'), ip_range='1.1.1.1-1.1.1.2', netlink_role='active')

member1 = MultilinkMember.create(
    StaticNetlink('netlink2'), ip_range='2.1.1.1-2.1.1.2', netlink_role='standby')
```

Then create the multilink specifying the multilink members:

```
Multilink.create(name='internet', multilink_members=[member, member1])
```

See also:

[`Multilink`](#)

**class** `smc.elements.netlink.DynamicNetlink`(*name=None, \*\*meta*)

Bases: [`Element`](#)

A Dynamic Netlink is automatically created when an interface is using DHCP to obtain it's network address. It is also possible to manually create a dynamic netlink.

#### Variables

- **input\_speed** (*int*) – input speed in Kbps, used for ratio-based load-balancing
- **output\_speed** (*int*) – output speed in Kbps, used for ratio-based load-balancing
- **probe\_address** (*list*) – list of IP addresses to use as probing addresses to validate connectivity
- **standby\_mode\_period** (*int*) – Specifies the probe period when standby mode is used (in seconds)
- **standby\_mode\_timeout** (*int*) – probe timeout in seconds

- **active\_mode\_period** (*int*) – Specifies the probe period when active mode is used (in seconds)
- **active\_mode\_timeout** (*int*) – probe timeout in seconds
- **learn\_dns\_automatically** (*bool*) – whether to obtain the DNS server address from the DHCP lease

```
classmethod create(name, connection_type=None, input_speed=None, learn_dns_automatically=True,
                    output_speed=None, provider_name=None, probe_address=None,
                    standby_mode_period=3600, standby_mode_timeout=30, active_mode_period=5,
                    active_mode_timeout=1, comment=None)
```

Create a Dynamic Netlink.

#### Parameters

- **name** (*str*) – name of netlink Element
- **connection\_type** – default QoS connection type. By default, we put Active.
- **input\_speed** (*int*) – input speed in Kbps, used for ratio-based load-balancing
- **output\_speed** (*int*) – output speed in Kbps, used for ratio-based load-balancing
- **learn\_dns\_automatically** (*bool*) – whether to obtain DNS automatically from the DHCP interface
- **provider\_name** (*str*) – optional name to identify provider for this netlink
- **probe\_address** (*list*) – list of IP addresses to use as probing addresses to validate connectivity
- **standby\_mode\_period** (*int*) – Specifies the probe period when standby mode is used (in seconds)
- **standby\_mode\_timeout** (*int*) – probe timeout in seconds
- **active\_mode\_period** (*int*) – Specifies the probe period when active mode is used (in seconds)
- **active\_mode\_timeout** (*int*) – probe timeout in seconds

#### Raises

*CreateElementFailed* – failure to create netlink with reason

#### Return type

*DynamicNetlink*

---

**Note:** To monitor the status of the network links, you must define at least one probe IP address.

---

```
class smc.elements.netlink.LinkType(name=None, **meta)
```

Bases: *Element*

This represents the Link Type.

```
class smc.elements.netlink.Multilink(name=None, **meta)
```

Bases: *Element*

You can use Multi-Link to distribute outbound traffic between multiple network connections and to provide High Availability and load balancing for outbound traffic.

Creating a multilink requires several steps:

- Create the static netlink/s
- Create the multilink using the netlinks
- Add the multilink to an outbound NAT rule

Create the static netlink:

```
StaticNetlink.create(  
    name='isp1',  
    gateway=Router('nexthop'),      # 10.10.0.1  
    network=[Network('comcast')],  # 10.10.0.0/16  
    probe_address=['10.10.0.1'])
```

Create the multilink members based on the pre-created netlinks. A multilink member specifies the ip range to use for source NAT, the role (active/standby) and obtains the defined network from the StaticNetlink:

```
member = MultilinkMember.create(  
    StaticNetlink('netlink1'), ip_range='1.1.1.1-1.1.1.2', netlink_role='active')  
  
member1 = MultilinkMember.create(  
    StaticNetlink('netlink2'), ip_range='2.1.1.1-2.1.1.2', netlink_role='standby')
```

Create the multilink using the multilink members:

```
Multilink.create(name='internet', multilink_members=[member, member1])
```

Lastly, add a NAT rule with dynamic source nat using the multilink:

```
policy = FirewallPolicy('outbound')  
policy.fw_ipv4_nat_rules.create(  
    name='mynat',  
    sources=[Network('mynetwork')],  
    destinations='any',  
    services='any',  
    dynamic_src_nat=Multilink('internet'))
```

---

**Note:** Multi-Link is supported on Single Firewalls, Firewall Clusters, and Virtual Firewalls

---

**classmethod create**(*name*, *multilink\_members*, *multilink\_method*='rtt', *retries*=2, *timeout*=3600, *comment*=None)

Create a new multilink configuration. Multilink requires at least one netlink for operation, although 2 or more are recommended.

#### Parameters

- **name** (*str*) – name of multilink
- **multilink\_members** (*list*) – the output of calling `multilink_member()` to retrieve the proper formatting for this sub element.
- **multilink\_method** (*str*) – ‘rtt’ or ‘ratio’. If ratio is used, each netlink must have a probe IP address configured and also have input and output speed configured (default: ‘rtt’)
- **retries** (*int*) – number of keep alive retries before a destination link is considered unavailable (default: 2)

- **timeout** (*int*) – timeout between retries (default: 3600 seconds)
- **comment** (*str*) – comment for multilink (optional)

**Raises**

*CreateElementFailed* – failure to create multilink

**Return type**

*Multilink*

**classmethod** `create_with_netlinks`(*name*, *netlinks*, *\*\*kwargs*)

Create a multilink with a list of StaticNetlinks. To properly create the multilink using this method, pass a list of netlinks with the following dict structure:

```
netlinks = [{'netlink': StaticNetlink,
              'ip_range': '1.1.1.1-1.1.1.2',
              'netlink_role': 'active'}]
```

The *netlink\_role* can be either *active* or *standby*. The remaining settings are resolved from the StaticNetlink. The IP range value must be an IP range within the StaticNetlink's specified network. Use kwargs to pass any additional arguments that are supported by the *create* constructor. A full example of creating a multilink using predefined netlinks:

```
multilink = Multilink.create_with_netlinks(
    name='mynewnetlink',
    netlinks=[{'netlink': StaticNetlink('netlink1'),
               'ip_range': '1.1.1.2-1.1.1.3',
               'netlink_role': 'active'},
              {'netlink': StaticNetlink('netlink2'),
               'ip_range': '2.1.1.2-2.1.1.3',
               'netlink_role': 'standby'}])
```

**Parameters**

- **netlink** (*StaticNetlink*, *DynamicNetlink*) – StaticNetlink element
- **ip\_range** (*str*) – ip range for source NAT on this netlink
- **netlink\_role** (*str*) – the role for this netlink, *active* or *standby*

**Raises**

*CreateElementFailed* – failure to create multilink

**Return type**

*Multilink*

**property members**

Multilink members associated with this multilink. This provides a reference to the existing netlinks and their member settings.

**Return type**

*MultilinkMember*

**classmethod** `update_or_create`(*with\_status=False*, *\*\*kwargs*)

Update or create the element. If the element exists, update it using the kwargs provided if the provided kwargs after resolving differences from existing values. When comparing values, strings and ints are compared directly. If a list is provided and is a list of strings, it will be compared and updated if different. If the list contains unhashable elements, it is skipped. To handle complex comparisons, override this method on the subclass and process the comparison separately. If an element does not have a *create* classmethod,

then it is considered read-only and the request will be redirected to `get()`. Provide a `filter_key` dict key/value if you want to match the element by a specific attribute and value. If no `filter_key` is provided, the name field will be used to find the element.

```
>>> host = Host('kali')
>>> print(host.address)
12.12.12.12
>>> host = Host.update_or_create(name='kali', address='10.10.10.10')
>>> print(host, host.address)
Host(name=kali) 10.10.10.10
```

#### Parameters

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements `create` method.
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

#### Raises

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

#### Returns

element instance by type

#### Return type

*Element*

**class** `smc.elements.netlink.MultilinkMember`(*kwargs*)

Bases: `object`

A multilink member represents an netlink member used on a multilink configuration. Multilink uses netlinks to specify settings specific to a connection, network, whether it should be active or standby and optionally QoS. Use this class to create multilink members that are required for creating a Multilink element.

#### Variables

- **network** (*Network*) – network element reference specifying netlink subnet
- **netlink** (*StaticNetlink*, *DynamicNetlink*) – netlink element reference

**classmethod** `create`(*netlink*, *ip\_range=None*, *netlink\_role='active'*)

Create a multilink member. Multilink members are added to an Outbound Multilink configuration and define the ip range, static netlink to use, and the role. This element can be passed to the Multilink constructor to simplify creation of the outbound multilink.

#### Parameters

- **netlink** (*StaticNetlink*, *DynamicNetlink*) – static netlink element to use as member
- **ip\_range** (*str*) – the IP range for source NAT for this member. The IP range should be part of the defined network range used by this netlink. Not required for dynamic netlink
- **netlink\_role** (*str*) – role of this netlink, ‘active’ or ‘standby’

#### Raises

*ElementNotFound* – Specified netlink could not be found

**Return type***MultilinkMember***property ip\_range**

Specifies the IP address range for dynamic source address translation (NAT) for the internal source IP addresses on the NetLink. Can also be set.

**Return type***str***property netlink\_role**

Shows whether the Netlink is active or standby. Active - traffic is routed through the NetLink according to the method you specify in the Outbound Multi-Link element properties. Standby - traffic is only routed through the netlink if all primary (active) netlinks are unavailable.

**Return type***str*

**class** `smc.elements.netlink.StaticNetlink`(*name=None, \*\*meta*)

Bases: *Element*

A Static Netlink is applied to an interface to provide an alternate route to a destination. It is typically used when you have fixed IP interfaces versus using DHCP (use a Dynamic NetLink).

**Variables**

- **gateway** (*Router, Engine*) – gateway for this netlink. Should be the ‘next hop’ element associated with the netlink
- **network** (*list(Network)*) – list of networks associated with this netlink
- **input\_speed** (*int*) – input speed in Kbps, used for ratio-based load-balancing
- **output\_speed** (*int*) – output speed in Kbps, used for ratio-based load-balancing
- **probe\_address** (*list*) – list of IP addresses to use as probing addresses to validate connectivity
- **standby\_mode\_period** (*int*) – Specifies the probe period when standby mode is used (in seconds)
- **standby\_mode\_timeout** (*int*) – probe timeout in seconds
- **active\_mode\_period** (*int*) – Specifies the probe period when active mode is used (in seconds)
- **active\_mode\_timeout** (*int*) – probe timeout in seconds

**classmethod** `create`(*name, gateway, network, connection\_type=None, input\_speed=None, output\_speed=None, domain\_server\_address=None, provider\_name=None, probe\_address=None, standby\_mode\_period=3600, standby\_mode\_timeout=30, active\_mode\_period=5, active\_mode\_timeout=1, comment=None*)

Create a new StaticNetlink to be used as a traffic handler.

**Parameters**

- **name** (*str*) – name of netlink Element
- **gateway\_ref** (*Router, Engine*) – gateway to map this netlink to. This can be an element or str href.
- **connection\_type** (*ConnectionType, str*) – default QoS connection type. By default, we put Active.

- **ref** (*list*(*str*, *Element*)) – network/s associated with this netlink.
- **connection\_type** – the mandatory connection type from v6.5
- **input\_speed** (*int*) – input speed in Kbps, used for ratio-based load-balancing
- **output\_speed** (*int*) – output speed in Kbps, used for ratio-based load-balancing
- **domain\_server\_address** (*list*) – dns addresses for netlink. Engine DNS can override this field
- **provider\_name** (*str*) – optional name to identify provider for this netlink
- **probe\_address** (*list*) – list of IP addresses to use as probing addresses to validate connectivity
- **standby\_mode\_period** (*int*) – Specifies the probe period when standby mode is used (in seconds)
- **standby\_mode\_timeout** (*int*) – probe timeout in seconds
- **active\_mode\_period** (*int*) – Specifies the probe period when active mode is used (in seconds)
- **active\_mode\_timeout** (*int*) – probe timeout in seconds

**Raises**

- *ElementNotFound* – if using type *Element* parameters that are not found.
- *CreateElementFailed* – failure to create netlink with reason

**Return type***StaticNetlink*

---

**Note:** To monitor the status of the network links, you must define at least one probe IP address.

---

**property domain\_server\_address**

Configured DNS servers for this netlink

**Returns**

list of DNS servers; if elements are specified, they will be returned as type *Element*

**Return type***RankedDNSAddress***classmethod update\_or\_create**(*with\_status=False*, *\*\*kwargs*)

Update or create static netlink. DNS entry differences are not resolved, instead any entries provided will be the final state for this netlink. If the intent is to add/remove DNS entries you can use the *domain\_server\_address()* method to add or remove.

**Raises**

*CreateElementFailed* – failed creating element

**Returns**

element instance by type or 3-tuple if *with\_status* set

## 14.5.2 Services

Module providing service configuration and creation.

Some services may be generic services while others might provide more in depth functionality using protocol agents. A protocol agent provides layer 7 configuration capabilities specific to the protocol it defines. If a given service inherits the ProtocolAgentMixin, this service type is eligible to have a protocol agent attached.

See also:

`smc.elements.protocols`

**class** `smc.elements.service.ProtocolAgentMixin`

ProtocolAgentMixin is used by services that allow a protocol agent.

**property** `protocol_agent`

Protocol Agent for this service

**Returns**

Return the protocol agent or None if this service does not reference a protocol agent

**Return type**

`ProtocolAgent`

**property** `protocol_agent_values`

Protocol agent values are protocol specific settings configurable on a service when a protocol agent is assigned to that service. This property will return an iterable that represents each protocol specific parameter and it's value.

**Return type**

`BaseIterable(ProtocolAgentValues)`

**update\_protocol\_agent**(`protocol_agent`)

Update this service to use the specified protocol agent. After adding the protocol agent to the service you must call *update* on the element to commit.

**Parameters**

**protocol\_agent** (`str`, `ProtocolAgent`) – protocol agent element or href

**Returns**

None

### 14.5.2.1 EthernetService

**class** `smc.elements.service.EthernetService(name=None, **meta)`

Bases: `Element`

Represents an ethernet based service in SMC Ethernet service only supports adding Ethernet II frame type.

The value1 field should be the ethernet2 ethertype hex code which will be converted to decimal format.

Create an ethernet rule representing the presence of an IEEE 802.1Q tag:

```
>>> EthernetService.create(name='8021q frame', value1='0x8100')
EthernetService(name=8021q frame)
```

---

**Note:** Ethernet Services are only available as of SMC version 6.1.2

---

```
classmethod create(name, frame_type='eth2', value1=None, value2=None, protocol_agent_ref=None,
                    comment=None)
```

Create an ethernet service

#### Parameters

- **name** (*str*) – name of service
- **frame\_type** (*str*) – ethernet frame type, eth2
- **value1** (*str*) – hex code representing ethertype field
- **value2** (*str*) – Following the frame\_type value: For llc: the DSAP (destination service access point) address that the traffic uses. For snap: the type that the traffic uses.
- **protocol\_agent\_ref** (*ProtocolAgent*) – The possible Protocol linked to this service.
- **comment** (*str*) – optional comment

#### Raises

*CreateElementFailed* – failure creating element with reason

#### Returns

instance with meta

#### Return type

*EthernetService*

#### property protocol\_agent\_ref

The possible Protocol linked to this service. :rtype: ProtocolAgent

### 14.5.2.2 RPCService

```
class smc.elements.service.RPCService(name=None, **meta)
```

Bases: *Element*

Represents an RPC service element

```
classmethod create(name, program_number=None, transport=None, rpc_version=None,
                    comment=None)
```

This represents a SUN-RPC service. :param str name: name of rpc service. :param str program\_number: The programe number. Not Required. :param str transport: The transport type:

tcp: Allows the RPC message when transported over TCP. udp: Allows the RPC message when transported over UDP. both: Allows the RPC message when transported over TCP and UDP.

#### Parameters

- **rpc\_version** (*str*) – The remote program version number. If you do not enter a program version, the element matches traffic of any version.
- **comment** (*str*) – optional comment.

:rtype RPCService.

#### property program\_number

The programe number. :rtype str

#### property rpc\_version

The remote program version number :rtype str

**property transport**

The transport type. :rtype str

**14.5.2.3 ICMPService**

**class** smc.elements.service.ICMPService(*name=None, \*\*meta*)

Bases: [Element](#), ICMPServiceMixin

Represents an ICMP Service in SMC Use the RFC icmp type and code fields to set values. ICMP type is required, icmp code is optional but will make the service more specific if type codes exist.

Create an ICMP service using type 3, code 7 (Dest. Unreachable):

```
>>> ICMPService.create(name='api-icmp', icmp_type=3, icmp_code=7)
ICMPService(name=api-icmp)
```

Available attributes:

**Variables**

- **icmp\_type** (*int*) – icmp type field
- **icmp\_code** (*int*) – icmp type code

**classmethod** **create**(*name, icmp\_type, icmp\_code=None, comment=None*)

Create the ICMP service element

**Parameters**

- **name** (*str*) – name of service
- **icmp\_type** (*int*) – icmp type field
- **icmp\_code** (*int*) – icmp type code
- **comment** (*str*) – optional comment.

**Raises**

[CreateElementFailed](#) – failure creating element with reason

**Returns**

instance with meta

**Return type**

[ICMPService](#)

**14.5.2.4 ICMPIPv6Service**

**class** smc.elements.service.ICMPIPv6Service(*name=None, \*\*meta*)

Bases: [Element](#), ICMPServiceMixin

Represents an ICMPv6 Service type in SMC Set the icmp type field at minimum. At time of writing the icmp code fields were all 0.

Create an ICMPv6 service for Neighbor Advertisement Message:

```
>>> ICMPIPv6Service.create('api-Neighbor Advertisement Message', icmp_type=4, icmp_
↪code=8)
ICMPIPv6Service(name=api-Neighbor Advertisement Message)
```

Available attributes:

#### Variables

- **icmp\_type** (*int*) – ipv6 icmp type field
- **icmp\_code** (*int*) – icmpv6 type code

**classmethod** **create**(*name, icmp\_type, icmp\_code=None, comment=None*)

Create the ICMPIPv6 service element

#### Parameters

- **name** (*str*) – name of service
- **icmp\_type** (*int*) – ipv6 icmp type field
- **icmp\_code** (*int*) – icmp type code
- **comment** (*str*) – optional comment.

#### Raises

*CreateElementFailed* – failure creating element with reason

#### Returns

instance with meta

#### Return type

*ICMPIPv6Service*

### 14.5.2.5 IPService

**class** `smc.elements.service.IPService`(*name=None, \*\*meta*)

Bases: *ProtocolAgentMixin, Element*

Represents an IP-Proto service in SMC IP Service is represented by a protocol number. This will display in the SMC under Services -> IP-Proto. It may also show up in Services -> With Protocol if the protocol is tied to a Protocol Agent.

Create an IP Service for protocol 93 (AX.25):

```
>>> IPService.create('ipservice', 93)
IPService(name=ipservice)
```

Available attributes:

#### Variables

**protocol\_number** (*str*) – IP protocol number for this service

**classmethod** **create**(*name, protocol\_number, protocol\_agent=None, comment=None*)

Create the IP Service

#### Parameters

- **name** (*str*) – name of ip-service
- **protocol\_number** (*int*) – ip proto number for this service
- **protocol\_agent** (*str, ProtocolAgent*) – optional protocol agent for this service
- **comment** (*str*) – optional comment

#### Raises

*CreateElementFailed* – failure creating element with reason

**Returns**

instance with meta

**Return type***IPService***property protocol\_number**

Protocol number for this IP Service

**Return type**

int

### 14.5.2.6 TCPService

**class** smc.elements.service.TCPService(name=None, \*\*meta)Bases: *ProtocolAgentMixin*, *Element*

Represents a TCP based service in SMC TCP Service can use a range of ports or single port. If using single port, set only min\_dst\_port. If using range, set both min\_dst\_port and max\_dst\_port.

Create a TCP Service for port 5000:

```
>>> TCPService.create('tcpervice', 5000, comment='my service')
TCPService(name=tcpervice)
```

Available attributes:

**Variables**

- **min\_dst\_port** (*int*) – starting destination port for this service. If the service is a single port service, use only this field
- **max\_dst\_port** (*int*) – used in conjunction with min\_dst\_port for creating a port range service.

**classmethod create**(name, min\_dst\_port, max\_dst\_port=None, min\_src\_port=None, max\_src\_port=None, protocol\_agent=None, comment=None)

Create the TCP service

**Parameters**

- **name** (*str*) – name of tcp service
- **min\_dst\_port** (*int*) – minimum destination port value
- **max\_dst\_port** (*int*) – maximum destination port value
- **min\_src\_port** (*int*) – minimum source port value
- **max\_src\_port** (*int*) – maximum source port value
- **protocol\_agent** (*str*, *ProtocolAgent*) – optional protocol agent for this service
- **comment** (*str*) – optional comment for service

**Raises**

*CreateElementFailed* – failure creating element with reason

**Returns**

instance with meta

**Return type***TCPService*

### 14.5.2.7 UDPService

**class** `smc.elements.service.UDPService`(*name=None, \*\*meta*)

Bases: *ProtocolAgentMixin, Element*

UDP Services can use a range of ports or single port. If using single port, set only `min_dst_port`. If using range, set both `min_dst_port` and `max_dst_port`.

Create a UDP Service for port range 5000-5005:

```
>>> UDPService.create('udpservice', 5000, 5005)
UDPService(name=udpservice)
```

Available attributes:

#### Variables

- **`min_dst_port`** (*int*) – starting destination port for this service. If the service is a single port service, use only this field
- **`max_dst_port`** (*int*) – used in conjunction with `min_dst_port` for creating a port range service

**classmethod** `create`(*name, min\_dst\_port, max\_dst\_port=None, min\_src\_port=None, max\_src\_port=None, protocol\_agent=None, comment=None*)

Create the UDP Service

#### Parameters

- **`name`** (*str*) – name of udp service
- **`min_dst_port`** (*int*) – minimum destination port value
- **`max_dst_port`** (*int*) – maximum destination port value
- **`min_src_port`** (*int*) – minimum source port value
- **`max_src_port`** (*int*) – maximum source port value
- **`protocol_agent`** (*str, ProtocolAgent*) – optional protocol agent for this service
- **`comment`** (*str*) – optional comment

#### Raises

*CreateElementFailed* – failure creating element with reason

#### Returns

instance with meta

#### Return type

*UDPService*

### 14.5.2.8 URLCategory

**class** `smc.elements.service.URLCategory`(*name=None, \*\*meta*)

Bases: *Element*

Represents a URL Category for policy. URL Categories are read only. To make whitelist or blacklists, use *smc.elements.network.IPList*.

### 14.5.2.9 With Protocol

New in version 0.6.2: Requires SMC version >= 6.4.3

Protocols define elements within the SMC that are specified to Protocol Agents. Protocol Agents can be attached to specific services by type. If a service inherits the ProtocolAgentMixin, the service type is eligible to add a protocol agent.

An example of attaching a protocol agent to a generic TCP Service, in this case used as a custom HTTP service:

```
>>> TCPService.create(name='testservice', min_dst_port=8080,
    protocol_agent=ProtocolAgent('HTTP'), comment='foo')
TCPService(name=testservice)
```

You can optionally also add a protocol agent to an existing service if the service is already created:

```
>>> from smc.elements.protocols import ProtocolAgent
>>> service.update_protocol_agent(ProtocolAgent('HTTP'))
>>> service.update()
...
>>> service.protocol_agent
ProtocolAgent(name=HTTP)
```

To make modifications on an existing Protocol Agent assigned to a service, you can iterate the protocol agent values to see the available parameter settings then call update on the same collection.

For example, to set the above service to redirect to a Proxy Server (CIS redirect), you can use this logic.

First view the available protocol agent parameters:

```
>>> service = TCPService('testservice')
>>> service.protocol_agent
ProtocolAgent(name=HTTP)
...
>>> for parameter in service.protocol_agent_values:
...     parameter
...
BooleanValue(name=http_enforce_safe_search,description=Enforce SafeSearch,value=0)
ProxyServiceValue(name=redir_cis,description=Redirect to Proxy Server,proxy_server=None)
StringValue(name=http_server_stream_by_user_agent,
description=Optimized server stream fingerprinting,value=Yes)
StringValue(name=http_url_logging,description=Logging of accessed URLs,value=Yes)
```

The parameters returned all inherit from a base class template *ProtocolParameterValue*. Each returned parameter is generated dynamically based on the type of input expected for the given parameter/field type.

**Note:** The *description* field of the parameter matches what you would see in the SMC under the Protocol Parameters tab of a service using a ProtocolAgent. The *name* field is the internal name that you would use to reference the setting

when calling `protocol_agent_values.update`.

---

We want to add the CIS (ProxyServer) redirect, so update is done on the 'redir\_cis' (name) field:

```
>>> from smc.elements.servers import ProxyServer
>>> service.protocol_agent_values.update(name='redir_cis', proxy_server=ProxyServer(
↪ 'generic5'))
True
```

The update was successful, and we can now validate the parameter repr shows an assigned proxy:

```
>>> for parameter in service.protocol_agent_values:
...     parameter
...
BooleanValue(name=http_enforce_safe_search,description=Enforce SafeSearch,value=0)
ProxyServiceValue(name=redir_cis,description=Redirect to Proxy Server,
proxy_server=ProxyServer(name=generic5))
StringValue(name=http_server_stream_by_user_agent,
description=Optimized server stream fingerprinting,value=Yes)
StringValue(name=http_url_logging,description=Logging of accessed URLs,value=Yes)
```

Lastly, to commit this change to SMC, you must still call `update` on the service element:

```
service.update()
```

You can unset a ProxyServer by setting the `proxy_server` field to `None` and updating:

```
service.update_protocol_agent(None)
service.update()
```

```
class smc.elements.protocols.ProtocolAgent(name=None, **meta)
```

Bases: *Element*

Protocol Agents ensure that related connections for a service are properly grouped and evaluated by the engine, as well as assisting the engine with content filtering or network address translation tasks.

```
class smc.elements.protocols.ProtocolAgentMixin
```

Bases: *object*

ProtocolAgentMixin is used by services that allow a protocol agent.

**property protocol\_agent**

Protocol Agent for this service

**Returns**

Return the protocol agent or `None` if this service does not reference a protocol agent

**Return type**

*ProtocolAgent*

**property protocol\_agent\_values**

Protocol agent values are protocol specific settings configurable on a service when a protocol agent is assigned to that service. This property will return an iterable that represents each protocol specific parameter and it's value.

**Return type**

*BaseIterable(ProtocolAgentValues)*

**update\_protocol\_agent**(*protocol\_agent*)

Update this service to use the specified protocol agent. After adding the protocol agent to the service you must call *update* on the element to commit.

**Parameters**

**protocol\_agent** (*str*, *ProtocolAgent*) – protocol agent element or href

**Returns**

None

**class** `smc.elements.protocols.ProtocolAgentValues`(*protocol\_agent*, *values*)

Bases: *BaseIterable*

Protocol Agent Values define settings that can be set for specific protocols when a protocol agent is referenced in a service.

This is a collection of parameters that are relevant based on the protocol agent type. This is called from the service itself when a service has a protocol agent attached. An example of iterating a given service with an HTTP protocol agent attached:

```
>>> from smc.elements.service import TCPService
>>> service = TCPService('mynewservice')
>>> service.protocol_agent
ProtocolAgent(name=HTTP)
>>> for parameter in service.protocol_agent_values:
...     parameter
...
BooleanValue(name=http_enforce_safe_search,description=Enforce SafeSearch,value=0)
ProxyServiceValue(name=redir_cis,description=Redirect to Proxy Server,proxy_
↪server=None)
StringValue(name=http_server_stream_by_user_agent,
description=Optimized server stream fingerprinting,value=Yes)
StringValue(name=http_url_logging,description=Logging of accessed URLs,value=Yes)
```

Each protocol agent parameter has a name value and description. The name is an internal name representation but the description is the value you would see within the SMC for the given field.

Each parameter class is dynamically generated based on the class template `ProtocolParameterValue`. The class name indicates the type of parameter value that is expected for the given field.

**Return type**

*ProtocolParameterValue*

**get**(*parameter\_name*)

Get the parameter by it's name. This is a convenience for fetching. For example, fetch the proxy server (redir\_cis) parameter from a HTTP or HTTPS protocol agent:

```
pv = newservice.protocol_agent_values.get('redir_cis')
```

**Returns**

Return the parameter value if it exists, otherwise None

**Return type**

*ProtocolParameterValue*

**update**(*name*, *\*\*kwargs*)

Update protocol agent parameters based on the parameter name. Provide the relevant keyword pairs based

on the parameter type. When `update` is called, a boolean is returned indicating whether the field was successfully updated or not. You should check the return value and call *update* on the service to commit to SMC.

Example of updating a TCP Service using the HTTPS Protocol Agent to set an HTTPS Inspection Exception:

```
>>> service = TCPService('httpsservice')
>>> service.protocol_agent
ProtocolAgent(name=HTTPS)
>>> for parameter in service.protocol_agent_values:
...     parameter
...
ProxyServiceValue(name=redir_cis,
                  description=Redirect connections to Proxy Server,proxy_
↪server=None)
BooleanValue(name=http_enforce_safe_search,description=Enforce SafeSearch,
↪value=0)
StringValue(name=http_server_stream_by_user_agent,
            description=Optimized server stream fingerprinting,value=Yes)
StringValue(name=http_url_logging,description=Logging of accessed URLs,
↪value=Yes)
TlsInspectionPolicyValue(name=tls_policy,
                        description=HTTPS Inspection Exceptions,tls_
↪policy=None)
StringValue(name=tls_inspection,description=HTTPS decryption and inspection,
↪value=No)
...
>>> service.protocol_agent_values.
        update(name='tls_policy',
               tls_policy=HTTPSInspectionExceptions('myexceptions'))
True
```

#### Parameters

- **name** (*str*) – The name of the parameter to update
- **kwargs** (*dict*) – The keyword args to perform the update

#### Raises

- ***ElementNotFound*** – Can be thrown when an element reference was passed but the element does not exist
- ***MissingDependency*** – A dependency was missing preventing the update. This can happen when adding a ProxyServer for a protocol that isn't enabled

**class** `smc.elements.protocols.ProtocolParameterValue`

Bases: `object`

A `ProtocolParameterValue` defines a protocol agent parameter setting when a protocol agent is assigned to a service. There are multiple protocol parameter types and each protocol agent will have specific parameters depending on functionality.

Read only attributes are:

#### Variables

- **protocol\_agent** (*ProtocolAgent*) – The protocol agent for this parameter value
- **protocol\_agent\_values** (*dict*) – The protocol agent values for this setting
- **description** (*str*) – The read-only description of this setting, used in SMC
- **type** (*str*) – The value type that this parameter is expected, i.e. string, integer, etc

Mutable attributes are:

#### Variables

**value** (*str*) – The mutable value for this particular setting

#### property description

Description of this protocol parameter. The description is what will be displayed on the service properties under the Protocol Parameters tab when a Protocol Agent is assigned to a service

#### Return type

*str*

#### property name

Name of this protocol setting

#### Return type

*str*

#### property type

The type of this parameter. Can be string value, integer value, etc. The type is returned as a string representation.

#### Return type

*str*

#### property value

The value for this given protocol parameter. The return type is defined by the *type* of parameter

#### Returns

value based on *type* of parameter. Will return None if this parameter does not support the *value* key for this parameter

**class** `smc.elements.protocols.ProxyServiceValue`

Bases: *ProtocolParameterValue*

This represents a protocol parameter specific to setting a redirect to proxy setting on a service with a protocol agent.

Mutable attributes are:

#### Variables

**proxy\_server** (*str*) – The mutable value for this particular setting. Represents the ProxyServer element

#### property proxy\_server

The ProxyServer element referenced in this protocol parameter, if any.

#### Returns

The proxy server element or None if one is not assigned

#### Return type

*ProxyServer*

**class** `smc.elements.protocols.TlsInspectionPolicyValue`

Bases: *ProtocolParameterValue*

This represents HTTPS Inspection Exceptions that would be a parameter for a HTTPS Protocol Agent service.

Mutable attributes are:

**Variables**

**tls\_policy** (*str*) – The mutable value for this particular setting. Represents the HTTPS Inspection Exceptions element

**property** `tls_policy`

The HTTPSInspectionExceptions element referenced in this protocol agent parameter. Will be None if one is not assigned.

**Returns**

The https inspection exceptions element or None if not assigned

**Return type**

*HTTPSInspectionExceptions*

### 14.5.3 Groups

Groups that are used for element types, such as TCPServiceGroup, Group (generic), etc. All group types inherit from GroupMixin which allow for modifications of existing groups and their members.

**class** `smc.elements.group.GroupMixin`

Methods associated with handling modification of Group objects for existing elements

**empty\_members()**

Empty members from group

**Returns**

None

**property** `members`

Return members in raw href format. If you want to obtain a resolved list of elements as instance of Element, call *~obtain\_members*.

**Return type**

*list*

**obtain\_members()**

Obtain all group members from this group

**Returns**

group members as elements

**Return type**

*list(Element)*

**update\_members**(*members*, *append\_lists=False*, *remove\_members=False*, *\*\*kwargs*)

Update group members with member list. Set *append=True* to append to existing members, or *append=False* to overwrite.

**Parameters**

- **members** (*list[str, Element]*) – new members for group by href or Element
- **append\_lists** (*bool*) – whether to append

- **remove\_members** (*bool*) – remove members from the group

**Returns**

bool was modified or not

**classmethod** **update\_or\_create**(*append\_lists=True, with\_status=False, remove\_members=False, \*\*kwargs*)

Update or create group entries. If the group exists, the members will be updated. Set *append\_lists=True* to add new members to the list, or *False* to reset the list to the provided members. If setting *remove\_members*, this will override *append\_lists* if set.

**Parameters**

- **append\_lists** (*bool*) – add to existing members, if any
- **remove\_members** (*bool*) – remove specified members instead of appending or overwriting
- **kwargs** (*dict*) – keyword arguments to satisfy the *create* constructor if the group needs to be created.

**Raises**

*CreateElementFailed* – could not create element with reason

**Returns**

element instance by type

**Return type**

*Element*

### 14.5.3.1 ICMPServiceGroup

**class** `smc.elements.group.ICMPServiceGroup`(*name=None, \*\*meta*)

Bases: *GroupMixin, Element*

IP Service Group Used for storing IP Services or IP Service Groups

Available attributes:

**Variables**

**element** (*list*) – list of elements by href. Call *~obtain\_members* to retrieved the resolved list of elements.

**classmethod** **create**(*name, members=None, comment=None*)

Create the IP Service group element

**Parameters**

- **name** (*str*) – name of service group
- **element** (*list*) – IP services or IP service groups by href

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*ICMPServiceGroup*

### 14.5.3.2 IPServiceGroup

**class** smc.elements.group.**IPServiceGroup**(name=None, \*\*meta)

Bases: *GroupMixin*, *Element*

IP Service Group Used for storing IP Services or IP Service Groups

Available attributes:

**Variables**

**element** (*list*) – list of elements by href. Call *~obtain\_members* to retrieved the resolved list of elements.

**classmethod** **create**(name, members=None, comment=None)

Create the IP Service group element

**Parameters**

- **name** (*str*) – name of service group
- **element** (*list*) – IP services or IP service groups by href

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*IPServiceGroup*

### 14.5.3.3 RpcServiceGroup

**class** smc.elements.group.**RpcServiceGroup**(name=None, \*\*meta)

Bases: *GroupMixin*, *Element*

This represents a SUN-RPC Service Group, it groups a list of SUN-RPC Services.

**classmethod** **create**(name, members=None, comment=None)

Create the SUN-RPC Service Group element.

**Parameters**

- **name** (*str*) – name of rpc service group
- **members** (*list*(*str*, *Element*)) – The elements that are included in the Service Group.
- **comment** (*str*) – optional comment.

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*RpcServiceGroup*

### 14.5.3.4 Group

**class** `smc.elements.group.Group`(*name=None, \*\*meta*)

Bases: [GroupMixin](#), [Element](#)

Class representing a Group object used in access rules Groups can hold other network element types as well as other groups.

Create a group element:

```
Group.create('mygroup') #no members
```

Group with members:

```
Group.create('mygroup', [Host('kali'), Network('mynetwork')])
```

Available attributes:

#### Variables

**element** (*list*) – list of elements by href. Call `~obtain_members` to retrieve the resolved list of elements.

**classmethod** `create`(*name, members=None, comment=None, is\_monitored=False*)

Create the group

#### Parameters

- **name** (*str*) – Name of element
- **members** (*str, Element*) – group members by element names
- **comment** (*str*) – optional comment
- **is\_monitored** (*bool*) – optional option Enable or not monitoring of the group. Default: False

#### Raises

[CreateElementFailed](#) – element creation failed with reason

#### Returns

instance with meta

#### Return type

[Group](#)

### 14.5.3.5 ServiceGroup

**class** `smc.elements.group.ServiceGroup`(*name=None, \*\*meta*)

Bases: [GroupMixin](#), [Element](#)

Represents a service group in SMC. Used for grouping objects by service. Services can be “mixed” TCP/UDP/ICMP/ IPService, Protocol or other Service Groups. Element is an href to the location of the resource.

Create a TCP and UDP Service and add to ServiceGroup:

```
tcp1 = TCPService.create('api-tcp1', 5000)
udp1 = UDPService.create('api-udp1', 5001)
ServiceGroup.create('servicegroup', element=[tcp1, udp1])
```

Available attributes:

**Variables**

**element** (*list*) – list of elements by href. Call *~obtain\_members* to retrieved the resolved list of elements.

**classmethod create**(*name, members=None, comment=None*)

Create the TCP/UDP Service group element

**Parameters**

- **name** (*str*) – name of service group
- **members** (*list(str, Element)*) – elements to add by href or Element

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*ServiceGroup*

### 14.5.3.6 TCPServiceGroup

**class** smc.elements.group.TCPServiceGroup(*name=None, \*\*meta*)

Bases: *GroupMixin, Element*

Represents a TCP Service group

Create TCP Services and add to TCPServiceGroup:

```
tcp1 = TCPService.create('api-tcp1', 5000)
tcp2 = TCPService.create('api-tcp2', 5001)
ServiceGroup.create('servicegroup', element=[tcp1, tcp2])
```

Available attributes:

**Variables**

**element** (*list*) – list of elements by href. Call *~obtain\_members* to retrieved the resolved list of elements.

**classmethod create**(*name, members=None, comment=None*)

Create the TCP Service group

**Parameters**

- **name** (*str*) – name of tcp service group
- **element** (*list(str, Element)*) – tcp services by element or href

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*TCPServiceGroup*

### 14.5.3.7 UDPServiceGroup

**class** `smc.elements.group.UDPServiceGroup`(*name=None, \*\*meta*)

Bases: [GroupMixin](#), [Element](#)

UDP Service Group Used for storing UDP Services or UDP Service Groups.

Create two UDP Services and add to UDP service group:

```
udp1 = UDPService.create('udp-svc1', 5000)
udp2 = UDPService.create('udp-svc2', 5001)
UDPServiceGroup.create('udpsvcgroup', element=[udp1, udp2])
```

Available attributes:

#### Variables

**element** (*list*) – list of elements by href. Call *~obtain\_members* to retrieved the resolved list of elements.

**classmethod** `create`(*name, members=None, comment=None*)

Create the UDP Service group

#### Parameters

- **name** (*str*) – name of service group
- **element** (*list*) – UDP services or service group by reference

#### Raises

[CreateElementFailed](#) – element creation failed with reason

#### Returns

instance with meta

#### Return type

[UDPServiceGroup](#)

### 14.5.3.8 URLCategoryGroup

**class** `smc.elements.group.URLCategoryGroup`(*name=None, \*\*meta*)

Bases: [Element](#)

### 14.5.3.9 EthernetServiceGroup

**class** `smc.elements.group.EthernetServiceGroup`(*name=None, \*\*meta*)

Bases: [GroupMixin](#), [Element](#)

This represents an Ethernet Service Group, it groups a list of Ethernet Services.

Create EthernetService and add to EthernetServiceGroup:

```
es1 = EthernetService.create('EthernetService1', frame_type="eth2",
                             protocol_agent_ref=protocol, value1='1',
                             value2='2', comment=COMMENT)
es2 = EthernetService.create('EthernetService2', frame_type="eth2",
                             protocol_agent_ref=protocol, value1='1',
                             value2='2', comment=COMMENT)
EthernetServiceGroup.create('ethernetservicegroup', element=[es1, es2])
```

**classmethod** `create(name, members=None, comment=None)`

Create the Ethernet Service Group

**Parameters**

- **name** (*str*) – name of ethernet service group.
- **members** (*list*(`EthernetService`)) – The elements that are included in the Ethernet-ServiceGroup.
- **comment** (*str*) – Optional comment.

**Raises**

`CreateElementFailed` – element creation failed with reason.

**Returns**

instance with meta

**Return type**

`EthernetServiceGroup`

## 14.5.4 Servers

Module that represents server based configurations

**class** `smc.elements.servers.AuthenticationServerMixin(name=None, **meta)`

This mixin class provide interface to a TACACS+/RADIUS Authentication Server.

**classmethod** `create(name, address=None, ipv6_address=None, clear_text=False, location_ref=None, provided_method=None, retries=2, timeout=10, secondary=None, shared_secret=None, port=49, third_party_monitoring=None, tools_profile_ref=None, comment=None)`

**Parameters**

- **name** (*str*) – Name of the TacacsServer/RadiusServer.
- **address** (*str*) – Single valid IPv4 address. Required.
- **ipv6\_address** (*str*) – Single valid IPv6 address
- **clear\_text** (*bool*) – Select Accepted by Firewall if you want the Firewall to accept un-encrypted replies from the TACACS+ authentication server. Not Required.
- **location\_ref** (`Location`) – The location of TacacsServer/RadiusServer.
- **provided\_method** (*list*(`AuthenticationMethod`)) – Specify provided Authentication methods. Not Required.
- **retries** (*int*) – Specify the number of times Firewalls try to connect to the RADIUS or TACACS+ authentication server if the connection fails. Required.
- **timeout** (*int*) – Specify the time (in seconds) that Firewalls wait for the RADIUS or TACACS+ authentication server to reply. Not Required.
- **secondary** (*list*(*str*)) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one)
- **shared\_secret** (*str*) – Specify the Shared Secret that you have defined for RADIUS clients on the Active Directory server.

- **port** (*int*) – Specify the port number if the server communicates on a port other than the default port. The predefined Firewall Template allows the engines to connect to the default port. If you change to a custom port, you must add a new IPv4 Access Rule to allow the traffic. Not Required.
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*tools\_profile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – optional comment.

**property port**

Specify the port number if the server communicates on a port other than the default port. :rtype int

**property provided\_method**

Provided Authentication methods. :rtype list(AuthenticationMethod)

**property shared\_secret**

Shared secrete text :rtype str

**property timeout**

**Specify the time (in seconds) that Firewalls wait for the TACACS+/RADIUS authentication server to reply**

:rtype Time out value

**class smc.elements.servers.ContactAddressMixin**

Mixin class to provide an interface to contact addresses on the management and log server. Contact addresses on servers can contain multiple IP's for a single location.

**add\_contact\_address**(*contact\_address, location*)

Add a contact address to the Log Server:

```
server = LogServer('LogServer 172.18.1.25')
server.add_contact_address('44.44.44.4', 'ARmoteLocation')
```

**Parameters**

- **contact\_address** (*str*) – IP address used as contact address
- **location** (*str*) – Name of location to use, will be created if it doesn't exist

**Raises**

*ModificationFailed* – failed adding contact address

**Returns**

None

**property contact\_addresses**

Provides a reference to contact addresses used by this server.

Obtain a reference to manipulate or iterate existing contact addresses:

```
>>> from smc.elements.servers import ManagementServer
>>> mgt_server = ManagementServer.objects.first()
>>> for contact_address in mgt_server.contact_addresses:
```

(continues on next page)

(continued from previous page)

```
...    contact_address
...
ContactAddress(location=Default,addresses=[u'1.1.1.1'])
ContactAddress(location=foolocation,addresses=[u'12.12.12.12'])
```

**Return type***MultiContactAddress***remove\_contact\_address(location)**

Remove contact address by name of location. You can obtain all contact addresses by calling *contact\_addresses()*.

**Parameters**

**location** (*str*) – str name of location, will be created if it doesn't exist

**Raises**

*ModificationFailed* – failed removing contact address

**Returns**

None

**class** smc.elements.servers.**DataContext**(*name=None, \*\*meta*)

This represents the Data Context.

**class** smc.elements.servers.**ManagementLogServerMixin**

**add\_netflow\_collector(netflow\_collectors)**

Add netflow collector/s to this log server.

**Parameters**

**netflow\_collectors** (*list(netflow\_collectors)*) – netflow\_collector/s to add to log server

**Raises**

*UpdateElementFailed* – failed updating log server

**Returns**

None

**property elasticsearch\_authentication\_settings**

Elasticsearch authentication settings.

**property es\_tls\_settings**

Elasticsearch TLS Settings. :rtype: TlsSettings

**property external\_pki\_certificate\_settings**

Get the certificate info of this component when working with External PKI. :rtype: PkiCertificateSettings

**property forwarding\_tls\_settings**

Log Forwarding TLS Settings. :rtype: TlsSettings

**property log\_disk\_space\_handling\_mode**

Mode chosen to handle extra logs when disk runs out of space. :rtype: str

**property netflow\_collector**

A collection of NetflowCollector

**Return type**`list(NetflowCollector)DomainController`**pki\_certificate\_info()**

Get the certificate info of this component when working with External PKI. This can return None if the component does not directly have a certificate.

**Return type**`PkiCertificateInfo`**pki\_certificate\_settings()**

Get the certificate info of this component when working with External PKI.

**Return type**`PkiCertificateSettings`**pki\_delete\_certificate\_request()**

Delete the certificate request if any is defined for this component.

**pki\_export\_certificate\_request(filename=None)**

Export the certificate request for the component when working with an External PKI. This can return None if the component does not have a certificate request.

**Raises**

`CertificateExportError` – error exporting certificate

**Return type**`str` or `None`**pki\_import\_certificate(certificate)**

Import a valid certificate. Certificate can be either a file path or a string of the certificate. If string certificate, it must include the —BEGIN CERTIFICATE— string.

**Parameters**

**certificate** (`str`) – fully qualified path or string

**Raises**

- `CertificateImportError` – failure to import cert with reason
- `IOError` – file not found, permissions, etc.

**Returns**`None`**pki\_renew\_certificate()**

Start renewal process on component when using external PKI mode. It generates new private key and prepare a new certificate request.

**remove\_netflow\_collector(netflow\_collector)**

Remove a netflow collector from this log server.

**Parameters**

**netflow\_collector** (`NetflowCollector`) – element to remove

**Returns**

remove element if it exists and return bool

**Return type**`bool`

**property uuid**

Installation ID (aka UUID). :rtype: str

**class** smc.elements.servers.**MultiContactAddress**(\*\*meta)

A MultiContactAddress is a location and contact address pair which can have multiple addresses. Server elements such as Management and Log Server can have configured locations with multiple addresses per location.

Use this server reference to create, add or remove contact addresses from servers:

```
mgt_server = ManagementServer.objects.first()
mgt_server.contact_addresses.update_or_create(
    location='mylocation', addresses=['1.1.1.1', '1.1.1.2'])
```

Or remove by location:

```
mgt_server.contact_addresses.delete('mylocation')
```

**delete**(location\_name)

Remove a given location by location name. This operation is performed only if the given location is valid, and if so, *update* is called automatically.

**Parameters**

**location** (*str*) – location name or location ref

**Raises**

*UpdateElementFailed* – failed to update element with reason

**Return type**

bool

**get**(location\_name)

Get a contact address by location name

**Parameters**

**location\_name** (*str*) – name of location

**Returns**

return contact address element or None

**Return type**

*ContactAddress*

**update\_or\_create**(location, contact\_addresses, with\_status=False, overwrite\_existing=False, \*\*kw)

Update or create a contact address and location pair. If the location does not exist it will be automatically created. If the server already has a location assigned with the same name, the contact address specified will be added if it doesn't already exist (Management and Log Server can have multiple address for a single location).

**Parameters**

- **contact\_addresses** (*list*(*str*)) – list of contact addresses for the specified location
- **location** (*str*) – location to place the contact address in
- **overwrite\_existing** (*bool*) – if you want to replace existing location to address mappings set this to True. Otherwise if the location exists, only new addresses are appended
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

**Raises**

*UpdateElementFailed* – failed to update element with reason

**Return type**

*MultiContactAddress*

```
class smc.elements.servers.NetflowCollector(data_context, host, netflow_collector_port,
                                           netflow_collector_service, netflow_collector_version,
                                           filter=None, tls_profile=None, tlsIdentity=None,
                                           kafka_topic=None)
```

Represents a Netflow collector. This is a sub part of Log Server entity. Log Servers can be configured to forward log data to external hosts. You can define which type of log data you want to forward and in which format. You can also use Filters to specify in detail which log data is forwarded.

**property data\_context**

The type of log data that is forwarded. :rtype: DataContext

**property filter**

An optional local filter that defines which log data is forwarded. The local filter is only applied to the log data that matches the Log Forwarding rule. :rtype: FilterExpression

**property host**

The Host element that represents the target host to which the log data is forwarded. :rtype: Host

**property kafka\_topic**

Kafka Topic: used only in the case forwarding logs through KAFKA. :rtype: str

**property netflow\_collector\_port**

The Port that is used for log forwarding. The default port used by IPFIX/NetFlow data collectors is 2055.  
**Note!** If you have to define an Access rule that allows traffic to the target host, make sure that the Port you select is also used as the Port in the Access rule. :rtype: int

**property netflow\_collector\_service**

The network protocol for forwarding the log data (udp/tcp/tcp\_with\_tls).  
**Note!** If you have to define an Access rule that allows traffic to the target host, make sure that the Service you specify is also used as the Service in the Access rule. :rtype: str

**property netflow\_collector\_version**

The format for forwarding the log data: 

- cef*: Logs are forwarded in CEF format.
- csv*: Logs are forwarded in CSV format.
- leef*: Logs are forwarded in LEEF format.
- netflow\_v11*: Logs are forwarded in NetFlow format.

The supported version is NetFlow v16.

- ipfix*: Logs are forwarded in IPFIX (NetFlow v16) format.
- xml*: Logs are forwarded in XML format.
- esm*: Logs are forwarded in McAfee ESM format.

**Only csv, xml and esm are supported for Audit Forwarding from Mgt Server** :rtype: str

**property tlsIdentity**

Field/value pair used to insure server identity when connecting to Sys Log server using TLS *Optional* If not provided, server identity is not checked This is ignored if service is not tcp\_with\_tls

**Return type****property tls\_profile**

TLS information required to establish TLS connection to SysLog servers *Mandatory* when service is “tcp\_with\_tls” :rtype: TLSProfile

```
class smc.elements.servers.TlsSettings(data)
```

```
    classmethod create(use_internal_credentials=None, tls_credentials=None)
```

TLS credentials used for Elasticsearch/Log Forwarding If useInternalCredentials is set to FALSE, this attribute will be read. If this field is set to null, no authentication will be requested. :param bool use\_internal\_credentials: Indicate if we need to use the server's internal TLS

credentials for Elasticsearch/Log Forwarding TRUE if we want to use internal credentials FALSE if we want to use the tlsServerCredentials attribute.

#### Parameters

**tls\_credentials** ([TLSServerCredential](#)) – TLS credentials used for Elasticsearch/Log Forwarding If useInternalCredentials is set to FALSE, this attribute will be read. If this field is set to null, no authentication will be requested.

```
class smc.elements.servers.WebApp(data)
```

Represents a Web Application parameter.

```
    classmethod create(enabled=False, eca_enabled=False, listening_address=None, log_access=False,
                        port=8083, server_credentials_ref=None, session_timeout=300,
                        ssl_session_id=False, tls_cipher_suites=None, standalone_enabled=False,
                        web_app_identifier=None, host_name=None)
```

#### Parameters

- **enabled** ([bool](#)) – If the Web Application is enabled or not.
- **eca\_enabled** ([bool](#)) – Indicates if ECA Rollout evaluation is enabled in SMC Download pages (SMC Downloads has to be enabled )
- **listening\_address** ([str](#)) – The Web Application listening address, null if listening on all addresses.
- **log\_access** ([bool](#)) – Indicates whether access to this Web Application are logged.
- **port** ([int](#)) – The port on which the Web Application listens for connections.
- **server\_credentials\_ref** ([TLSServerCredential](#)) – TLS Credentials of server.
- **session\_timeout** ([int](#)) – Session Timeout for Webswing.
- **ssl\_session\_id** ([bool](#)) – Indicates whether session ID must be used with SSL.
- **tls\_cipher\_suites** ([TLSCryptographySuite](#)) – TLS Cipher suite.
- **standalone\_enabled** ([bool](#)) – Indicates if Standalone client bundles download are enabled in SMC Download pages (SMC Downloads has to be enabled )
- **web\_app\_identifier** ([str](#)) – The Web Application Identifier.
- **host\_name** ([str](#)) – The Web Application host name, null if none.

### 14.5.4.1 LogServer

**class** `smc.elements.servers.LogServer`(*name=None, \*\*meta*)

Bases: [ContactAddressMixin](#), [Element](#), [ManagementLogServerMixin](#)

Log Server elements are used to receive log data from the security engines. Most settings on Log Server generally do not need to be changed, however it may be useful to set a contact address location and IP mapping if the Log Server needs to be reachable from an engine across NAT.

It's easiest to get the log server reference through a collection:

```
>>> LogServer.objects.first()
LogServer(name=LogServer 172.18.1.150)
```

**property** `backup_log_server`

Several backup Log Servers. :rtype list(LogServer)

**property** `channel_port`

Log Server's TCP Port Number. :rtype int

**classmethod** `create`(*name, address=None, ipv6\_address=None, location\_ref=None, external\_pki\_certificate\_settings=None, uuid='N/A', tools\_profile\_ref=None, secondary=None, es\_tls\_settings=None, elasticsearch\_authentication\_settings=None, forwarding\_tls\_settings=None, netflow\_collector=None, log\_disk\_space\_handling\_mode=None, backup\_log\_server=None, channel\_port=3020, inactive=False, comment=None*)

Log Server elements are used to receive log data from the security engines. Most settings on Log Server generally do not need to be changed, however it may be useful to set a contact address location and IP mapping if the Log Server needs to be reachable from an engine across NAT. :param str name: Name of the Log Server. :param str address: Single valid IPv4 address. Required. :param str ipv6\_address: IPv6 address. :param Location location\_ref: Location of the server. :param PkiCertificateSettings external\_pki\_certificate\_settings: Certificate Settings for

External PKI mode.

#### Parameters

- **uuid** (*str*) – Mgt Server Installation ID (aka UUID).
- **tools\_profile\_ref** (*tools\_profile*) – Allows you to add commands to the element's right-click menu.
- **secondary** (*list(str)*) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one).
- **es\_tls\_settings** (*TlsSettings*) – Elasticsearch TLS Settings. Not null when we decide to override the ES TLS Settings in the Log Server or Management Server.
- **elasticsearch\_authentication\_settings** (*dict*) – if we want to override defined authentication setting of Elasticsearch Server (> 7.1) dict : method: None, basic, api\_key, certificate  
api\_key : api\_key certificate : es\_tls\_settings aside basic : login, password
- **forwarding\_tls\_settings** (*TlsSettings*) – Log Forwarding TLS Settings. Should be NULL if no Log Forwarding has been defined for this Log Server. Not required

- **netflow\_collector** (*list*([NetflowCollector](#))) – Log Servers can be configured to forward log data to external hosts. You can define which type of log data you want to forward and in which format. You can also use Filters to specify in detail which log data is forwarded.
- **log\_disk\_space\_handling\_mode** (*str*) – Mode chosen to handle extra logs when disk runs out of space.
- **backup\_log\_server** (*list*([LogServer](#))) – You can specify several backup Log Servers. The same Log Server can simultaneously be the main Log Server for some components and a backup Log Server for components that primarily use another Log Server. You can also set Log Servers to be backup Log Servers for each other so that whenever one goes down, the other Log Server is used. If Domain elements have been configured, a Log Server and its backup Log Server(s) must belong to the same Domain. Caution: If the log volumes are very high, make sure that the backup Log Server can handle the traffic load in fail-over situations.
- **channel\_port** (*int*) – Log Server’s TCP Port Number. We recommend using default port 3020 if possible. To use a non-standard port, manually add Access rules to allow communications using the new port from the NGFW Engines to the Log Server.
- **inactive** (*bool*) – Is excluded from Log Browsing, Reporting and Statistics.
- **comment** (*str*) – optional comment.

**property inactive**

Is excluded from Log Browsing, Reporting and Statistics. :rtype bool

**update(\*exception, \*\*kwargs)**

Update the existing element and clear the instance cache. Removing the cache will ensure subsequent calls requiring element attributes will force a new fetch to obtain the latest copy.

Calling update() with no args will assume the element has already been modified directly and the *data* cache will be used to update. You can also override the following attributes: href, etag, json and params. If json is sent, it is expected to be a complete payload to satisfy the update.

For kwargs, if attribute values are a list, you can pass ‘append\_lists=True’ to add to an existing list, otherwise overwrite the existing (default: overwrite)

**See also:**

To see different ways to utilize this method for updating, see: [Update](#).

**Parameters**

- **exception** – pass a custom exception to throw if failure
- **kwargs** – optional kwargs to update request data to server.

**Raises**

- [ModificationFailed](#) – raised if element is tagged as System element
- [UpdateElementFailed](#) – failed to update element with reason

**Returns**

href of the element modified

**Return type**

[str](#)

### 14.5.4.2 ManagementServer

**class** `smc.elements.servers.ManagementServer`(*name=None, \*\*meta*)

Bases: [ContactAddressMixin](#), [Element](#), [ManagementLogServerMixin](#)

Management Server configuration. Most configuration settings are better set through the SMC, such as HA, however this object can be used to do simple tasks such as add a contact addresses to the Management Server when a security engine needs to communicate over NAT.

It's easiest to get the management server reference through a collection:

```
>>> ManagementServer.objects.first()
ManagementServer(name=Management Server)
```

#### property `alert_server_ref`

Specify the Log Server to which you want the Mgt Server to send its logs :rtype: [LogServer](#)

**classmethod** `create`(*name, address=None, ipv6\_address=None, alert\_server=None, location\_ref=None, web\_app=None, announcement\_enabled=False, announcement\_message=None, external\_pki\_certificate\_settings=None, uuid=None, tools\_profile\_ref=None, secondary=None, updates\_check\_enabled=False, license\_update\_enabled=False, updates\_proxy\_enabled=False, updates\_proxy\_address=None, updates\_proxy\_port=82, updates\_proxy\_authentication\_enabled=False, updates\_proxy\_username=None, updates\_proxy\_password=None, db\_replication=False, tls\_profile=None, tls\_credentials=None, es\_tls\_settings=None, elasticsearch\_authentication\_settings=None, forwarding\_tls\_settings=None, netflow\_collector=None, mgt\_integration\_container=None, smtp\_server\_ref=None, sender\_address=None, sender\_name=None, snmp\_gateways=None, script\_path=None, sms\_http\_channel=None, sms\_smtp\_channel=None, sms\_script\_channel=None, radius\_method='eap-md5', tacacs\_method='mschap', comment=None*)

This represents a Management Server. A system component that stores all information about the configurations of all NGFW Engines, and other components in the Stonesoft Management Center, monitors their state, and provides access for Management Clients when administrators want to change the configurations or command the engines. The most important component in the system. :param str name: Name of the Management Server. :param str address: Single valid IPv4 address. Required. :param str ipv6\_address: IPv6 address. :param [LogServer](#) alert\_server: Specify the Log Server to which you want the Management

Server to send its logs. Required.

#### Parameters

- **location\_ref** ([Location](#)) – Location of the server.
- **web\_app** ([list](#) ([WebApp](#))) – Mgt Server can be configured to define several Web Applications like webclient, webswing.
- **announcement\_enabled** ([bool](#)) – Is announcement enabled for Mgt Server: Announcements in the Mgt Server element are shown for all users that connect to that Mgt Server. Not Required.
- **announcement\_message** ([str](#)) – The announcement message for Mgt Server.
- **external\_pki\_certificate\_settings** ([PkiCertificateSettings](#)) – Certificate Settings for External PKI mode.
- **uuid** ([str](#)) – Mgt Server Installation ID (aka UUID).

- **tools\_profile\_ref** (*tools\_profile*) – Allows you to add commands to the element’s right-click menu.
- **secondary** (*list(str)*) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one)
- **updates\_check\_enabled** (*bool*) – Are you notified of new dynamic updates?
- **license\_update\_enabled** (*bool*) – Select Generate and Install New Licenses Automatically to automatically regenerate and install the licenses required for upgrading system components to a major new release.
- **updates\_proxy\_enabled** (*bool*) – If the connection from the Management Server to the Internet servers requires a proxy server, select Use Proxy Server for HTTPS Connection.
- **updates\_proxy\_address** – If the connection from the Management Server to the Internet servers requires a proxy server, select an FQDN as proxy address.
- **updates\_proxy\_port** (*int*) – If the connection from the Management Server to the Internet servers requires a proxy server, select a port as proxy port.
- **updates\_proxy\_authentication\_enabled** (*bool*) – If the connection from the Management Server to the Internet servers requires a proxy server with authentication.
- **updates\_proxy\_username** (*str*) – If the connection from the Management Server to the Internet servers requires a proxy server with authentication, select an users’s name.
- **updates\_proxy\_password** (*str*) – If the connection from the Management Server to the Internet servers requires a proxy server with authentication, select an users’s password
- **db\_replication** (*bool*) – Disable automatic database replication.
- **tls\_profile** (*TLSPProfile*) – Select the TLS Profile to be used for admin login with Client Certificate authentication.
- **tls\_credentials** (*TLSServerCredential*) – Select the Credentials to be used for admin login with Client Certificate authentication.
- **es\_tls\_settings** (*TlsSettings*) – Elasticsearch TLS Settings. Not null when we decide to override the ES Tls Settings in the Log Server or Management Server.
- **elasticsearch\_authentication\_settings** (*dict*) – if we want to override defined authentication setting of Elasticsearch Server (> 7.1) dict : method: None, basic, api\_key, certificate  
  
api\_key : api\_key certificate : es\_tls\_settings aside basic : login, password
- **forwarding\_tls\_settings** (*TlsSettings*) – Log Forwarding TLS Settings. Should be NULL if no Log Forwarding has been defined for this Log Server. Not required
- **netflow\_collector** (*list(NetflowCollector)*) – Log Servers can be configured to forward log data to external hosts. You can define which type of log data you want to forward and in which format. You can also use Filters to specify in detail which log data is forwarded.
- **mgt\_integration\_container** (*list*) – Mgt Server can be configured to define several Management Integration instances.
- **smtp\_server\_ref** (*SmtpServer*) – The SMTP Server used for sending emails.
- **sender\_address** (*str*) – The sender email address.

- **sender\_name** (*str*) – The sender name.
- **snmp\_gateways** (*str*) – The SNMP gateway.
- **script\_path** (*str*) – The custom alert script path.
- **sms\_http\_channel** (*list*) – The SMS HTTP channels.
- **sms\_smtp\_channel** (*list*) – The SMS SMTP channels.
- **sms\_script\_channel** (*list*) – The SMS Script channels.
- **radius\_method** (*str*) – Radius Method used in authentication when using Radius Server for authenticating administrators. One of the following values: “pap”, “chap”, “mschap”, “mschap2”, “eap-md5” Default is eap-md5.

**:param str tacacs\_method:** Tacacs Method used in authentication when using Tacas Server for authenticating administrators. One of the following values: “ascii”, “pap”, “chap”, “mschap” Default is mschap.

#### Parameters

**comment** (*str*) – optional comment.

#### property db\_replication

Disable automatic database replication? :rtype: bool

#### property license\_update\_enabled

**Select Generate and Install New Licenses Automatically to automatically regenerate and** install the licenses required for upgrading system components to a major new release.

#### Return type

*bool*

#### property mgt\_integration\_container

Several Management Integration instances. :rtype: list

#### restart\_web\_access()

Restart Web Access on Mgt Server. :raises SMCOperationFailure: failed to restart SMC Web Access  
:return: None

#### property script\_path

The custom alert script path. :rtype: str

#### property sender\_address

The sender email address. :rtype: str

#### property sender\_name

The sender name. :rtype: str

#### property sms\_http\_channel

The SMS HTTP channels. :rtype: list

#### property sms\_script\_channel

The SMS Script channels. :rtype: list

#### property sms\_smtp\_channel

The SMS SMTP channels. :rtype: list

**property smtp\_server\_ref**

The SMTP Server used for sending emails. :rtype: Smtplib.SMTP

**property snmp\_gateways**

The SNMP gateway. :rtype: str

**property tls\_credentials**

Credentials to be used for admin login with Client Certificate authentication. :rtype: TLSServerCredential

**property tls\_profile**

TLS Profile to be used for admin login with Client Certificate authentication. :rtype: TLSProfile.

**update(\*exception, \*\*kwargs)**

Update the existing element and clear the instance cache. Removing the cache will ensure subsequent calls requiring element attributes will force a new fetch to obtain the latest copy.

Calling update() with no args will assume the element has already been modified directly and the *data* cache will be used to update. You can also override the following attributes: href, etag, json and params. If json is sent, it is expected to be a complete payload to satisfy the update.

For kwargs, if attribute values are a list, you can pass 'append\_lists=True' to add to an existing list, otherwise overwrite the existing (default: overwrite)

**See also:**

To see different ways to utilize this method for updating, see: [Update](#).

**Parameters**

- **exception** – pass a custom exception to throw if failure
- **kwargs** – optional kwargs to update request data to server.

**Raises**

- **ModificationFailed** – raised if element is tagged as System element
- **UpdateElementFailed** – failed to update element with reason

**Returns**

href of the element modified

**Return type**

str

**property updates\_check\_enabled**

Are you notified of new dynamic updates? :rtype: bool

**property updates\_proxy\_address**

FQDN as proxy address :rtype: str

**property updates\_proxy\_authentication\_enabled**

**If the connection from the Management Server to the Internet servers requires a proxy server with authentication.**

**Return type**

bool

**property updates\_proxy\_enabled**

If the connection from the Management Server to the Internet servers requires a proxy server, select Use Proxy Server for HTTPS Connection.

**Return type**

`bool`

**property updates\_proxy\_password**

Proxy server authentication password. :rtype: str

**property updates\_proxy\_port**

Proxy server port. :rtype: int

**property updates\_proxy\_username**

Proxy server authentication username. :rtype: str

**property web\_app**

Represents a Web Application parameter. :rtype: list(WebApp):

**14.5.4.3 DNSServer**

**class** `smc.elements.servers.DNSServer`(*name=None, \*\*meta*)

There are some cases in which you must define an External DNS Server element.

- For dynamic DNS (DDNS) updates with a Multi-Link configuration.
- If you want to use a DNS server for resolving malware signature mirrors.
- If you want to use a DNS server for resolving domain names and URL filtering categorization services on Firewalls, IPS engines, and Layer 2 Firewalls.

You can also optionally use External DNS Server elements to specify the DNS servers to which the firewall forwards DNS requests when you configure DNS relay.

**Variables**

- **time\_to\_live** (*int*) – how long a DNS entry can be cached
- **update\_interval** (*int*) – how often DNS entries can be updated

**classmethod** `create`(*name, address, time\_to\_live=20, update\_interval=10, secondary=None, comment=None*)

Create a DNS Server element.

**Parameters**

- **name** (*str*) – Name of DNS Server
- **address** (*str*) – IP address for DNS Server element
- **time\_to\_live** (*int*) – Defines how long a DNS entry can be cached before querying the DNS server again (default: 20)
- **update\_interval** (*int*) – Defines how often the DNS entries can be updated to the DNS server if the link status changes constantly (default: 10)
- **secondary** (*list*) – a secondary set of IP address for this element

**Raises**

`CreateElementFailed` – Failed to create with reason

**Return type**  
*DNSServer*

#### 14.5.4.4 HttpProxy

**class** `smc.elements.servers.HttpProxy`(*name=None, \*\*meta*)

An HTTP Proxy based element. Used in various areas of the configuration such as engine properties to define proxies for File Reputation, etc.

**property** `address`

Single valid IPv4 address. :rtype str

**classmethod** `create`(*name, address, proxy\_port=8080, username=None, password=None, secondary=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None*)

Create a new HTTP Proxy service. Proxy must define at least one primary address but can optionally also define a list of secondary addresses.

**Parameters**

- **name** (*str*) – Name of the proxy element
- **address** (*str*) – Primary address for proxy
- **proxy\_port** (*int*) – proxy port (default: 8080)
- **username** (*str*) – optional username for authentication (default: None)
- **password** (*str*) – password for username if defined (default: None)
- **secondary** (*list*) – secondary list of proxy server addresses
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – Failed to create the proxy element

**Return type**  
*HttpProxy*

**property** `http_proxy_port`

Listening proxy port. :rtype int

**property** `http_proxy_username`

Username for authentication. :rtype str

### 14.5.4.5 ProxyServer

**class** `smc.elements.servers.ProxyServer`(*name=None, \*\*meta*)

Bases: `ContactAddressMixin`, `Element`, `MultiContactServer`

A ProxyServer element is used in the firewall policy to provide the ability to send HTTP, HTTPS, FTP or SMTP traffic to a next hop proxy. There are two types of next hop proxies, 'Generic' and 'Forcepoint AP Web'.

Example of creating a configuration for a Forcepoint AP-Web proxy redirect:

```
server = ProxyServer.update_or_create(name='myproxy',
    address='1.1.1.1', proxy_service='forcepoint_ap-web_cloud',
    fp_proxy_key='mypassword', fp_proxy_key_id=3, fp_proxy_user_id=1234,
    inspected_service=[{'service_type': 'HTTP', 'port': '80'}])
```

Create a Generic Proxy forward service:

```
server = ProxyServer.update_or_create(name='generic', address='1.1.1.1,1.1.1.2',
    inspected_service=[{'service_type': 'HTTP', 'port': 80},
        {'service_type': 'HTTPS', 'port': 8080}])
```

Inspected services take a list of keys *service\_type* and *port*. Service type key values are 'HTTP', 'HTTPS', 'FTP' and 'SMTP'. Port value is the port for the respective protocol.

#### Parameters

**http\_proxy** (*str*) – type of proxy configuration, either generic or forcepoint\_ap-web\_cloud

#### property `add_x_forwarded_for`

Add X-Forwarded-For header when using the Generic Proxy forwarding method (default: False) :rtype: bool

#### property `balancing_mode`

**How to balance traffic, valid options are ha (first available server), src, dst, srcdst**  
(default: ha)

#### Return type

*str*

**classmethod** `create`(*name, address, secondary=None, ipv6\_address=None, balancing\_mode='ha', proxy\_service='generic', location=None, add\_x\_forwarded\_for=False, trust\_host\_header=False, inspected\_service=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None, \*\*kw*)

Create a Proxy Server element

#### Parameters

- **name** (*str*) – name of proxy server element
- **address** (*str*) – address of element. Can be a single FQDN or comma separated list of IP addresses
- **secondary** (*list*) – List of secondary IP addresses
- **ipv6\_address** (*str*) – Single valid IPv6 address.
- **balancing\_mode** (*str*) – how to balance traffic, valid options are ha (first available server), src, dst, srcdst (default: ha)

- **proxy\_service** (*str*) – which proxy service to use for next hop, options are generic or forcepoint\_ap-web\_cloud
- **location** (*Location*) – location for this proxy server
- **add\_x\_forwarded\_for** (*bool*) – add X-Forwarded-For header when using the Generic Proxy forwarding method (default: False)
- **trust\_host\_header** (*bool*) – trust the host header when using the Generic Proxy forwarding method (default: False)
- **inspected\_service** (*dict*) – inspection services dict. Valid keys are service\_type and port. Service type valid values are HTTP, HTTPS, FTP or SMTP and are case sensitive
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – optional comment
- **kw** – keyword arguments are used to collect settings when the proxy\_service value is forcepoint\_ap-web\_cloud. Valid keys are *fp\_proxy\_key*, *fp\_proxy\_key\_id*, *fp\_proxy\_user\_id*. The *fp\_proxy\_key* is the password value. All other values are of type int

**property fp\_proxy\_key**

Password of Customer ID used in HTTP/HTTPS properties. :rtype: str

**property fp\_proxy\_key\_id**

Key ID in case of Web-Gateway choice :rtype: int

**property fp\_proxy\_user\_id**

Customer ID used in HTTP/HTTPS properties. :rtype: str

**property inspected\_services**

The specified services for inspection. An inspected service is a reference to a protocol that can be forwarded for inspection, such as HTTP, HTTPS, FTP and SMTP.

**Return type**

*list*(*InspectedService*)

**property ip\_address**

**List of IP Addresses to be used in addition to the ‘address’ field to allow using multiple IP Addresses for the element.**

**Return type**

*list*(*str*)

**property proxy\_service**

The proxy service for this proxy server configuration

**Return type**

*str*

**property trust\_host\_header**

Trust the host header when using the Generic Proxy forwarding method (default: False) :rtype: bool

**classmethod** `update_or_create`(*with\_status=False*, *\*\*kwargs*)

Update or create the element. If the element exists, update it using the kwargs provided if the provided kwargs after resolving differences from existing values. When comparing values, strings and ints are compared directly. If a list is provided and is a list of strings, it will be compared and updated if different. If the list contains unhashable elements, it is skipped. To handle complex comparisons, override this method on the subclass and process the comparison separately. If an element does not have a *create* classmethod, then it is considered read-only and the request will be redirected to `get()`. Provide a *filter\_key* dict key/value if you want to match the element by a specific attribute and value. If no *filter\_key* is provided, the name field will be used to find the element.

```
>>> host = Host('kali')
>>> print(host.address)
12.12.12.12
>>> host = Host.update_or_create(name='kali', address='10.10.10.10')
>>> print(host, host.address)
Host(name=kali) 10.10.10.10
```

#### Parameters

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements *create* method.
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

#### Raises

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

#### Returns

element instance by type

#### Return type

*Element*

### 14.5.4.6 DHCP Server

**class** `smc.elements.servers.DHCP Server`(*name=None*, *\*\*meta*)

Bases: *Element*

A DHCP Server based element. Used in various areas to define External DHCP Server.

**classmethod** `create`(*name*, *address*, *ipv6\_address=None*, *location=None*, *comment=None*)

Create a DHCP Server element.

#### Parameters

- **name** (*str*) – Name of DHCP Server
- **address** (*str*) – IP address for DHCP Server element
- **ipv6\_address** (*str*) – IPv6 address for DHCP Server
- **location** (*str*) – Specifies the location for the server if there is a NAT device between the server and other SMC components.

- **comment** (*str*) – Comment for DHCP Server element

**Raises**

*CreateElementFailed* – Failed to create with reason

**Return type**

*DHCPServer*

#### 14.5.4.7 WebPortalServer

**class** smc.elements.servers.**WebPortalServer**(*name=None, \*\*meta*)

Bases: *Element*, *MultiContactServer*

This represents a Web Portal Server. A component of the Management Center responsible for browsing logs, Policy Snapshots and reports from a Web Browser.

**property alert\_server**

Specify the Log Server to which you want the Web Portal Server to send its logs rtype *LogServer*

**classmethod create**(*name, address=None, ipv6\_address=None, alert\_server=None, location\_ref=None, web\_app=None, announcement\_enabled=False, announcement\_message=None, external\_pki\_certificate\_settings=None, uuid=None, tools\_profile\_ref=None, secondary=None, comment=None*)

**This represents a Web Portal Server. A component of the Management Center responsible for** browsing logs, Policy Snapshots and reports from a Web Browser.

**Parameters**

- **name** (*str*) – Name of the Web Portal Server.
- **address** (*str*) – Single valid IPv4 address. Required.
- **ipv6\_address** (*str*) – IPv6 address.
- **alert\_server** (*LogServer*) – Specify the Log Server to which you want the Web Portal Server to send its logs. Required.
- **location\_ref** (*Location*) – Location of the server.
- **web\_app** (*list(WebApp)*) – Web Portal Server can be configured to define several Web

Application like webclient, webswing. :param bool *announcement\_enabled*: Is announcement enabled for WebPortal Server:Announcements

in the Web Portal Server element are shown for all users that connect to that Web Portal Server. Not Required.

**Parameters**

- **announcement\_message** (*str*) – The announcement message for WebPortal users.
- **external\_pki\_certificate\_settings** (*PkiCertificateSettings*) – Certificate Settings for External PKI mode.
- **uuid** (*str*) – Web Portal Server Installation ID (aka UUID).
- **tools\_profile\_ref** (*tools\_profile*) – Allows you to add commands to the element's right-click menu.

- **secondary** (*list(str)*) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one)
- **comment** (*str*) – optional comment.

#### property external\_pki\_certificate\_settings

Certificate Settings for External PKI mode. return PkiCertificateSettings

#### property web\_app

Represents a Web Application parameter. :rtype list(WebApp):

### 14.5.4.8 EpoServer

**class** smc.elements.servers.**EpoServer**(name=None, \*\*meta)

Bases: *Element*, *ContactAddressMixin*, *IPv6Node*

This represents an ePO server: A Network Element that represents an ePO instance of server.

**classmethod** **create**(name, address=None, ipv6\_address=None, epo\_password=None, epo\_login=None, epo\_port=8444, location\_ref=None, secondary=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None)

This represents an ePO server: A Network Element that represents an ePO instance of server. :param str name: Name of the EpoServer. :param str address: Single valid IPv4 address. Required. :param str ipv6\_address: Single valid IPv6 address. :param str epo\_password: The ePO password Must be entered in clear. If filled with stars,the

field won't be updated. Required.

#### Parameters

- **epo\_login** (*str*) – The ePO login user name.
- **epo\_port** (*int*) – The ePO port.
- **location\_ref** (*Location*) – The location of EpoServer.
- **secondary** (*list(str)*) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one)
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – optional comment.

#### property epo\_login

The ePO login user name. :rtype: str

#### property epo\_port

The ePO port. :rtype: int

#### 14.5.4.9 LDAPServer

#### 14.5.4.10 ActiveDirectoryServer

#### 14.5.4.11 SmtplibServer

**class** smc.elements.servers.SmtplibServer(*name=None, \*\*meta*)

Bases: *Element*, *ContactAddressMixin*, *MultiContactServer*

**This represents a Simple Mail Transfer Protocol (SMTP) server. Server used to process notifications by e-mails.**

**classmethod create**(*name, address=None, ipv6\_address=None, port=25, email\_sender\_address=None, email\_sender\_name=None, location\_ref=None, secondary=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None*)

This represents a Simple Mail Transfer Protocol (SMTP) server. Server used to process notifications by e-mails. :param str name: Name of the SmtplibServer. :param str address: Single valid IPv4 address. Required. :param str ipv6\_address: Single valid IPv6 address. :param int port: The port on which the SMTP server is listening. default port 25.Required :param str email\_sender\_address: E-mail address to be used in the From field of the e-mail.

This default value can be overridden in the properties of the element where the SMTP Server is used. Not Required.

##### Parameters

- **email\_sender\_name** (*str*) – Name to be used in the From field of the e-mail. This default value can be overridden in the properties of the element where the SMTP Server is used. Not Required.
- **location\_ref** (*Location*) – The location of SmtplibServer.
- **secondary** (*list(str)*) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one)
- **third\_party\_monitoring** (*ThirdPartyMonitoring*) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** (*DeviceToolsProfile*) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – optional comment.

##### property email\_sender\_address

E-mail address to be used in the From field of the e-mail. :rtype str

##### property email\_sender\_name

Name to be used in the From field of the e-mail. :rtype str

##### property port

The port on which the SMTP server is listening. :rtype int

#### 14.5.4.12 IcapServer

**class** `smc.elements.servers.IcapServer`(*name=None, \*\*meta*)

Bases: [Element](#), [MultiContactServer](#)

This represents an ICAP server: A Network Element that represents an ICAP instance of server.

**classmethod** `create`(*name, address=None, ipv6\_address=None, icap\_include\_xhdrs=False, icap\_path=None, icap\_port=1344, icap\_secure=False, icap\_xhdr\_clientip=None, icap\_xhdr\_serverip=None, icap\_xhdr\_username=None, tls\_profile\_ref=None, location\_ref=None, secondary=None, third\_party\_monitoring=None, tools\_profile\_ref=None, comment=None*)

##### Parameters

- **name** (*str*) – Name of the IcapServer.
- **address** (*str*) – Single valid IPv4 address. Required.
- **ipv6\_address** (*str*) – Single valid IPv6 address.

:param bool icap\_include\_xhdrs: Include X-Headers or not. :param str icap\_path: The path to the service. Not Required. Defaults to “reqmod”. :param int icap\_port: The port on which the ICAP server is listening. Defaults to 1344, or

11344 for Secure

##### Parameters

- **icap\_secure** (*bool*) – Secure ICAP Enabled. Not Required.
- **icap\_xhdr\_clientip** (*str*) – X-Header Client IP. Not Required. Defaults to “X-Client-IP”
- **icap\_xhdr\_serverip** (*str*) – X-Header Server IP. Not Required. Defaults to “X-Server-IP”
- **icap\_xhdr\_username** (*str*) – X-Header Username. Not Required. Defaults to “X-Authenticated-User”
- **tls\_profile\_ref** ([TLSProfile](#)) – Represents a TLS Profile Contains common parameters for establishing TLS based connections.
- **location\_ref** ([Location](#)) – The location of TacacsServer/RadiusServer.
- **secondary** (*list(str)*) – If the device has additional IP addresses, you can enter them here instead of creating additional elements for the other IP addresses. The secondary IP addresses are valid in policies and in routing and antispoofing. You can add several IPv4 and IPv6 addresses (one by one)
- **third\_party\_monitoring** ([ThirdPartyMonitoring](#)) – The optional Third Party Monitoring configuration.
- **tools\_profile\_ref** ([DeviceToolsProfile](#)) – Allows you to add commands to the element’s right-click menu. Not Required.
- **comment** (*str*) – optional comment.

**property** `icap_include_xhdrs`

Include X-Headers or not. :rtype bool

**property icap\_path**

The path to the service. Not Required. Defaults to “reqmod”.

**property icap\_port**

The port on which the ICAP server is listening. :rtype int

**property icap\_secure**

Secure ICAP Enabled. :rtype bool

**property icap\_xhdr\_clientip**

X-Header Client IP. :rtype str

**property icap\_xhdr\_serverip**

X-Header Server IP. :rtype str

**property icap\_xhdr\_username**

X-Header Username. :rtype str

**property tls\_profile\_ref**

Represents a TLS Profile Contains common parameters for establishing TLS based connections. :rtype TLSProfile

#### 14.5.4.13 RadiusServer

```
class smc.elements.servers.RadiusServer(name=None, **meta)
```

Bases: [AuthenticationServerMixin](#), [ContactAddressMixin](#)

This represents a RADIUS Server. It is an Authentication server using RADIUS authentication method. It can be used as authentication method for Administrators. An external authentication server can be any server that supports either the RADIUS or the TACACS+ protocol, including Microsoft Active Directory servers. External authentication servers are integrated with the help of Active Directory Server, RADIUS Authentication Server, TACACS+ Authentication Server, and Authentication Method elements. The RADIUS Authentication Server elements define the settings necessary for connecting to an external authentication server. The Authentication Method elements define an authentication method, and can include several RADIUS Authentication Servers or TACACS+ Authentication Servers that support the method and can be used as backups to each other.

#### 14.5.4.14 TacacsServer

```
class smc.elements.servers.TacacsServer(name=None, **meta)
```

Bases: [AuthenticationServerMixin](#), [ContactAddressMixin](#)

This represents a TACACS Server. An external authentication server can be any server that supports either the RADIUS or the TACACS+ protocol, including Microsoft Active Directory servers. External authentication servers are integrated with the help of Active Directory Server, RADIUS Authentication Server, TACACS+ Authentication Server, and Authentication Method elements. The RADIUS Authentication Server and TACACS+ Authentication Server elements define the settings necessary for connecting to an external authentication server. The Authentication Method elements define an authentication method, and can include several RADIUS Authentication Servers or TACACS+ Authentication Servers that support the method and can be used as backups to each other.

**property clear\_text**

Select Accepted by Firewall if you want the Firewall to accept unencrypted replies from the TACACS+ authentication server.

:rtype bool

#### 14.5.4.15 NTPServer

**class** `smc.elements.servers.NTPServer(name=None, **meta)`

Bases: [Element](#), [IPv6Node](#)

This represents an NTP server: A Network Element that represents an NTP instance of server.

**classmethod** `create(name, address, ipv6_address=None, ntp_host_name=None, ntp_auth_key_type='none', ntp_auth_key_id=None, ntp_auth_key=None, secondary=None, third_party_monitoring=None, tools_profile_ref=None, comment=None)`

Create NTP server

##### Parameters

- **name** (*str*) – name for the Element
- **address** (*str*) – The NTP address (Required)
- **ipv6\_address** (*str*) – Single valid IPv6 address.
- **ntp\_host\_name** (*str*) – NTP server name to use

:param str ntp\_auth\_key\_type: The NTP Authentication Key Type (Required) possible values are (none, md5, sha1, sha256) :param str ntp\_auth\_key\_id: The NTP Authentication Key ID (Not Required) value between 1 - 65534 :param str ntp\_auth\_key: The NTP Authentication Key (Not Required) :param list secondary: secondary ip address (optional) :param ThirdPartyMonitoring third\_party\_monitoring: The optional Third Party Monitoring

configuration.

##### Parameters

- **tools\_profile\_ref** ([DeviceToolsProfile](#)) – Allows you to add commands to the element's right-click menu. Not Required.
- **comment** (*str*) – comment for the element

##### Raises

[CreateElementFailed](#) – Failed to create with reason

##### Return type

[NTPServer](#)

##### property `ntp_auth_key`

The NTP Authentication Key (Not Required)

##### property `ntp_auth_key_id`

The NTP Authentication Key ID (Not Required) value between 1 - 65534

##### property `ntp_auth_key_type`

The NTP Authentication Key Type (Required) Possible values are: - none - md5 - sha1 - sha256

##### property `ntp_host_name`

The NTP Host Name (Not Required)

#### 14.5.4.16 ElasticsearchCluster

**class** `smc.elements.servers.ElasticsearchCluster`(*name=None, \*\*meta*)

Bases: [\*ContactAddressMixin\*](#), [\*Element\*](#)

An ElasticsearchCluster server type element.

**property** `addresses`

The list of hostnames / IP addresses (at least one) used to establish connection to the Elasticsearch Cluster.

**Return type**

[\*list\(str\)\*](#)

**property** `authentication_settings`

Elasticsearch authentication settings.

**classmethod** `create`(*name, addresses, port=9200, es\_retention\_period=30, es\_shard\_number=0, es\_replica\_number=0, enable\_cluster\_sniffer=False, location=None, comment=None, tls\_profile=None, es\_tls\_settings=None, authentication\_settings=None*)

Create a Elasticsearch Cluster Server element.

**Parameters**

- **name** (*str*) – Name of Elasticsearch Cluster
- **addresses** (*list, str*) – str comma-separated list or list of one or more FQDNs or IP addresses
- **port** (*int*) – Default port is 9200
- **es\_retention\_period** (*int*) – How much time logs will be kept

30days default :param int es\_shard\_number: Auto by default, number of shards :param int es\_replica\_number : number of ES replicas :param bool enable\_cluster\_sniffer : Enable cluster sniffer (False default) :param str location: Specifies the location for the server if there is a NAT device between the server and other SMC components. :param str comment: Comment for Elasticsearch cluster Server element :param str tls\_profile: tls profile name to use :param [\*TlsSettings\*](#) es\_tls\_settings: Elasticsearch TLS Settings. :param dict authentication\_settings: method: str can be none, basic, api\_key or certificate for basic

login : elasticsearch user login password : elasticsearch user password

**for api\_key:**

api\_key: elasticsearch api key for user

**Raises**

[\*CreateElementFailed\*](#) – Failed to create with reason

**Return type**

[\*ElasticsearchCluster\*](#)

**property** `es_enable_cluster_sniffer`

Enabling the cluster sniffer. Default: false. :rtype: bool

**property es\_retention\_period**

Retention period (in days) for logs in Elasticsearch cluster. :rtype: int

**property es\_shard\_number**

The number of shards for the fwlogsandalerts indices, a strictly positive number, or auto (default value).

**Return type**

int

**property es\_tls\_settings**

Elasticsearch TLS Settings. :rtype: TlsSettings

**property port**

The port on which connection is established on Elasticsearch Cluster nodes. :rtype: int

**update(\*exception, \*\*kwargs)**

Update the existing element and clear the instance cache. Removing the cache will ensure subsequent calls requiring element attributes will force a new fetch to obtain the latest copy.

Calling update() with no args will assume the element has already been modified directly and the *data* cache will be used to update. You can also override the following attributes: href, etag, json and params. If json is sent, it is expected to be a complete payload to satisfy the update.

For kwargs, if attribute values are a list, you can pass 'append\_lists=True' to add to an existing list, otherwise overwrite the existing (default: overwrite)

**See also:**

To see different ways to utilize this method for updating, see: [Update](#).

**Parameters**

- **exception** – pass a custom exception to throw if failure
- **kwargs** – optional kwargs to update request data to server.

**Raises**

- **ModificationFailed** – raised if element is tagged as System element
- **UpdateElementFailed** – failed to update element with reason

**Returns**

href of the element modified

**Return type**

str

## 14.5.5 Other

Other element types that treated more like generics, or that can be applied in different areas within the SMC. They will not independently be created as standalone objects and will be more generic container classes that define the required json when used by API functions or methods. For example, blocklist can be applied to an engine directly or system wide. This class will define the format when calling blocklist functions.

**class** smc.elements.other.Blacklist

Blacklist provides a simple container to add multiple blacklist entries. Pass an instance of this to `smc.core.engine.blacklist_bulk` to upload to the engine.

---

**Note:** This method requires SMC version < 7.0

---

since this version, “blacklist” is renamed “block\_list”

```
add_entry(src, dst, duration=3600, src_port1=None, src_port2=None, src_proto='predefined_tcp',
           dst_port1=None, dst_port2=None, dst_proto='predefined_tcp')
```

Create a blacklist entry.

A blacklist can be added directly from the engine node, or from the system context. If submitting from the system context, it becomes a global blacklist. This will return the properly formatted json to submit.

**Parameters**

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or ‘any’
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or ‘any’
- **duration** (*int*) – length of time to blacklist

Both the system and engine context blacklist allow kw to be passed to provide additional functionality such as adding source and destination ports or port ranges and specifying the protocol. The following parameters define the kw that can be passed.

The following example shows creating an engine context blacklist using additional kw:

```
engine.blacklist('1.1.1.1/32', '2.2.2.2/32', duration=3600,
                 src_port1=1000, src_port2=1500, src_proto='predefined_udp',
                 dst_port1=3, dst_port2=3000, dst_proto='predefined_udp')
```

**Parameters**

- **src\_port1** (*int*) – start source port to limit blacklist
- **src\_port2** (*int*) – end source port to limit blacklist
- **src\_proto** (*str*) – source protocol. Either ‘predefined\_tcp’ or ‘predefined\_udp’. (default: ‘predefined\_tcp’)
- **dst\_port1** (*int*) – start dst port to limit blacklist
- **dst\_port2** (*int*) – end dst port to limit blacklist
- **dst\_proto** (*str*) – dst protocol. Either ‘predefined\_tcp’ or ‘predefined\_udp’. (default: ‘predefined\_tcp’)

---

**Note:** if blocking a range of ports, use both `src_port1` and `src_port2`, otherwise providing only `src_port1` is adequate. The same applies to `dst_port1` / `dst_port2`. In addition, if you provide `src_portX` but not `dst_portX` (or vice versa), the undefined port side definition will default to all ports.

---

**class** smc.elements.other.Blocklist

Blocklist provides a simple container to add multiple `block_list` entries. Pass an instance of this to `smc.core.engine.block_list_bulk` to upload to the engine.

---

**Note:** This method requires SMC version >= 7.0

---

```
add_entry(src, dst, duration=3600, src_port1=None, src_port2=None, src_proto='predefined_tcp',
          dst_port1=None, dst_port2=None, dst_proto='predefined_tcp')
```

Create a blocklist entry.

A blocklist can be added directly from the engine node, or from the system context. If submitting from the system context, it becomes a global blocklist. This will return the properly formatted json to submit.

#### Parameters

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or 'any'
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or 'any'
- **duration** (*int*) – length of time to blocklist

Both the system and engine context blocklist allow kw to be passed to provide additional functionality such as adding source and destination ports or port ranges and specifying the protocol. The following parameters define the kw that can be passed.

The following example shows creating an engine context blocklist using additional kw:

```
engine.block_list('1.1.1.1/32', '2.2.2.2/32', duration=3600,
                 src_port1=1000, src_port2=1500, src_proto='predefined_udp',
                 dst_port1=3, dst_port2=3000, dst_proto='predefined_udp')
```

#### Parameters

- **src\_port1** (*int*) – start source port to limit blocklist
- **src\_port2** (*int*) – end source port to limit blocklist
- **src\_proto** (*str*) – source protocol. Either 'predefined\_tcp' or 'predefined\_udp'. (default: 'predefined\_tcp')
- **dst\_port1** (*int*) – start dst port to limit blocklist
- **dst\_port2** (*int*) – end dst port to limit blocklist
- **dst\_proto** (*str*) – dst protocol. Either 'predefined\_tcp' or 'predefined\_udp'. (default: 'predefined\_tcp')

---

**Note:** if blocking a range of ports, use both src\_port1 and src\_port2, otherwise providing only src\_port1 is adequate. The same applies to dst\_port1 / dst\_port2. In addition, if you provide src\_portX but not dst\_portX (or vice versa), the undefined port side definition will default to all ports.

---

```
class smc.elements.other.Category(name=None, **meta)
```

A Category is used by an element to group and categorize elements by some criteria. Once a category is created, it can be assigned to the element and used as a search filter when managing large numbers of elements. A category can be added to a category tag (or tags) to provide a higher level container/group for searching.

```
>>> from smc.elements.other import Category
>>> Category.create(name='footag', comment='test tag')
Category(name=footag)
```

**Variables**

**categories** (*list*(*CategoryTag*)) – category tags for this category

**add\_category**(*tags*)

Category Tags are used to characterize an element by a type identifier. They can then be searched and returned as a group of elements. If the category tag specified does not exist, it will be created. This change will take effect immediately.

**Parameters**

**tags** (*list*(*str*)) – list of category tag names to add to this element

**Raises**

*ElementNotFound* – Category tag element name not found

**Returns**

None

**See also:**

*smc.elements.other.Category*

**add\_category\_tag**(*tags*, *append\_lists=True*)

Add this category to a category tag (group). This provides drop down filters in the SMC by category tag.

**Parameters**

- **tags** (*list*(*str*)) – category tag by name
- **append\_lists** (*bool*) – append to existing tags or overwrite default: append)

**Returns**

None

**add\_element**(*element*)

Element can be href or type *smc.base.model.Element*

```
>>> from smc.elements.other import Category
>>> category = Category('foo')
>>> category.add_element(Host('kali'))
```

**Parameters**

**element** (*str*, *Element*) – element to add to tag

**Raises**

*ModificationFailed*: failed adding element

**Returns**

None

**classmethod create**(*name*, *comment=None*)

Add a category element

**Parameters**

**name** – name of location

**Returns**

instance with meta

**Return type**

*Category*

**remove\_element(*element*)**

Remove an element from this category tag. Find elements assigned by [search\\_elements\(\)](#). Element can be str href or type `smc.base.model.Element`.

```
>>> from smc.elements.other import Category
>>> from smc.elements.network import Host
>>> category.remove_element(Host('kali'))
```

**Parameters**

**element** (*str*, *Element*) – element to remove

**Raises**

[ModificationFailed](#) – cannot remove element

**Returns**

None

**search\_elements()**

Find all elements assigned to this category tag. You can also find category tags assigned directly to an element also:

```
>>> host = Host('kali')
>>> host.categories
[Category(name=myelements), Category(name=foocategory)]
```

**Returns**

`smc.base.model.Element`

**Return type**

`list`

**class smc.elements.other.CategoryTag(*name=None, \*\*meta*)**

A Category Tag is a grouping of categories within SMC. Category Tags are used as filters (typically in the SMC) to change the view based on the tag.

**Variables**

- **child\_categories** (`list(Category, CategoryTag)`) – child categories
- **parent\_categories** (`list(Category, CategoryTag)`) – parent categories

**classmethod create(*name, comment=None*)**

Create a CategoryTag. A category tag represents a group of categories or a group of category tags (nested groups). These are used to provide filtering views within the SMC and organize elements by user defined criteria.

**Parameters**

- **name** (*str*) – name of category tag
- **comment** (*str*) – optional comment

**Raises**

[CreateElementFailed](#) – problem creating tag

**Returns**

instance with meta

**Return type***CategoryTag***remove\_category**(*categories*)

Remove a category from this Category Tag (group).

**Parameters****categories** (*list*(*str*, *Element*)) – categories to remove**Returns**

None

**class** `smc.elements.other.ContactAddress`(*data=None*, *\*\*kwargs*)

A contact address is used by elements to provide an alternative IP or FQDN mapping based on a location

**property addresses**

List of addresses set as contact address

**Return type***list***property name**

Location name for this contact address

**Return type***str***class** `smc.elements.other.FilterExpression`(*name=None*, *\*\*meta*)

A filter expression defines either a system element filter or a user defined filter based on an expression. For example, a system level filter is ‘Match All’. For classes that allow filters as input, a filter expression can be used.

**class** `smc.elements.other.Geolocation`(*name=None*, *\*\*meta*)

Geolocation objects are mutable as of SMC version 6.6

New in version 0.7.0.

**class** `smc.elements.other.HTTPSInspectionExceptions`(*name=None*, *\*\*meta*)

The HTTPS Inspection Exceptions element is a list of domains that are excluded from decryption and inspection. HTTPS Inspection Exceptions are used in a custom HTTPS service to define a list of domains for which HTTPS traffic is not decrypted. The custom HTTPS service must be used in a rule, and only traffic that matches the rule is excluded from decryption and inspection.

---

**Note:** As of SMC 6.4.3, this is a read-only element

---

**class** `smc.elements.other.Location`(*name=None*, *\*\*meta*)

Locations are used by elements to identify when they are behind a NAT connection. For example, if you have an engine that connects to the SMC across the internet using a public address, a location will be the tag applied to the Management Server (with contact address) and on the engine to identify how to connect. In this case, the location will map to a contact address using a public IP.

---

**Note:** Locations require SMC API version >= 6.1

---

**classmethod** **create**(*name*, *comment=None*)

Create a location element

**Parameters****name** – name of location

**Raises***CreateElementFailed* – failed creating element with reason**Returns**

instance with meta

**Return type***Location***property used\_on**

Return all NAT'd elements using this location.

---

**Note:** Available only in SMC version 6.2

---

**Returns**

elements used by this location

**Return type**

list

**class** smc.elements.other.LogicalInterface(name=None, \*\*meta)

Logical interface is used on either inline or capture interfaces. If an engine has both inline and capture interfaces (L2 Firewall or IPS role), then you must use a unique Logical Interface on the interface type.

Create a logical interface:

```
LogicalInterface.create('mylogical_interface')
```

**classmethod** create(name, comment=None)

Create the logical interface

**Parameters**

- **name** (*str*) – name of logical interface
- **comment** (*str*) – optional comment

**Raises***CreateElementFailed* – failed creating element with reason**Returns**

instance with meta

**Return type***LogicalInterface***class** smc.elements.other.MacAddress(name=None, \*\*meta)

Mac Address network element that can be used in L2 and IPS policy source and destination fields.

Creating a MacAddress:

```
>>> MacAddress.create(name='mymac', mac_address='22:22:22:22:22:22')
MacAddress(name=mymac)
```

**classmethod** create(name, mac\_address, comment=None)

Create the Mac Address

**Parameters**

- **name** (*str*) – name of mac address
- **mac\_address** (*str*) – mac address notation
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta

**Return type**

*MacAddress*

**class** smc.elements.other.**QueryDataFilter**(*name=None, \*\*meta*)

A Query Data Filter is an Internal filter container. Created on the fly when adding filtering context in some components

**class** smc.elements.other.**RuleValidityTime**(*name=None, \*\*meta*)

This represents a Rule Validity Time.

**class** smc.elements.other.**SituationTag**(*name=None, \*\*meta*)

A situation tag is used to categorize situations based on some sort of user defined criteria such as Botnet, Attacks, etc. These can help with categorization of specific threat event types.

**class** smc.elements.other.**UpdateServerProfile**(*name=None, \*\*meta*)

This represents Update Server Profile (aka Update Service)

**classmethod** **create**(*name, retry=0, timeout=0, urls=None, tls\_profile\_ref=None, comment=None*)

Create a UpdateServerProfile. A Update Server Profile represents a update service. These are used to provide server providing the updates within the SMC.

**Parameters**

- **name** (*str*) – name of category tag.
- **retry** (*int*) – number of retries.
- **timeout** (*int*) – connection timeout.
- **urls** (*list*) – list of URL, at least one is mandatory.
- **tls\_profile\_ref** (*str*) – TLS profile used to connect to the server(s).
- **comment** (*str*) – optional comment.

**Raises**

*CreateElementFailed* – problem creating tag.

**Returns**

instance with meta.

**Return type**

CategoryTag.

**property** **ordered\_url**

The list of url used by update service. :rtype List(url)

**property** **retry**

number of retries

**property timeout**

It is connection timeout

**property tls\_profile**

TLS profile used to connect to the server(s). :rtype TLSProfile

```
smc.elements.other.prepare_blacklist(src, dst, duration=3600, src_port1=None, src_port2=None,
                                     src_proto='predefined_tcp', dst_port1=None, dst_port2=None,
                                     dst_proto='predefined_tcp')
```

Create a blacklist entry.

A blacklist can be added directly from the engine node, or from the system context. If submitting from the system context, it becomes a global blacklist. This will return the properly formatted json to submit.

**Parameters**

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or ‘any’
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or ‘any’
- **duration** (*int*) – length of time to blacklist

Both the system and engine context blacklist allow kw to be passed to provide additional functionality such as adding source and destination ports or port ranges and specifying the protocol. The following parameters define the kw that can be passed.

The following example shows creating an engine context blacklist using additional kw:

```
engine.blacklist('1.1.1.1/32', '2.2.2.2/32', duration=3600,
                 src_port1=1000, src_port2=1500, src_proto='predefined_udp',
                 dst_port1=3, dst_port2=3000, dst_proto='predefined_udp')
```

**Parameters**

- **src\_port1** (*int*) – start source port to limit blacklist
- **src\_port2** (*int*) – end source port to limit blacklist
- **src\_proto** (*str*) – source protocol. Either ‘predefined\_tcp’ or ‘predefined\_udp’. (default: ‘predefined\_tcp’)
- **dst\_port1** (*int*) – start dst port to limit blacklist
- **dst\_port2** (*int*) – end dst port to limit blacklist
- **dst\_proto** (*str*) – dst protocol. Either ‘predefined\_tcp’ or ‘predefined\_udp’. (default: ‘predefined\_tcp’)

---

**Note:** if blocking a range of ports, use both src\_port1 and src\_port2, otherwise providing only src\_port1 is adequate. The same applies to dst\_port1 / dst\_port2. In addition, if you provide src\_portX but not dst\_portX (or vice versa), the undefined port side definition will default to all ports.

---



---

**Note:** This method requires SMC version < 7.0

---

```
smc.elements.other.prepare_block_list(src, dst, duration=3600, src_port1=None, src_port2=None,
                                       src_proto='predefined_tcp', dst_port1=None, dst_port2=None,
                                       dst_proto='predefined_tcp')
```

Create a `block_list` entry.

A blocklist can be added directly from the engine node, or from the system context. If submitting from the system context, it becomes a global blocklist. This will return the properly formatted json to submit.

#### Parameters

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or ‘any’
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or ‘any’
- **duration** (*int*) – length of time to blocklist

Both the system and engine context blocklist allow kw to be passed to provide additional functionality such as adding source and destination ports or port ranges and specifying the protocol. The following parameters define the kw that can be passed.

The following example shows creating an engine context blocklist using additional kw:

```
engine.block_list('1.1.1.1/32', '2.2.2.2/32', duration=3600,
                 src_port1=1000, src_port2=1500, src_proto='predefined_udp',
                 dst_port1=3, dst_port2=3000, dst_proto='predefined_udp')
```

#### Parameters

- **src\_port1** (*int*) – start source port to limit blocklist
- **src\_port2** (*int*) – end source port to limit blocklist
- **src\_proto** (*str*) – source protocol. Either ‘predefined\_tcp’ or ‘predefined\_udp’. (default: ‘predefined\_tcp’)
- **dst\_port1** (*int*) – start dst port to limit blocklist
- **dst\_port2** (*int*) – end dst port to limit blocklist
- **dst\_proto** (*str*) – dst protocol. Either ‘predefined\_tcp’ or ‘predefined\_udp’. (default: ‘predefined\_tcp’)

---

**Note:** if blocking a range of ports, use both `src_port1` and `src_port2`, otherwise providing only `src_port1` is adequate. The same applies to `dst_port1` / `dst_port2`. In addition, if you provide `src_portX` but not `dst_portX` (or vice versa), the undefined port side definition will default to all ports.

---

---

**Note:** This method requires SMC version >= 7.0

---

### 14.5.5.1 Blacklist

#### `class smc.elements.other.Blacklist`

Blacklist provides a simple container to add multiple blacklist entries. Pass an instance of this to `smc.core.engine.blacklist_bulk` to upload to the engine.

---

**Note:** This method requires SMC version < 7.0

---

since this version, “blacklist” is renamed “block\_list”

```
add_entry(src, dst, duration=3600, src_port1=None, src_port2=None, src_proto='predefined_tcp',
          dst_port1=None, dst_port2=None, dst_proto='predefined_tcp')
```

Create a blacklist entry.

A blacklist can be added directly from the engine node, or from the system context. If submitting from the system context, it becomes a global blacklist. This will return the properly formatted json to submit.

#### Parameters

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or 'any'
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or 'any'
- **duration** (*int*) – length of time to blacklist

Both the system and engine context blacklist allow kw to be passed to provide additional functionality such as adding source and destination ports or port ranges and specifying the protocol. The following parameters define the kw that can be passed.

The following example shows creating an engine context blacklist using additional kw:

```
engine.blacklist('1.1.1.1/32', '2.2.2.2/32', duration=3600,
                src_port1=1000, src_port2=1500, src_proto='predefined_udp',
                dst_port1=3, dst_port2=3000, dst_proto='predefined_udp')
```

#### Parameters

- **src\_port1** (*int*) – start source port to limit blacklist
- **src\_port2** (*int*) – end source port to limit blacklist
- **src\_proto** (*str*) – source protocol. Either 'predefined\_tcp' or 'predefined\_udp'. (default: 'predefined\_tcp')
- **dst\_port1** (*int*) – start dst port to limit blacklist
- **dst\_port2** (*int*) – end dst port to limit blacklist
- **dst\_proto** (*str*) – dst protocol. Either 'predefined\_tcp' or 'predefined\_udp'. (default: 'predefined\_tcp')

---

**Note:** if blocking a range of ports, use both src\_port1 and src\_port2, otherwise providing only src\_port1 is adequate. The same applies to dst\_port1 / dst\_port2. In addition, if you provide src\_portX but not dst\_portX (or vice versa), the undefined port side definition will default to all ports.

---

### 14.5.5.2 Category

```
class smc.elements.other.Category(name=None, **meta)
```

Bases: *Element*

A Category is used by an element to group and categorize elements by some criteria. Once a category is created, it can be assigned to the element and used as a search filter when managing large numbers of elements. A category can be added to a category tag (or tags) to provide a higher level container/group for searching.

```
>>> from smc.elements.other import Category
>>> Category.create(name='footag', comment='test tag')
Category(name=footag)
```

**Variables**

**categories** (*list*(*CategoryTag*)) – category tags for this category

**add\_category**(*tags*)

Category Tags are used to characterize an element by a type identifier. They can then be searched and returned as a group of elements. If the category tag specified does not exist, it will be created. This change will take effect immediately.

**Parameters**

**tags** (*list*(*str*)) – list of category tag names to add to this element

**Raises**

*ElementNotFound* – Category tag element name not found

**Returns**

None

**See also:**

*smc.elements.other.Category*

**add\_category\_tag**(*tags*, *append\_lists=True*)

Add this category to a category tag (group). This provides drop down filters in the SMC by category tag.

**Parameters**

- **tags** (*list*(*str*)) – category tag by name
- **append\_lists** (*bool*) – append to existing tags or overwrite default: append)

**Returns**

None

**add\_element**(*element*)

Element can be href or type *smc.base.model.Element*

```
>>> from smc.elements.other import Category
>>> category = Category('foo')
>>> category.add_element(Host('kali'))
```

**Parameters**

**element** (*str*, *Element*) – element to add to tag

**Raises**

*ModificationFailed*: failed adding element

**Returns**

None

**classmethod create**(*name*, *comment=None*)

Add a category element

**Parameters**

**name** – name of location

**Returns**

instance with meta

**Return type**

*Category*

**remove\_element(*element*)**

Remove an element from this category tag. Find elements assigned by [search\\_elements\(\)](#). Element can be str href or type `smc.base.model.Element`.

```
>>> from smc.elements.other import Category
>>> from smc.elements.network import Host
>>> category.remove_element(Host('kali'))
```

**Parameters**

**element** (*str*, *Element*) – element to remove

**Raises**

[ModificationFailed](#) – cannot remove element

**Returns**

None

**search\_elements()**

Find all elements assigned to this category tag. You can also find category tags assigned directly to an element also:

```
>>> host = Host('kali')
>>> host.categories
[Category(name=myelements), Category(name=foocategory)]
```

**Returns**

`smc.base.model.Element`

**Return type**

list

**14.5.5.3 CategoryTag**

**class** `smc.elements.other.CategoryTag`(*name=None*, *\*\*meta*)

Bases: [Element](#)

A Category Tag is a grouping of categories within SMC. Category Tags are used as filters (typically in the SMC) to change the view based on the tag.

**Variables**

- **child\_categories** (*list*(*Category*, *CategoryTag*)) – child categories
- **parent\_categories** (*list*(*Category*, *CategoryTag*)) – parent categories

**classmethod** `create`(*name*, *comment=None*)

Create a CategoryTag. A category tag represents a group of categories or a group of category tags (nested groups). These are used to provide filtering views within the SMC and organize elements by user defined criteria.

**Parameters**

- **name** (*str*) – name of category tag
- **comment** (*str*) – optional comment

**Raises**

[CreateElementFailed](#) – problem creating tag

**Returns**

instance with meta

**Return type***CategoryTag***remove\_category**(*categories*)

Remove a category from this Category Tag (group).

**Parameters****categories** (*list*(*str*, *Element*)) – categories to remove**Returns**

None

#### 14.5.5.4 FilterExpression

**class** `smc.elements.other.FilterExpression`(*name=None*, *\*\*meta*)Bases: *Element*

A filter expression defines either a system element filter or a user defined filter based on an expression. For example, a system level filter is ‘Match All’. For classes that allow filters as input, a filter expression can be used.

#### 14.5.5.5 Location

**class** `smc.elements.other.Location`(*name=None*, *\*\*meta*)Bases: *Element*

Locations are used by elements to identify when they are behind a NAT connection. For example, if you have an engine that connects to the SMC across the internet using a public address, a location will be the tag applied to the Management Server (with contact address) and on the engine to identify how to connect. In this case, the location will map to a contact address using a public IP.

---

**Note:** Locations require SMC API version >= 6.1

---

**classmethod** **create**(*name*, *comment=None*)

Create a location element

**Parameters****name** – name of location**Raises***CreateElementFailed* – failed creating element with reason**Returns**

instance with meta

**Return type***Location***property** **used\_on**

Return all NAT'd elements using this location.

---

**Note:** Available only in SMC version 6.2

---

**Returns**

elements used by this location

**Return type**

list

**14.5.5.6 LogicalInterface**

**class** smc.elements.other.**LogicalInterface**(name=None, \*\*meta)

Bases: *Element*

Logical interface is used on either inline or capture interfaces. If an engine has both inline and capture interfaces (L2 Firewall or IPS role), then you must use a unique Logical Interface on the interface type.

Create a logical interface:

```
LogicalInterface.create('mylogical_interface')
```

**classmethod** **create**(name, comment=None)

Create the logical interface

**Parameters**

- **name** (*str*) – name of logical interface
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta

**Return type**

*LogicalInterface*

**14.5.5.7 MacAddress**

**class** smc.elements.other.**MacAddress**(name=None, \*\*meta)

Bases: *Element*

Mac Address network element that can be used in L2 and IPS policy source and destination fields.

Creating a MacAddress:

```
>>> MacAddress.create(name='mymac', mac_address='22:22:22:22:22:22')
MacAddress(name=mymac)
```

**classmethod** **create**(name, mac\_address, comment=None)

Create the Mac Address

**Parameters**

- **name** (*str*) – name of mac address
- **mac\_address** (*str*) – mac address notation
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating element with reason

**Returns**

instance with meta

**Return type**

*MacAddress*

#### 14.5.5.8 HTTPSInspectionExceptions

**class** smc.elements.other.HTTPSInspectionExceptions(*name=None, \*\*meta*)

Bases: *Element*

The HTTPS Inspection Exceptions element is a list of domains that are excluded from decryption and inspection. HTTPS Inspection Exceptions are used in a custom HTTPS service to define a list of domains for which HTTPS traffic is not decrypted. The custom HTTPS service must be used in a rule, and only traffic that matches the rule is excluded from decryption and inspection.

---

**Note:** As of SMC 6.4.3, this is a read-only element

---

#### 14.5.6 Sidewinder Elements

Module providing sidewinder element creation.

The different sidewinder elements in this module that can be configured are SSM Logging Profiles, SSH Profiles, SSH Known Hosts, and SSH Known Hosts Lists.

**class** smc.elements.ssm.DeviceToolsProfile(*name=None, \*\*meta*)

**class** smc.elements.ssm.LoggingProfile(*name=None, \*\*meta*)

**class** smc.elements.ssm.ProbingProfile(*name=None, \*\*meta*)

**class** smc.elements.ssm.SSHKnownHosts(*name=None, \*\*meta*)

Class representing a SSH Known Host object used in SSH Known Hosts Lists

Create a SSH Known Host element with ipv4 and with ssh-rsa:

```
SSHKnownHosts.create(name='ipv4_rsa_known_host',
                      sshkey_type='ssh-rsa',
                      host_key='<ssh_host_rsa_key.pub output>',
                      ipaddress='1.1.1.1', port=22000,
                      comment='some comment for my known host')
```

Create a SSH Known Host element with ipv6 and with ecdsa-sha2-nistp256:

```
Host.create(name='ipv6_ecdsa_known_host',
            ssh_type='ecdsa-sha2-nistp256',
            host_key='<ssh_host_ecdsa_key.pub output>',
            ipv6_address='2001:cdba::3257:9652')
```

Available attributes:

**Variables**

- **ipaddress** (*str*) – IPv4 address for this element
- **ipv6\_address** (*str*) – IPv6 address for this element
- **sshkey\_type** (*str*) – ssh key type (i.e. ssh-rsa, ecdsa-sha2-nistp256)
- **host\_key** (*str*) – public host key of host
- **port** (*int*) – port number for this element

**classmethod** **create**(*name*, *host\_key*, *sshkey\_type*, *ipaddress=None*, *ipv6\_address=None*, *port=22*, *comment=None*)

Create the SSH Known Host

#### Parameters

- **name** (*str*) – name of ssh known host
- **host\_key** (*str*) – string of public host key of known host
- **sshkey\_type** (*str*) – ssh key type of the known host public key
- **ipaddress** (*str*) – IPv4 address of known host
- **ipv6\_ipaddress** (*str*) – IPv6 address of known host
- **port** (*int*) – port number of the ssh known host element
- **comment** (*str*) – comment (optional)

#### Raises

**CreateElementFailed** – failure creating element with reason

#### Returns

instance with meta

#### Return type

*SSHKnownHosts*

---

**Note:** Either an ipaddress or ipv6\_address must be specified

---

**class** `smc.elements.ssm.SSHKnownHostsLists`(*name=None*, *\*\*meta*)

Class representing a SSH Known Hosts List object used in the Sidewinder Proxy Add-On on the engine

Create a SSH Known Hosts List element and add a created Known Host to it:

```
known_host = SSHKnownHosts.create(name='ipv4_rsa_known_host',
                                   sshkey_type='ssh-rsa',
                                   host_key='<ssh_host_rsa_key.pub output>',
                                   ipaddress='1.1.1.1', port=22000,
                                   comment='some comment for my known host')

SSHKnownHostsLists.create(name='myknownhostslist',
                           known_host=[known_host.href],
                           comment='some comment for my known host list')
```

Available attributes:

#### Variables

**known\_host** (*list*) – href of known host for this element

**classmethod** `create(name, known_host=[], comment=None)`

Create the SSH Known Host List

**Parameters**

- **name** (*str*) – name of ssh known host list
- **known\_host** (*list*) – href of ssh known hosts to add to the list
- **comment** (*str*) – comment (optional)

**Raises**

*CreateElementFailed* – failure creating element with reason

**Returns**

instance with meta

**Return type**

*SSHKnownHostsLists*

**class** `smc.elements.ssm.SSHProfile(name=None, **meta)`

Class representing a SSH Profile object used in SSM SSH Proxy services

Create a SSH Profile element with ciphers, kex, and macs:

```
SSHProfile.create(name='mysshprofile',
                  cipher='aes256-ctr,aes128-ctr,aes192-ctr,aes128-cbc',
                  kex='ecdh-sha2-nistp256,diffie-hellman-group14-sha1',
                  mac='hmac-sha2-256,hmac-sha2-512,hmac-md5-etm@openssh.com',
                  comment='some comment for my ssh profile')
```

Available attributes:

**Variables**

- **cipher** (*str*) – cipher algorithms set for this element
- **kex** (*str*) – key exchange algorithms set for this element
- **mac** (*str*) – mac algorithms set for this element

---

**Note:** Each algorithm type must have an algorithm set

---

**classmethod** `create(name, cipher, kex, mac, comment=None)`

Create the SSH Profile

**Parameters**

- **name** (*str*) – name of ssh profile
- **cipher** (*str*) – string of cipher algorithms (comma separated)
- **kex** (*str*) – string of key exchange algorithms (comma separated)
- **mac** (*str*) – string of mac algorithms (comma separated)
- **comment** (*str*) – comment (optional)

**Raises**

*CreateElementFailed* – failure creating element with reason

**Returns**

instance with meta

**Return type**  
*SSHProfile*

---

**Note:** No algorithm type can be empty

---

```
class smc.elements.ssm.SSMLoggingProfile(name=None, **meta)
```

#### 14.5.6.1 SSHProfile

```
class smc.elements.ssm.SSHProfile(name=None, **meta)
```

Bases: *Element*

Class representing a SSH Profile object used in SSM SSH Proxy services

Create a SSH Profile element with ciphers, kex, and macs:

```
SSHProfile.create(name='mysshprofile',
                  cipher='aes256-ctr,aes128-ctr,aes192-ctr,aes128-cbc',
                  kex='ecdh-sha2-nistp256,diffie-hellman-group14-sha1',
                  mac='hmac-sha2-256,hmac-sha2-512,hmac-md5-etm@openssh.com',
                  comment='some comment for my ssh profile')
```

Available attributes:

##### Variables

- **cipher** (*str*) – cipher algorithms set for this element
- **kex** (*str*) – key exchange algorithms set for this element
- **mac** (*str*) – mac algorithms set for this element

---

**Note:** Each algorithm type must have an algorithm set

---

```
classmethod create(name, cipher, kex, mac, comment=None)
```

Create the SSH Profile

##### Parameters

- **name** (*str*) – name of ssh profile
- **cipher** (*str*) – string of cipher algorithms (comma separated)
- **kex** (*str*) – string of key exchange algorithms (comma separated)
- **mac** (*str*) – string of mac algorithms (comma separated)
- **comment** (*str*) – comment (optional)

##### Raises

*CreateElementFailed* – failure creating element with reason

##### Returns

instance with meta

**Return type**  
*SSHProfile*

---

**Note:** No algorithm type can be empty

---

### 14.5.6.2 SSHKnownHosts

**class** `smc.elements.ssm.SSHKnownHosts`(*name=None, \*\*meta*)

Bases: [Element](#)

Class representing a SSH Known Host object used in SSH Known Hosts Lists

Create a SSH Known Host element with ipv4 and with ssh-rsa:

```
SSHKnownHosts.create(name='ipv4_rsa_known_host',
                      sshkey_type='ssh-rsa',
                      host_key='<ssh_host_rsa_key.pub output>',
                      ipaddress='1.1.1.1', port=22000,
                      comment='some comment for my known host')
```

Create a SSH Known Host element with ipv6 and with ecdsa-sha2-nistp256:

```
Host.create(name='ipv6_ecdsa_known_host',
            ssh_type='ecdsa-sha2-nistp256',
            host_key='<ssh_host_ecdsa_key.pub output>',
            ipv6_address='2001:cdba::3257:9652')
```

Available attributes:

#### Variables

- **ipaddress** (*str*) – IPv4 address for this element
- **ipv6\_address** (*str*) – IPv6 address for this element
- **sshkey\_type** (*str*) – ssh key type (i.e. ssh-rsa, ecdsa-sha2-nistp256)
- **host\_key** (*str*) – public host key of host
- **port** (*int*) – port number for this element

**classmethod** `create`(*name, host\_key, sshkey\_type, ipaddress=None, ipv6\_address=None, port=22, comment=None*)

Create the SSH Known Host

#### Parameters

- **name** (*str*) – name of ssh known host
- **host\_key** (*str*) – string of public host key of known host
- **sshkey\_type** (*str*) – ssh key type of the known host public key
- **ipaddress** (*str*) – IPv4 address of known host
- **ipv6\_ipaddress** (*str*) – IPv6 address of known host
- **port** (*int*) – port number of the ssh known host element
- **comment** (*str*) – comment (optional)

#### Raises

[CreateElementFailed](#) – failure creating element with reason

**Returns**

instance with meta

**Return type***SSHKnownHosts*


---

**Note:** Either an `ipaddress` or `ipv6_address` must be specified

---

### 14.5.6.3 SSHKnownHostsLists

**class** `smc.elements.ssm.SSHKnownHostsLists`(*name=None, \*\*meta*)Bases: *Element*

Class representing a SSH Known Hosts List object used in the Sidewinder Proxy Add-On on the engine

Create a SSH Known Hosts List element and add a created Known Host to it:

```
known_host = SSHKnownHosts.create(name='ipv4_rsa_known_host',
                                   sshkey_type='ssh-rsa',
                                   host_key='<ssh_host_rsa_key.pub output>',
                                   ipaddress='1.1.1.1', port=22000,
                                   comment='some comment for my known host')

SSHKnownHostsLists.create(name='myknownhostslist',
                           known_host=[known_host.href],
                           comment='some comment for my known host list')
```

Available attributes:

**Variables****known\_host** (*list*) – href of known host for this element**classmethod** `create`(*name, known\_host=[], comment=None*)

Create the SSH Known Host List

**Parameters**

- **name** (*str*) – name of ssh known host list
- **known\_host** (*list*) – href of ssh known hosts to add to the list
- **comment** (*str*) – comment (optional)

**Raises***CreateElementFailed* – failure creating element with reason**Returns**

instance with meta

**Return type***SSHKnownHostsLists*

## 14.5.7 Situations

Module that represents inspection and correlated situations.

New in version 0.6.3: Requires SMC version  $\geq 6.5$

Situations can be either inspection related or correlated. Both types can be searched to obtain collections.

Every situation has an associated ‘context’ which identifies properties of the situation and how matching or correlation is performed.

A situation context group is a top level structure that encapsulates similar individual inspection contexts. You can retrieve these as follows:

```
>>> from smc.elements.situations import SituationContextGroup
>>> for group in SituationContextGroup.objects.all():
...     group
...
SituationContextGroup(name=DoS Detection)
SituationContextGroup(name=FINGER)
SituationContextGroup(name=SMTP Deprecated)
SituationContextGroup(name=PPTP)
SituationContextGroup(name=IPv6)
SituationContextGroup(name=NETBIOS)
SituationContextGroup(name=SIP)
SituationContextGroup(name=SNMP)
```

You can optionally retrieve situation context groups directly, and iterate the inspection contexts (sub\_elements), which might be additional situation context groups or inspection contexts:

```
>>> group = SituationContextGroup('DoS Detection')
>>> group.sub_elements
[InspectionSituationContext(name=TCP synflood detection (SYN-ACK timeout based_
↪detection)),
 InspectionSituationContext(name=TCP synflood detection (SYN-timeout method)),
 InspectionSituationContext(name=Non-ratebased DoS attacks),
 InspectionSituationContext(name=TCP DoS events),
 InspectionSituationContext(name=UDP DoS events),
 InspectionSituationContext(name=UDP DoS detected)]
```

If you are interested in inspection contexts directly (i.e. groups are ‘flattened’ out), you can retrieve these as follows:

```
>>> from smc.elements.situations import InspectionSituationContext
>>> for context in InspectionSituationContext.objects.all():
...     context
...
InspectionSituationContext(name=Context for DNS_POLICY_NOTIFY_FAIL)
InspectionSituationContext(name=Context for FTP AUTH success)
InspectionSituationContext(name=TCP PPTP Server Stream)
InspectionSituationContext(name=Context for SMTP_INCONSISTENT_REPLIES)
InspectionSituationContext(name=Context for TCP Option Too Short)
InspectionSituationContext(name=RIFF File Stream)
InspectionSituationContext(name=Context for IP Total Length Error)
...
```

You can optionally retrieve an inspection situation context directly. Most situation contexts are system level elements and will be read only, but you can fetch them to view configurations if necessary.

Every situation context will have at least one *situation parameter*, which is the parameter / value pair used to match the on inspection situations which are categorized by the situation context. For example, in the case of detecting a text file stream, a single regular expression type situation parameter is used:

```
>>> context = InspectionSituationContext('Text File Stream')
>>> for parameter in context.situation_parameters:
...     parameter
...
SituationParameter(name=Regular Expression)
```

Inspection Situations are the individual events that are either predefined or system defined that identify specific events to inspect for. All inspection situations have an inspection context (see above), and can also be customized or be duplicated.

Creating an inspection situation is a two step process. You must first create the situation with a specified context, then add the necessary parameter values.

An example of creating a new situation that uses a regular expression pattern to match within a Text File Stream:

```
>>> from smc.elements.situations import InspectionSituation
>>> from smc.elements.situations import InspectionSituationContext
>>>
>>> situation = InspectionSituation.create(name='foosituation',
...                                     situation_context=InspectionSituationContext('Text File Stream'),
...                                     severity='high')
>>> situation
InspectionSituation(name=foosituation)
>>> situation.create_regular_expression(r'(?x)\n.*ActiveXObject \x28 \x22 WScript\.'
...                                   'Shell(?:[s_file_text_script -> sid()])\n')
>>>
```

```
class smc.elements.situations.CorrelationSituation(name=None, **meta)
```

Bases: *Situation*

Correlation Situations are used by NGFW Engines and Log Servers to conduct further analysis of detected events. Correlation Situations do not handle traffic directly. Instead they analyze the events generated by matches to Situations found in traffic. Correlation Situations use Event Binding elements to define the log events that bind together different types of events in traffic.

```
class smc.elements.situations.CorrelationSituationContext(name=None, **meta)
```

Bases: *SituationContext*

Correlation Contexts define the patterns for matching groups of related events in traffic. Examples of correlation contexts are Count, Compress, Group, Match and Sequence. See SMC documentation for more details on each context type and meaning.

**property situation\_parameters**

Situation parameters defining detection logic for the context. This will return a list of SituationParameter indicating how the detection is made, i.e. regular expression, integer value, etc.

**Return type**

*list(SituationParameter)*

```
class smc.elements.situations.EcaOperatingSystemSituation(name=None, **meta)
```

Bases: *Situation*

Used to configure ECA Endpoint setting

```
class smc.elements.situations.InspectionSituation(name=None, **meta)
```

Bases: [Situation](#)

It is an element that identifies and describes detected events in the traffic or in the operation of the system. Situations contain the Context information, i.e., a pattern that the system is to look for in the inspected traffic.

```
classmethod create(name, situation_context, attacker=None, target=None, severity='information',
                   situation_type=None, description=None, comment=None)
```

Create an inspection situation.

#### Parameters

- **name** (*str*) – name of the situation
- **situation\_context** ([InspectionSituationContext](#)) – The situation context type used to define this situation. Identifies the proper parameter that identifies how the situation is defined (i.e. regex, etc).
- **attacker** (*str*) – Attacker information, used to identify last packet the triggers attack and is only used for blacklisting. Values can be packet\_source, packet\_destination, connection\_source, or connection\_destination
- **target** (*str*) – Target information, used to identify the last packet that triggers the attack and is only used for blacklisting. Values can be packet\_source, packet\_destination, connection\_source, or connection\_destination
- **severity** (*str*) – severity for this situation. Valid values are critical, high, low, information
- **description** (*str*) – optional description
- **comment** (*str*) – optional comment

```
create_regular_expression(regex)
```

Create a regular expression for this inspection situation context. The inspection situation must be using an inspection context that supports regex.

#### Parameters

**regex** (*str*) – regular expression string

#### Raises

[CreateElementFailed](#) – failed to modify the situation

```
property vulnerability_references
```

If this inspection situation has associated CVE, OSVDB, BID, etc references, this will return those reference IDs

#### Return type

[list](#)(*str*)

```
class smc.elements.situations.InspectionSituationContext(name=None, **meta)
```

Bases: [SituationContext](#)

Represents groups of situation contexts that can be characterized by a common technique used for identifying the situation. Contexts also typically have in common the type of situation they apply to, i.e. *File Text Stream* would be an inspection context, and encapsulates inspection situations such as ActiveX in text file stream detection, etc.

```
class smc.elements.situations.Situation(name=None, **meta)
```

Bases: [Element](#)

Situation defines a common interface for inspection and correlated situations.

**property attacker**

How the Attacker is determined when the Situation matches. This information is used for blacklisting and in log entries and may be None

**Return type**

`str` or None

**property description**

The description for this situation

**Return type**

`str`

**property parameter\_values**

Parameter values for this inspection situation. This correlate to the the situation\_context.

**Return type**

`list(SituationParameterValue)`

**property severity**

The severity of this inspection situation, critical, high, low, information

**Return type**

`int`

**property target**

How the Target is determined when the Situation matches. This information is used for blacklisting and in log entries and may be None

**Return type**

`str` or None

**class** `smc.elements.situations.SituationContext`(*name=None, \*\*meta*)

Bases: `Element`

A situation context can be used by an inspection situation or by a correlated situation. The context defines the situation parameters used to define a pattern match and how that match is made.

**Variables**

- **name** (`str`) – name of this situation context
- **comment** (`str`) – comment for the context

**property description**

Description for this context

**Return type**

`str`

**property situation\_parameters**

Situation parameters defining detection logic for the context. This will return a list of SituationParameter indicating how the detection is made, i.e. regular expression, integer value, etc.

**Return type**

`list(SituationParameter)`

**class** `smc.elements.situations.SituationContextGroup`(*name=None, \*\*meta*)

Bases: `Element`

A situation context group is simply a top level group for organizing individual situation contexts. This is a top level element that can be retrieved directly:

```
>>> from smc.elements.situations import SituationContextGroup
>>> for group in SituationContextGroup.objects.all():
...     group
...
SituationContextGroup(name=DoS Detection)
SituationContextGroup(name=FINGER)
SituationContextGroup(name=SMTP Deprecated)
SituationContextGroup(name=PPTP)
SituationContextGroup(name=IPv6)
SituationContextGroup(name=NETBIOS)
SituationContextGroup(name=SIP)
SituationContextGroup(name=SNMP)
...
```

### Variables

**sub\_elements** (*list*(*InspectionContext*, *InspectionContextGroup*)) – the members of this inspection context group

**class** smc.elements.situations.**SituationParameter**(\*\**meta*)

Bases: *SubElement*

A situation parameter defines the parameter type used to define the inspection situation context. For example, Regular Expression would be a situation parameter.

#### **property display\_name**

The display name as shown in the SMC

##### **Return type**

*str*

#### **property order**

The order placement for this parameter. This is only relevant when there are multiple parameters in an inspection context definition.

##### **Return type**

*int*

#### **property type**

The type of this situation parameter in textual format. For example, integer, regexp, etc.

##### **Return type**

*str*

**class** smc.elements.situations.**SituationParameterValue**(\*\**meta*)

Bases: *SubElement*

The situation parameter value is associated with a situation parameter and as the name implies, provides the value payload for the given parameter.

**class** smc.elements.situations.**SubTLSMatchSituation**(*name=None*, \*\**meta*)

Bases: *Situation*

Used by *TLSMatchSituation*

**classmethod create**(*name*, *context*)

Create the sub tls match situation Used by *TLSMatchSituation*

### Parameters

- **name** (*str*) – name of sub tls match
- **context** (*str*) – context for sub tls match

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type**

*SubTLSMatchSituation*

```
class smc.elements.situations.TLSMatchSituation(name=None, **meta)
```

Bases: *Situation*

TLS Match elements define matching criteria for the use of the TLS protocol in traffic, and allow you to prevent the specified traffic from being decrypted. TLS Matches that deny decrypting are applied globally, even if the TLS Match elements are not used in the policy. However, TLS Match elements that are used in specific Access rules can override globally-applied TLS matches.

```
classmethod create(name, matching_domains=None,
                   match_certificate_validation='succeed_tls_validation',
                   validation_failed_matches=None, deny_decrypting=False, comment=None)
```

Create TLS Match

**Parameters**

- **name** (*str*) –
- **matching\_domains** (*list*) – list of domain url's
- **match\_certificate\_validation** (*str*) – possible values:
  - “succeed\_tls\_validation” to be used with matching\_domains parameter
  - “no\_validation”
  - “validation\_failed” to be used with validation\_failed\_matches parameter

**Parameters**

- **validation\_failed\_matches** (*list*) – possible values:
  - “match\_self\_signed\_certificates”
  - “match\_non\_trusted\_CAs”
  - “match\_expired\_certificates”
  - “match\_invalid\_certificates”

**Parameters**

- **deny\_decrypting** (*bool*) – deny decrypting default=False
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – element creation failed with reason

**Returns**

instance with meta

**Return type***TLSSMatchSituation***class** smc.elements.situations.TLSSMatchSituationContext(*name=None, \*\*meta*)Bases: *SituationContext*

Used by TLSSMatchSituation

## 14.5.8 Profiles

Profiles are generic container settings that are used in other areas of the SMC configuration. Each profile should document its usage and how it is referenced.

### 14.5.8.1 DNSRelayProfile

Profiles are templates used in other parts of the system to provide default functionality for specific feature sets. For example, to enable DNS Relay on an engine you must specify a DNSRelayProfile to use which defines the common settings (or sub-settings) for that feature.

A DNS Relay Profile allows multiple DNS related mappings that can be configured. Example usage:

```
>>> from smc.elements.profiles import DNSRelayProfile
>>> profile = DNSRelayProfile('mynewprofile')
```

---

**Note:** If the DNSRelayProfile does not exist, it will automatically be created when a DNS relay rule is added to the DNSRelayProfile instance.

---

Add a fixed domain answer rule:

```
>>> profile.fixed_domain_answer.add([('microsoft3.com', 'foo.com'), ('microsoft4.com',)])
>>> profile.fixed_domain_answer.all()
[{u'domain_name': u'microsoft3.com', u'translated_domain_name': u'foo.com'},
 {u'domain_name': u'microsoft4.com'}]
```

Translate hostnames (not fqdn) to a specific IP address:

```
>>> profile.hostname_mapping.add([('hostname1,hostname2', '1.1.1.12')])
>>> profile.hostname_mapping.all()
[{u'hostnames': u'hostname1,hostname2', u'ipaddress': u'1.1.1.12'}]
```

Translate an IP address to another:

```
>>> profile.dns_answer_translation.add([('12.12.12.12', '172.18.1.20')])
>>> profile.dns_answer_translation.all()
[{u'translated_ipaddress': u'172.18.1.20', u'original_ipaddress': u'12.12.12.12'}]
```

Specify a DNS server to handle specific domains:

```
>>> profile.domain_specific_dns_server.add([('myfoo.com', '172.18.1.20')])
>>> profile.domain_specific_dns_server.all()
[{u'dns_server_addresses': u'172.18.1.20', u'domain_name': u'myfoo.com'}]
```

**class** smc.elements.profiles.DNSAnswerTranslation(*profile*)

Bases: *DNSRule*

Map IPv4 addresses resolved by external DNS servers to IPv4 addresses in the internal network.

**add**(*answers*)

Takes an IPv4 address and translates to a specified IPv4 value. Provide a list of two-tuple with the first entry providing the original address and second entry specifying the translated address:

```
profile = DNSRelayProfile('dnsrules')
profile.dns_answer_translation.add([('12.12.12.12', '172.18.1.20')])
```

**Parameters**

**answers** (*tuple*[*str*, *str*]) – (original\_ipaddress, translated\_ipaddress)

**Raises**

*UpdateElementFailed* – failure to add to SMC

**Returns**

None

**class** smc.elements.profiles.DNSRelayProfile(*name=None*, *\*\*meta*)

Bases: *Element*

DNS Relay Settings specify a profile to handle how the engine will interpret DNS queries. The engine can act as a DNS relay, rewrite DNS queries or redirect domains to the specified DNS servers.

**property** dns\_answer\_translation

Add a DNS answer translation

**Return type**

*DNSAnswerTranslation*

**property** domain\_specific\_dns\_server

Add domain to DNS server mapping

**Return type**

*DomainSpecificDNSServer*

**property** fixed\_domain\_answer

Add a fixed domain answer entry.

**Return type**

*FixedDomainAnswer*

**property** hostname\_mapping

Add a hostname to IP mapping

**Return type**

*HostnameMapping*

**class** smc.elements.profiles.DNSRule(*profile*)

Bases: *object*

DNSRule is the parent class for all DNS relay rules.

**all**()

Return all entries

**Return type**`list(dict)`**class** `smc.elements.profiles.DomainSpecificDNSServer(profile)`Bases: `DNSRule`

Forward DNS requests to different DNS servers based on the requested domain.

**add**(*answers*)

Relay specific domains to a specified DNS server. Provide a list of two-tuple with first entry the domain name to relay for. The second entry is the DNS server that should handle the query:

```
profile = DNSRelayProfile('dnsrules')
profile.domain_specific_dns_server.add([('myfoo.com', '172.18.1.20')])
```

**Parameters**

**answers** (*tuple*[*str*, *str*]) – (domain\_name, dns\_server\_addresses), dns server addresses can be a comma separated string

**Raises**

`UpdateElementFailed` – failure to add to SMC

**Returns**

None

**class** `smc.elements.profiles.FixedDomainAnswer(profile)`Bases: `DNSRule`

Direct requests for specific domains to IPv4 addresses, IPv6 addresses, fully qualified domain names (FQDNs), or empty DNS replies

**add**(*answers*)

Add a fixed domain answer. This should be a list of two-tuples, the first entry is the domain name, and the second is the translated domain value:

```
profile = DNSRelayProfile('dnsrules')
profile.fixed_domain_answer.add([
    ('microsoft.com', 'foo.com'), ('microsoft2.com',)])
```

**Parameters**

**answers** (*tuple*[*str*, *str*]) – (domain\_name, translated\_domain\_name)

**Raises**

`UpdateElementFailed` – failure to add to SMC

**Returns**

None

---

**Note:** translated\_domain\_name can be none, which will cause the NGFW to return NXDomain for the specified domain.

---

**class** `smc.elements.profiles.HostnameMapping(profile)`Bases: `DNSRule`

Statically map host names, aliases for host names, and unqualified names (a host name without the domain suffix) to IPv4 or IPv6 addresses

**add**(*answers*)

Map specific hostname to specified IP address. Provide a list of two-tuples. The first entry is the hostname/s to translate (you can provide multiple comma separated values). The second entry should be the IP address to map the hostnames to:

```
profile = DNSRelayProfile('dnsrules')
profile.hostname_mapping.add([('hostname1,hostname2', '1.1.1.1')])
```

**Parameters**

**answers** (*tuple*[*str*, *str*]) – (hostnames, ipaddress), hostnames can be a comma separated list.

**Raises**

*UpdateElementFailed* – failure to add to SMC

**Returns**

None

### 14.5.8.2 SNMPAgent

**class** smc.elements.profiles.**SNMPAgent**(*name=None*, *\*\*meta*)

Bases: *Element*

Minimal implementation of SNMPAgent

### 14.5.8.3 LLDPProfile

**class** smc.core.lldp.**LLDPProfile**(*name=None*, *\*\*meta*)

Bases: *Element*

LLDP Profile represents a set of attributes used for configuring LLDP(Link Layer Discovery Protocol). LLDP information is advertised by devices at a fixed interval in the form of LLDP data units represented by TLV structures.

**property chassis\_id**

TLV field: Chassis ID. The MAC address of the first Ethernet port (Always enabled)

**classmethod create**(*name*, *transmit\_delay*, *hold\_time\_multiplier*, *system\_name*, *system\_description*, *system\_capabilities*, *management\_address*, *comment=None*)

Create a LLDPProfile. :param str name: name of TLS Profile :param int transmit\_delay: The transmit delay determines the delay between any two consecutive LLDP advertisement frames. :param int hold\_time\_multiplier: Represents the multiplier to apply to the advertisement interval. :param bool system\_name: The system name that the SNMP Agent uses :param bool system\_description: The system description that the SNMP Agent uses :param bool system\_capabilities: Capability bit-map. Depends on the interface type :param bool management\_address: Management Address IP addresses of the control interfaces :param str comment: optional comment :raises CreateElementFailed: failed to create element with reason :raises ElementNotFound: specified element reference was not found :rtype: TLSProfile

**property hold\_time\_multiplier**

Represents the multiplier to apply to the advertisement interval. The product of the advertisement interval and the hold time multiplier gives cache life time for the learned LLDP information, after which it is discarded.

**property management\_address**

TLV Field: Management Address IP addresses of the control interfaces

**property port\_description**

TLV field: Port Description. The description that the SNMP Agent uses for the interface (Always enabled)

**property port\_id**

TLV field: Port ID. The name that the SNMP Agent uses for the interface (Always enabled)

**property system\_capabilities**

TLV field: System Capabilities. Capability bit-map. Depends on the interface type

**property system\_description**

TLV field: System Description. The system description that the SNMP Agent uses

**property system\_name**

TLV field: System Name. The system name that the SNMP Agent uses

**property time\_to\_live**

TLV field: Time to Live. Automatically calculated based on transmit delay and hold time multiplier (Always enabled)

**property transmit\_delay**

The transmit delay determines the delay between any two consecutive LLDP advertisement frames.

## 14.6 Engine

**class** `smc.core.engine.Engine`(*name=None, \*\*meta*)

Bases: *Element*

An engine is the top level representation of a firewall, IPS or virtualized software.

Engine can be referenced directly and will be loaded when attributes are accessed:

```
>>> from smc.core.engine import Engine
>>> engine = Engine('testfw')
>>> print(engine.href)
http://1.1.1.1:8082/6.1/elements/single_fw/39550
```

Generically search for engines of all types:

```
>>> list(Engine.objects.all())
[Layer3Firewall(name=i-06145fc6c59a04335 (us-east-2a)), FirewallCluster(name=sg_vm),
Layer3VirtualEngine(name=ve-5), MasterEngine(name=master-eng)]
```

Or only search for specific engine types:

```
>>> from smc.core.engines import Layer3Firewall
>>> list(Layer3Firewall.objects.all())
[Layer3Firewall(name=i-06145fc6c59a04335 (us-east-2a))]
```

Engine types are defined in *smc.core.engines*.

**add\_interface**(*interface*, *\*\*kw*)

Add interface is a lower level option to adding interfaces directly to the engine. The interface is expected to be an instance of `Layer3PhysicalInterface`, `Layer2PhysicalInterface`, `TunnelInterface`, or `ClusterInterface`. The engines instance cache is flushed after this call is made to provide an updated cache after modification.

**See also:**

`smc.core.engine.interface.update_or_create`

**Parameters**

**interface** (`PhysicalInterface`, `TunnelInterface`) – instance of pre-created interface

**Returns**

None

**add\_link\_usage\_exception\_rules**(*link\_usage\_exception\_rules*)

Add link\_usage\_exception\_rules/s to this engine.

**Parameters**

**link\_usage\_exception\_rules** (`list(link_usage_exception_rules)`) – link\_usage\_exception\_rules/s to add to engine

**Raises**

`UpdateElementFailed` – failed updating engine

**Returns**

None

**add\_route**(*gateway=None*, *network=None*, *payload=None*)

Add a route to engine. Specify gateway and network. If this is the default gateway, use a network address of 0.0.0.0/0.

**Parameters**

- **gateway** (*str*) – gateway of an existing interface
- **network** (*str*) – network address in cidr format
- **payload** (*href*) – the payload to add route with href of element Example:  
 {"gateway\_ip": X.Y.Z.Z, "network\_ip": A.B.C.D} OR {"gateway\_ref": href, "network\_ref": href}

**Raises**

`EngineCommandFailed` – invalid route, possibly no network

**Returns**

None

**property adsl\_interface**

Get only adsl interfaces for this engine node.

**Raises**

`UnsupportedInterfaceType` – adsl interfaces are only supported on layer 3 engines

**Returns**

list of dict entries with href,name,type, or None

**alias\_resolving**()

Alias definitions with resolved values as defined on this engine. Aliases can be used in rules to simplify multiple object creation

```
fw = Engine('myfirewall')
for alias in fw.alias_resolving():
    print(alias, alias.resolved_value)
...
(Alias(name=$$ Interface ID 0.ip), [u'10.10.0.1'])
(Alias(name=$$ Interface ID 0.net), [u'10.10.0.0/24'])
(Alias(name=$$ Interface ID 1.ip), [u'10.10.10.1'])
```

**Returns**

generator of aliases

**Return type**

*Alias*

**property all\_vpns**

Engine level all VPN gateway information. Example:

```
>>> list_of_all_internal_gateways=engine.all_vpns
>>> first_vpn_instance= list_of_all_internal_gateways[0]
>>> first_vpn_instance.name
```

**Raises**

*UnsupportedEngineFeature* – internal gateway is only supported on layer 3 engine types.

**Returns**

list of engine internal gateways

**Return type**

List of All VPN Gateway Configuration

**property antispoofing**

Antispoofing interface information. By default is based on routing but can be modified.

```
for entry in engine.antispoofing.all():
    print(entry)
```

**Returns**

top level antispoofing node

**Return type**

*Antispoofing*

**property antivirus**

AntiVirus engine settings. Note that for virtual engines the AV settings are configured on the Master Engine. Get current status:

```
engine.antivirus.status
```

**Raises**

*UnsupportedEngineFeature* – Invalid engine type for AV

**Return type**

*AntiVirus*

**property automatic\_rules\_settings**

Represents the container for all automatic rules settings for a cluster. Example of using automatic rules settings:

```
>>> engine = Engine("testme")
>>> automatic_rules_settings=engine.automatic_rules_settings
>>> update_automatic_rules_settings(allow_auth_traffic=False, allow_no_
↳ nat=False)
>>> engine.update()
>>> engine.automatic_rules_settings.allow_auth_traffic
False
>>> engine.automatic_rules_settings.allow_listening_interfaces_to_dns_relay_port
True
```

**Return type**

AutomaticRulesSettings

**blacklist**(*src, dst, duration=3600, \*\*kw*)

Add blacklist entry to engine node by name. For blacklist to work, you must also create a rule with action “Apply Blacklist”.

**Parameters**

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or ‘any’
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or ‘any’
- **duration** (*int*) – how long to blacklist in seconds

**Raises**

*EngineCommandFailed* – blacklist failed during apply

**Returns**

None

---

**Note:** This method requires SMC version >= 6.4 and SMC version <7.0

---

since this version, “blacklist” is renamed “block\_list”

**blacklist\_bulk**(*block\_list*)

Add block list entries to the engine node in bulk. For block list to work, you must also create a rule with action “Apply Block List”. First create your block\_list entries using *smc.elements.other.Blacklist* then provide the block list to this method.

:param Blacklist block\_list : pre-configured block list entries

---

**Note:** This method requires SMC version >= 6.4 and SMC version <7.0

---

since this version, “blacklist” is renamed “block\_list”

**blacklist\_flush**()

Flush entire blacklist for engine

**Raises**

*EngineCommandFailed* – flushing blacklist failed with reason

**Returns**

None

---

**Note:** This method requires SMC version < 7.0

---

since this version, “blacklist” is renamed “block\_list”

**blacklist\_show(\*\*kw)**

New in version 0.5.6: Requires pip install smc-python-monitoring

Blacklist show requires that you install the smc-python-monitoring package. To obtain Blacklist entries from the engine you need to use this extension to plumb the websocket to the session. If you need more granular controls over the blacklist such as filtering by source and destination address, use the smc-python-monitoring package directly. Blacklist entries that are returned from this generator have a delete() method that can be called to simplify removing entries. A simple query would look like:

```
for bl_entry in engine.blacklist_show():
    print(bl_entry)
```

**Parameters**

**kw** – keyword arguments passed to blacklist query. Common setting is to pass max\_recv=20, which specifies how many “receive” batches will be retrieved from the SMC for the query. At most, 200 results can be returned in a single query. If max\_recv=5, then 1000 results can be returned if they exist. If less than 1000 events are available, the call will be blocking until 5 receives has been reached.

**Returns**

generator of results

**Return type***smc\_monitoring.monitors.blacklist.BlacklistEntry*

---

**Note:** This method requires SMC version < 7.0

---

since this version, “blacklist” is renamed “block\_list”

**block\_list(src, dst, duration=3600, \*\*kw)**

Add block\_list entry to engine node by name. For block\_list to work, you must also create a rule with action “Apply Blocklist”.

**Parameters**

- **src** – source address, with cidr, i.e. 10.10.10.10/32 or ‘any’
- **dst** – destination address with cidr, i.e. 1.1.1.1/32 or ‘any’
- **duration** (*int*) – how long to block list in seconds

**Raises***EngineCommandFailed* – block list failed during apply**Returns**

None

---

**Note:** This method requires SMC version >= 7.0

---

**block\_list\_bulk**(*block\_list*)

Add *block\_list* entries to the engine node in bulk. For *block\_list* to work, you must also create a rule with action “Apply Blocklist”. First create your *block\_list* entries using `smc.elements.other.Blocklist` then provide the *block\_list* to this method.

:param Blocklist *block\_list* : pre-configured *block\_list* entries

---

**Note:** This method requires SMC version >= 7.0

---

**block\_list\_flush**()

Flush entire block list for engine

**Raises**

`EngineCommandFailed` – flushing block list failed with reason

**Returns**

None

---

**Note:** This method requires SMC version >= 7.0

---

**block\_list\_show**(\*\**kw*)

New in version 0.5.6: Requires pip install smc-python-monitoring

Block list show requires that you install the smc-python-monitoring package. To obtain Blocklist entries from the engine you need to use this extension to plumb the websocket to the session. If you need more granular controls over the *block\_list* such as filtering by source and destination address, use the smc-python-monitoring package directly. Blocklist entries that are returned from this generator have a `delete()` method that can be called to simplify removing entries. A simple query would look like:

```
for bl_entry in engine.block_list_show():
    print(bl_entry)
```

**Parameters**

**kw** – keyword arguments passed to block list query. Common setting is to pass `max_rcv=20`, which specifies how many “receive” batches will be retrieved from the SMC for the query. At most, 200 results can be returned in a single query. If `max_rcv=5`, then 1000 results can be returned if they exist. If less than 1000 events are available, the call will be blocking until 5 receives has been reached.

**Returns**

generator of results

**Return type**

`smc_monitoring.monitors.blocklist.BlocklistEntry`

**property client\_inspection**

Client TLS Inspection settings manage certificates assigned to the engine for TLS client decryption (out-bound). In order to enable either, you must first assign certificates to the engine. Example of adding `ClientInspection` to an engine:

```
>>> engine = Engine('myfirewall')
>>> tls = ClientInspection('client.test.local')
>>> engine.client_inspection.enable(tls)
>>> engine.update()
```

**Return type***ClientInspection***property connection\_timeout**

This is definition of timeout by protocol or by TCP connection state. You can define general timeouts for removing idle connections from the state table, including non-TCP communications that are handled like connections. The timeout prevents wasting engine resources on storing information about abandoned connections. Timeouts are a normal way to clear traffic information with protocols that have no closing mechanism. Timeouts do not affect active connections. The connections are kept in the state table as long as the interval of packets within a connection is shorter than the timeouts set. Example of using idle timeout settings:

```
>>> engine = Engine("testme")
>>> connection_timeout=engine.connection_timeout
>>> connection_timeout.add('tcp_syn_seen',120)
>>> engine.connection_timeout.data
{'connection_timeout': [{'protocol': 'tcp', 'timeout': 1800}, {'protocol':
→ 'udp',
  'timeout': 50}, {'protocol': 'icmp', 'timeout': 5}, {'protocol': 'other',
→ 'timeout':
  180}, {'protocol': 'tcp_syn_seen', 'timeout': 120}]}
>>> engine.update()
>>> engine.connection_timeout.data
>>> connection_timeout.remove('tcp_syn_seen')
{'connection_timeout': [{'protocol': 'tcp', 'timeout': 1800}, {'protocol':
→ 'udp',
  'timeout': 50}, {'protocol': 'icmp', 'timeout': 5}, {'protocol': 'other',
→ 'timeout':
  180}]}

```

**Return type***IdleTimeout***property contact\_addresses**

Contact addresses are NAT addresses that are assigned to interfaces. These are used when a component needs to communicate with another component through a NAT'd connection. For example, if a firewall is known by a public address but the interface uses a private address, you would assign the public address as a contact address for that interface.

---

**Note:** Contact addresses are only supported with SMC >= 6.2.

---

Obtain all eligible interfaces for contact addresses:

```
>>> engine = Engine('dingo')
>>> for ca in engine.contact_addresses:
...     ca
...
ContactAddressNode(interface_id=11, interface_ip=10.10.10.20)
ContactAddressNode(interface_id=120, interface_ip=120.120.120.100)
ContactAddressNode(interface_id=0, interface_ip=1.1.1.1)
ContactAddressNode(interface_id=12, interface_ip=3.3.3.3)
ContactAddressNode(interface_id=12, interface_ip=17.17.17.17)

```

**See also:**[`smc.core.contact\_address`](#)

This is set to a private method because the logic doesn't make sense with respects to how this is configured under the SMC.

**Return type**[`ContactAddressCollection\(ContactAddressNode\)`](#)

```
create_internal_gateway(name, antivirus=None, auto_certificate=None, auto_site_content=None,
                        dhcp_relay=None, end_point=None, firewall=None, gateway_profile=None,
                        ssl_vpn_portal_setting=None, ssl_vpn_proxy=None, ssl_vpn_tunneling=None,
                        trust_all_cas=None, trusted_certificate_authorities=None,
                        vpn_client_mode=None, **kwargs)
```

Create internal gateway Example of creating internal gateway:

```
>>> engine.create_internal_gateway("test")
```

**Parameters**

- **name** (*str*) – Name of the internal gateway
- **antivirus** (*str*) – Antivirus
- **auto\_certificate** (*str*) – Automated RSA Certificate Management
- **auto\_site\_content** (*str*) – Indicates whether the site content is automatically generated from the routing view.
- **dhcp\_relay** (*str*) – DHCP Relay.
- **end\_point** (*str*) – List of end-points.
- **firewall** (*str*) – Firewall
- **gateway\_profile** (*str*) – Gateway Profile
- **ssl\_vpn\_portal\_setting** (*str*) – SSL VPN Settings for the Portal.
- **ssl\_vpn\_proxy** (*str*) – vpn proxy
- **ssl\_vpn\_tunneling** (*str*) – SSL VPN Settings for the VPN Client.
- **trust\_all\_cas** (*str*) – Indicates if the EndPoint trust all VPN Certificate Authorities.
- **trusted\_certificate\_authorities** (*str*) – List of trusted VPN Certificate Authorities. Valid only if the EndPoint does not trust all VPN CAs.
- **vpn\_client\_mode** (*str*) – VPN Client Mode accepted values given below: *\*no \*ipsec \*ssl \*both*

**Raises**

[`UnsupportedEngineFeature`](#) – internal gateway is only supported on layer 3 engine types

**Returns**

None

**Return type**

None

**property default\_nat**

Configure default nat on the engine. Default NAT provides automatic NAT without the requirement to add specific NAT rules. This is a more common configuration for outbound traffic. Inbound traffic will still require specific NAT rules for redirection.

**Return type**

*DefaultNAT*

**delete\_user\_authentication\_certificate()**

Delete the certificate if any is defined for this component.

**delete\_user\_authentication\_certificate\_request()**

Delete the certificate request if any is defined for this component.

**property discard\_quic\_if\_cant\_inspect**

Discard or allow QUIC if inspection is impossible

**Return type**

bool

**property dns**

Current DNS entries for the engine. Add and remove DNS entries. This resource is iterable and yields instances of `smc.core.addon.DNSEntry`. Example of adding entries:

```
>>> from smc.elements.servers import DNSServer
>>> server = DNSServer.create(name='mydnsserver', address='10.0.0.1')
>>> engine.dns.add(['8.8.8.8', server])
>>> engine.update()
'http://172.18.1.151:8082/6.4/elements/single_fw/948'
>>> list(engine.dns)
[DNSEntry(rank=0,value=8.8.8.8,ne_ref=None),
 DNSEntry(rank=1,value=None,ne_ref=DNSServer(name=mydnsserver))]
```

**Return type**

*RankedDNSAddress*

**property dns\_relay**

Enable, disable or get status for the DNS Relay Service on this engine. You must still separately configure the `smc.elements.profiles.DNSRelayProfile` that the engine references.

**Raises**

*UnsupportedEngineFeature* – unsupported feature on this engine type.

**Return type**

*DNSRelay*

**property dynamic\_routing**

Dynamic Routing entry point. Access BGP, OSPF configurations

**Raises**

*UnsupportedEngineFeature* – Only supported on layer 3 engines

**Return type**

DynamicRouting

**property endpoint\_integration**

Endpoint Integration status on engine. Note that for master engines the endpoint integrations settings are configured on Virtual Engines. Get current status:

```
engine.endpoint_integration.status
```

**Raises**

*UnsupportedEngineFeature* – Invalid engine type for file rep

**Return type**

EndpointIntegration

**export\_user\_authentication\_certificate**(filename=None)

Export the certificate if any is defined for this component.

**export\_user\_authentication\_certificate\_request**(filename=None)

Export the certificate request if any is defined for this component.

**property file\_reputation**

File reputation status on engine. Note that for virtual engines the AV settings are configured on the Master Engine. Get current status:

```
engine.file_reputation.status
```

**Raises**

*UnsupportedEngineFeature* – Invalid engine type for file rep

**Return type**

*FileReputation*

**generate\_and\_sign\_user\_authentication\_certificate**()

Generate and internally sign User Authentication certificate.

**generate\_snapshot**(filename='snapshot.zip')

Generate and retrieve a policy snapshot from the engine This is blocking as file is downloaded

**Parameters**

**filename** (*str*) – name of file to save file to, including directory path

**Raises**

*EngineCommandFailed* – snapshot failed, possibly invalid filename specified

**Returns**

None

**generate\_user\_authentication\_certificate\_request**()

Export the certificate request for the node when working external CA. This can return None if the engine type does not have a certificate request.

**Raises**

*CertificateExportError* – error exporting certificate

**Return type**

*str* or None

**property geolocation**

Return the geolocation for the given engine. This attribute requires at least SMC version >= 6.5.x. If no geolocation is assigned or the SMC is not a correct version this will return None. If setting a new geolocation, call update() after modification.

Example:

```
>>> from smc.elements.other import Geolocation
>>> engine = Engine('azure')

>>> geo = Geolocation.create(name='MyGeo', latitude='44.97997', longitude='-93.
↳26384')
>>> geo
Geolocation(name=MyGeo)
>>> engine.geolocation = geo
>>> engine.update()
>>> engine.geolocation
Geolocation(name=MyGeo)
```

**Parameters**

**value** (*Geolocation*) – Geolocation to assign engine. Can be str href or type Geolocation element.

**Return type**

*Geolocation* or None

**get\_session\_monitoring**(*sesmon\_type*, *full=True*)

Available for all SMC API Versions but only for SMC Version above 7.1 (7.1 included)

Return session monitoring for the requested session monitoring type for the engine Find all routes for engine resource:

:param sesmon\_type requested session monitoring *type*. Possible value are defined

in session\_monitoring.EngineSessionMonitoringType

**:optional param full ( default value is true ). When set to false, juste retrieve**

the log key of each entry ( timestamp, component id, event id ).

:raises EngineCommandFailed : session monitoring result cannot be retrieved :return: list of session monitoring entries : session\_monitoring.SessionMonitoringResult :rtype: SerializedIterable(Route)

Example:

```
from smc.core.session_monitoring import EngineSessionMonitoringType
engine.get_session_monitoring(EngineSessionMonitoringType.CONNECTION)
```

**property granted\_permissions**

Retrieve the access control list permissions for this engine instance.

```
>>> from smc.core.engine import Engine
>>> engine = Engine('myfirewall')
>>> for x in engine.permissions.granted_permissions:
...     print(x)
...
AccessControlList(name=ALL Elements)
AccessControlList(name=ALL Firewalls)
```

**Raises**

*UnsupportedEngineFeature* – requires SMC version >= 6.1

**Returns**

access control list permissions

**Return type**`list(AccessControlList)`**property installed\_policy**

Return the name of the policy installed on this engine. If no policy, None will be returned.

**Return type**`str` or `None`**property interface**Get all interfaces, including non-physical interfaces such as tunnel or capture interfaces. These are returned as `Interface` objects and can be used to load specific interfaces to modify, etc.

```
for interfaces in engine.interface:
    .....
```

**Return type**`InterfaceCollection`See `smc.core.interfaces.Interface` for more info**property interface\_options**

Interface options specify settings related to setting primary/ backup management, outgoing, and primary/backup heartbeat interfaces. For example, set primary management interface (this unsets it from the currently assigned interface):

```
engine.interface_options.set_primary_mgt(10)
```

Obtain the primary management interface:

```
print(engine.interface_options.primary_mgt)
```

**Return type**`InterfaceOptions`**property internal\_gateway**

Engine level VPN gateway information. This is a link from the engine to VPN level settings like VPN Client, Enabling/disabling an interface, adding VPN sites, etc. Example of adding a new VPN site to the engine's site list with associated networks:

```
>>> network = Network.get_or_create(name='mynetwork', ipv4_network='1.1.1.0/24')
Network(name=mynetwork)
>>> engine.internal_gateway.vpn_site.create(name='mynewsite', site_
↪element=[network])
VPNSite(name=mynewsite)
```

**Raises**`UnsupportedEngineFeature` – internal gateway is only supported on layer 3 engine types.**Returns**

this engines internal gateway

**Return type***InternalGateway***property known\_host\_lists**

Configure SSH known host lists on the engine. Can only be set if Sidewinder Proxy is enabled.

**Raises***MissingRequiredInput* – requires sidewinder proxy to be enabled**Return type***KnownHostLists***property l2fw\_settings**

Layer 2 Firewall Settings make it possible for a layer 3 firewall to run specified interfaces in layer 2 mode. This requires that a layer 2 interface policy is assigned to the engine and that inline\_l2fw interfaces are created.

**Raises***UnsupportedEngineFeature* – requires layer 3 engine**Return type***Layer2Settings***property lbfilter\_useports**

Load Balancing Filter use ports :raises *UnsupportedEngineFeature*: Invalid engine type :rtype: bool

**property lbfilters**

Load balancing filter list :raises *UnsupportedEngineFeature*: Invalid engine type :rtype: list *LBFilter*

**ldap\_replication(enable)**

Enable or disable LDAP replication

**Raises***EngineCommandFailed* – the LDAP replication is already enabled or disabled**Parameters****enable** (*boolean*) – True enable the LDAP replication False disable it**property link\_usage\_exception\_rules**

A collection of link\_usage\_exception\_rules

**Return type***list(LinkUsageExceptionRules)***property link\_usage\_profile**

Represent link usage profile :param value: Link usage profile to assign engine. Can be str href, or LinkUsageProfile element. :raises *UpdateElementFailed*: failure to update element :return: LinkUsage-Profile element or None

**property lldp\_profile**

It represents a set of attributes used for configuring LLDP(Link Layer Discovery Protocol). LLDP information is advertised by devices at a fixed interval in the form of LLDP data units represented by TLV structures.

**Parameters****value** – LLDP Profile to assign engine. Can be str href, or LLDPProfile element.**Raises***UpdateElementFailed* – failure to update element

**Returns**

LLDPProfile element or None

**property local\_log\_storage**

Local Log Storage Settings for not virtual engines. Example of using local log storage settings:

```

>>> engine = Engine("testme")
>>> local_log_storage=engine.local_log_storage
>>> local_log_storage.local_log_storage_activated
True
>>> local_log_storage.lls_max_time
10
>>> local_log_storage.update(lls=20_max_time)
>>> engine.update()
>>> local_log_storage=engine.local_log_storage
>>> local_log_storage.lls_max_time
20

```

:rtype LocalLogStorageSettings

**property location**

The location for this engine. May be None if no specific location has been assigned.

**Parameters**

**value** – location to assign engine. Can be name, str href, or Location element. If name, it will be automatically created if a Location with the same name doesn't exist.

**Raises**

*UpdateElementFailed* – failure to update element

**Returns**

Location element or None

**property log\_moderation**

This is the definition of Log Compression for the engine or for an interface. You can also configure Log Compression to save resources on the engine. By default, each generated Antispoofing and Discard log entry is logged separately and displayed as a separate entry in the Logs view. Log Compression allows you to define the maximum number of separately logged entries. When the defined limit is reached, a single Antispoofing log entry or Discard log entry is logged. The single entry contains information on the total number of the generated Antispoofing log entries or Discard log entries. After this, logging returns to normal and all the generated entries are once more logged and displayed separately. Example of using log moderation settings:

```

>>> engine = Engine("testme")
>>> log_moderation_obj=engine.log_moderation
>>> log_moderation_obj.get(1)["rate"]
100
>>> log_moderation_obj.get(1)["burst"]
1000
>>> log_moderation_obj.add(rate=200,burst=1100,log_event=2)
>>> engine.update(log_spooling_policy='discard')
>>> log_moderation_obj=engine.log_moderation
>>> log_moderation_obj.get(2)["rate"]
200
>>> log_moderation_obj.get(2)["burst"]
1100

```

:rtype LogModeration

**property log\_server**

Log server for this engine.

**Returns**

The specified log server

**Return type**

*LogServer*

**property loopback\_interface**

Retrieve any loopback interfaces for this engine. Loopback interfaces are only supported on layer 3 firewall types.

Retrieve all loopback addresses:

```
for loopback in engine.loopback_interface:
    print(loopback)
```

**Raises**

*UnsupportedInterfaceType* – supported on layer 3 engine only

**Return type**

*LoopbackCollection*

**property modem\_interface**

Get only modem interfaces for this engine node.

**Raises**

*UnsupportedInterfaceType*: modem interfaces are only supported on layer 3 engines

**Returns**

list of dict entries with href,name,type, or None

**property nodes**

Return a list of child nodes of this engine. This can be used to iterate to obtain access to node level operations

```
>>> print(list(engine.nodes))
[Node(name=myfirewall node 1)]
>>> engine.nodes.get(0)
Node(name=myfirewall node 1)
```

**Returns**

nodes for this engine

**Return type**

*SubElementCollection(Node)*

**property ntp\_settings**

NTP settings definition for the engine :rtype: NTPSettings

**property pending\_changes**

Pending changes provides insight into changes on an engine that are pending approval or disapproval. Feature requires SMC >= v6.2.

**Raises**

*UnsupportedEngineFeature* – SMC version >= 6.2 is required to support pending changes

**Return type***PendingChanges***property permissions**

Retrieve the permissions for this engine instance. :rtype: GrantedElementPermissions

**Note:** This method has changed in fp-NGFW-SMC-python >= 1.0.24 so need to make your script compatible with this change.” .. seealso:: `smc.administration.access_rights.GrantedElementPermissions`.

**property physical\_interface**

Returns a PhysicalInterface. This property can be used to add physical interfaces to the engine. For example:

```
engine.physical_interface.add_inline_interface(...)
engine.physical_interface.add_layer3_interface(...)
```

**Raises**

*UnsupportedInterfaceType* – engine doesn’t support this type

**Return type***PhysicalInterfaceCollection***property policy\_route**

Configure policy based routes on the engine.

```
engine.policy_route.create(
    source='172.18.2.0/24', destination='192.168.3.0/24',
    gateway_ip='172.18.2.1')
```

**Return type***PolicyRoute***query\_route**(source\_ref=None, destination\_ref=None, source\_ip=None, destination\_ip=None)

Allows querying a route for the specific supported engine Options: A. Using Query Parameters:

source\_ip: the IP Address A.B.C.D corresponding to the source query ip address. destination\_ip: the IP Address A.B.C.D corresponding to the destination query ip address

- B. Using payload to be able to specify source network element uri and/or destination network element uri.

**Find route for source to destination using ip address**

```
>>> engine = Engine('Plano')
>>> engine.query_route(source_ip='0.0.0.0', destination_ip='0.0.0.0')
[Routing(name=Interface 1,level=None,type=routing), Routing(name=net-172.31.
↳ 14.0/24,
level=None,type=routing), Routing(name=AT&T Plano Router,level=None,
↳ type=routing),
Routing(name=Any network,level=None,type=routing)]
```

**Find the route using query route with ref**

```
>>> list_of_routing = list(Host.objects.all())
>>> host1 = list_of_routing[0]
>>> host2 = list_of_routing[1]
>>> engine.query_route(source_ref=host1.href, destination_ref=host2.href)
```

#### Parameters

- **source\_ref** (*str*) – specify source network element uri
- **destination\_ref** (*str*) – destination network element uri
- **source\_ip** (*str*) – source ip address
- **destination\_ip** (*str*) – destination ip address

#### Return list(Routing)

the result pages containing the result routing.

**refresh**(*timeout=3, wait\_for\_finish=False, preserve\_connections=True, generate\_snapshot=True, \*\*kw*)

Refresh existing policy on specified device. This is an asynchronous call that will return a ‘follower’ link that can be queried to determine the status of the task.

```
poller = engine.refresh(wait_for_finish=True)
while not poller.done():
    poller.wait(5)
    print('Percentage complete {}'.format(poller.task.progress))
```

#### Parameters

- **timeout** (*int*) – timeout between queries
- **wait\_for\_finish** (*bool*) – poll the task waiting for status
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

#### Raises

*TaskRunFailed* – refresh failed, possibly locked policy

#### Return type

*TaskOperationPoller*

**remove\_alternative\_policies()**

Remove all alternative policies on engine.

**remove\_link\_usage\_exception\_rules**(*link\_usage\_exception\_rules*)

Remove a *link\_usage\_exception\_rules* from this engine.

#### Parameters

**link\_usage\_exception\_rules** (*link\_usage\_exception\_rules*) – element to remove

#### Returns

remove element if it exists and return bool

#### Return type

*bool*

**rename(*name*)**

Rename the firewall engine, nodes, and internal gateway (VPN gw)

**Returns**

None

**property routing**

Find all routing nodes within engine:

```
for routing in engine.routing.all():
    for routes in routing:
        ...
```

Or just retrieve a routing configuration for a single interface:

```
interface = engine.routing.get(0)
```

**Returns**

top level routing node

**Return type**

*Routing*

**property routing\_monitoring**

Return route table for the engine, including gateway, networks and type of route (dynamic, static). Calling this can take a few seconds to retrieve routes from the engine.

Find all routes for engine resource:

```
>>> engine = Engine('sg_vm')
>>> for route in engine.routing_monitoring:
...     route
...
Route(route_network=u'0.0.0.0', route_netmask=0, route_gateway=u'10.0.0.1',
      route_type=u'Static', dst_if=1, src_if=-1)
...
```

**Raises**

*EngineCommandFailed* – routes cannot be retrieved

**Returns**

list of route elements

**Return type**

*SerializedIterable(Route)*

**property sandbox**

Configure sandbox settings on the engine. Get current status:

```
engine.sandbox.status
```

**Raises**

*UnsupportedEngineFeature* – not supported on virtual engine

**Return type**

*Sandbox*

**property sidewinder\_proxy**

Configure Sidewinder Proxy settings on this engine. Sidewinder proxy is supported on layer 3 engines and require SMC and engine version  $\geq 6.1$ . Get current status:

```
engine.sidewinder_proxy.status
```

**Raises**

*UnsupportedEngineFeature* – requires layer 3 engine

**Return type**

*SidewinderProxy*

**property snapshots**

References to policy based snapshots for this engine, including the date the snapshot was made

**Raises**

*EngineCommandFailed* – failure downloading, or IOError

**Return type**

*SubElementCollection(Snapshot)*

**property snmp**

SNMP engine settings. SNMP is supported on all engine types, however can be enabled only on NDI interfaces (interfaces that have assigned addresses).

**Return type**

*SNMP*

**property switch\_physical\_interface**

Get only switch physical interfaces for this engine node. This is an iterable property:

```
for interface in engine.switch_physical_interface:
    ...
```

Or you can fetch a switch port interface/module directly by using the generic interface property:

```
engine.interface.get('SWP_0')
```

Or through this property directly:

```
engine.switch_physical_interface.get('SWP_0')
```

**Raises**

*UnsupportedInterfaceType* – switch interfaces are only supported on specific firewall models

**Returns**

list of dict entries with href,name,type, or None

**property tls\_inspection**

TLS Inspection settings manage certificates assigned to the engine for TLS server decryption (inbound) and TLS client decryption (outbound). In order to enable either, you must first assign certificates to the engine. Example of adding TLSServerCredentials to an engine:

```
>>> engine = Engine('myfirewall')
>>> tls = TLSServerCredential('server2.test.local')
>>> engine.tls_inspection.add_tls_credential([tls])
>>> engine.tls_inspection.server_credentials
[TLSServerCredential(name=server2.test.local)]
```

**Return type***TLSInspection***property tunnel\_interface**

Get only tunnel interfaces for this engine node.

**Raises***UnsupportedInterfaceType* – supported on layer 3 engine only**Return type***TunnelInterfaceCollection*

**upload**(*policy=None, timeout=5, wait\_for\_finish=False, preserve\_connections=True, generate\_snapshot=True, \*\*kw*)

Upload policy to engine. This is used when a new policy is required for an engine, or this is the first time a policy is pushed to an engine. If an engine already has a policy and the intent is to re-push, then use *refresh()* instead. The policy argument can use a wildcard \* to specify in the event a full name is not known:

```
engine = Engine('myfw')
task = engine.upload('Amazon*', wait_for_finish=True)
for message in task.wait():
    print(message)
```

**Parameters**

- **policy** (*str*) – name of policy to upload to engine; if None, current policy
- **wait\_for\_finish** (*bool*) – poll the task waiting for status
- **timeout** (*int*) – timeout between queries
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises***TaskRunFailed* – upload failed with reason**Return type***TaskOperationPoller*

**upload\_alternative\_slot**(*alternative\_slot=None, policy=None, timeout=5, wait\_for\_finish=False, generate\_snapshot=True, \*\*kw*)

Upload policy to engine alternative slot. This is used when multiple policies are required for an engine. If an engine already has a policy and the intent is to re-push, then use *refresh()* instead. The policy argument can use a wildcard \* to specify in the event a full name is not known:

```
engine = Engine('myfw')
task = engine.upload_alternative_slot(1, 'Amazon*', wait_for_finish=True)
for message in task.wait():
    print(message)
```

**Parameters**

- **alternative\_slot** (*int*) – Slot of policy to upload to engine(1 to 3)
- **policy** (*str*) – name of policy to upload to engine; if None, current policy
- **wait\_for\_finish** (*bool*) – poll the task waiting for status
- **timeout** (*int*) – timeout between queries
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises**

*TaskRunFailed* – upload failed with reason

**Return type**

*TaskOperationPoller*

**property url\_filtering**

Configure URL Filtering settings on the engine. Get current status:

```
engine.url_filtering.status
```

**Raises**

*UnsupportedEngineFeature* – not supported on virtual engines

**Return type**

*UrlFiltering*

**user\_authentication\_import\_certificate**(*certificate*)

Import a valid certificate. Certificate can be either a file path or a string of the certificate. If string certificate, it must include the —BEGIN CERTIFICATE— string.

**Parameters**

**certificate** (*str*) – fully qualified path or string

**Raises**

- *CertificateImportError* – failure to import cert with reason
- *IOError* – file not found, permissions, etc.

**property version**

Version of this engine. Can be none if the engine has not been initialized yet.

**Return type**

*str* or None

**property virtual\_physical\_interface**

Master Engine virtual instance only

A virtual physical interface is for a master engine virtual instance. This interface type is just a subset of a normal physical interface but for virtual engines. This interface only sets Auth\_Request and Outgoing on the interface.

To view all interfaces for a virtual engine:

```
for intf in engine.virtual_physical_interface:
    print(intf)
```

**Raises**

*UnsupportedInterfaceType* – supported on virtual engines only

**Return type**

*VirtualPhysicalInterfaceCollection*

**property virtual\_resource**

Available on a Master Engine only.

To get all virtual resources call:

```
engine.virtual_resource.all()
```

**Raises**

*UnsupportedEngineFeature* – master engine only

**Return type**

*CreateCollection(VirtualResource)*

**property vpn**

VPN configuration for the engine.

**Raises**

*UnsupportedEngineFeature*: VPN is only supported on layer 3 engines.

**Return type**

*VPN*

**property vpn\_broker\_interface**

Get only vpn broker interfaces for this engine node.

**Raises**

*UnsupportedInterfaceType* – supported on layer 3 engine only

**Return type**

*VPNBrokerInterfaceCollection*

**property vpn\_endpoint**

A VPN endpoint is an address assigned to a layer 3 interface that can be enabled to turn on VPN capabilities. As an interface may have multiple IP addresses assigned, the endpoints are returned based on the address. Endpoints are properties of the engines Internal Gateway.

**Raises**

*UnsupportedEngineFeature* – only supported on layer 3 engines

**Return type**

*SubElementCollection(InternalEndpoint)*

**property vpn\_mappings**

New in version 0.6.0: Requires SMC version >= 6.3.4

VPN policy mappings (by name) for this engine. This is a shortcut method to determine which VPN policies are used by the firewall.

**Raises**

***UnsupportedEngineFeature*** – requires a layer 3 firewall and SMC version >= 6.3.4.

**Return type**

*VPNMappingCollection(VPNMapping)*

**property wireless\_interface**

Get only wireless interfaces for this engine node.

**Raises**

***UnsupportedInterfaceType*** – wireless interfaces are only supported on layer 3 engines

**Returns**

list of dict entries with href,name,type, or None

**class** `smc.core.engine.HAForSingleEngine(data=None)`

Bases: `object`

High availability configuration on the single engine. Access through an engine reference:

```
engine.ha_settings.ha_backup_unit
engine.ha_settings.set_ha_backup_unit(pos='1234567890-1234567890')
engine.ha_settings.disable_ha_backup_unit()

engine.ha_settings.connection_sync_mode
engine.ha_settings.set_connection_sync(ha_connection_sync_interface=0,
                                       connection_sync_group=connection_sync_group_
                                       ↪name)
engine.ha_settings.disable_connection_sync()
```

When making changes to the HA configuration, any methods called that change the configuration also require that `engine.update()` is called once changes are complete. This way you can make multiple changes without refreshing the engine cache.

**Variables**

**`connection_sync_group`** (*ConnectionSynchronizationGroup*) – ConnectionSynchronizationGroup

reference for this engine. Applicable when HA mode is `connection_sync`.

**property connection\_sync\_mode**

Get connection synchronization for external high availability mode for this engine

**Returns**

str or None

**disable\_connection\_sync()**

Disable connection synchronization for external high availability on this engine.

**Returns**

None

**disable\_ha\_backup\_unit()**

Disable HA backup unit mode on this engine.

**Returns**

None

**property ha\_backup\_unit\_mode**

Get high availability backup unit mode for this engine

**Returns**

str or None

**property pos**

proof of serial for hugh availability. Applicable when HA backup unit mode is enabled.

**Return type**

str or None

**set\_connection\_sync**(*connection\_sync\_group*, *ha\_connection\_sync\_interface*)

Enable connection synchronization for external high availability on this engine. Disable high availability backup unit.

**Parameters**

- **connection\_sync\_group** (str, *ConnectionSynchronizationGroup*) – Connection-SynchronizationGroup element or str href
- **ha\_connection\_sync\_interface** (str) – ID of the interface for unicast state

synchronization :raises ElementNotFound: ConnectionSynchronizationGroup not found :return: None

**set\_ha\_backup\_unit**(*pos=None*)

Enable high availability backup unit on this engine. Disable connection synchronization for external high availability.

**Parameters****pos** (str) – proof of serial for pairing**Returns**

None

**property sync\_interface**

ID of the interface for unicast state Applicable when HA mode is connection\_sync.

**Return type**

int or None

**class** smc.core.engine.IdleTimeout(*engine*)

Bases: NestedDict

This is definition of timeout by protocol or by TCP connection state. You can define general timeouts for removing idle connections from the state table, including non-TCP communications that are handled like connections. The timeout prevents wasting engine resources on storing information about abandoned connections. Timeouts are a normal way to clear traffic information with protocols that have no closing mechanism. Timeouts do not affect active connections. The connections are kept in the state table as long as the interval of packets within a connection is shorter than the timeouts set.

**add**(*name*, *timeout=None*)

Add a timeout setting for the new protocol. :param str name: name of the protocol. :param int timeout: timeout value.

**remove**(*name*)

Remove the timeout setting for specific protocols on the engine. :param str name: name of the protocol to be removed.

**class** smc.core.engine.LBFilter(*action*, *ip\_descriptor*, *replace\_ip*, *nodeid*, *ignore\_other=False*, *nat\_enforce=False*, *use\_ipsec=False*, *use\_ports=False*)

Bases: NestedDict

This represents the Load Balancing Filter.

**property action**

Action for the filter. possible values are: none, replace, node, select\_none, replace\_offset :rtype: str

**property ignore\_other**

Tell that other entries might not be concerned.

**Return type**

bool

**property ip\_descriptor**

Represents the IPNetwork or the IPAddressRange

**Return type**

str

**property nat\_enforce**

Tells NAT to enforce translated packet headers to the same hash value to the matching packet.

**Return type**

bool

**property replace\_ip**

Address in case of replace action.

**Return type**

str

**property use\_ipsec**

Tells the engine that this entry has to be handled with special care because part of VPN

**Return type**

bool

**property use\_ports**

Defines whether to use port numbers when calculating the hash value for the packet.

**Return type**

bool

```
class smc.core.engine.LinkUsageExceptionRules(destinations=None, services=None, sources=None,
                                              isp_link_ref=None, comment=None)
```

Bases: NestedDict

**property comment**

comment. :rtype: str

**property isp\_link\_ref**

isp\_link :rtype: isp\_link

```
class smc.core.engine.LocalLogStorageSettings(engine)
```

Bases: NestedDict

Local Log Storage Settings for not virtual engines.

**property lls\_guaranteed\_free\_percent**

Minimum amount of spool space that must be left available for other uses in percentage

**property lls\_guaranteed\_free\_size\_in\_mb**

Minimum amount of spool space that must be left available for other uses in MegaBytes

**property** `lls_max_time`

The maximum amount of hours before the stored logs are deleted.

**property** `local_log_storage_activated`

Activate the Local Log Storage feature. At least one of the Guaranteed free disk partition values must be set up.

**class** `smc.core.engine.VPN(engine, internal_gateway=None)`

Bases: `object`

VPN is the top level interface to all engine based VPN settings. To enable IPSEC, SSL or SSL VPN on the engine, enable on the endpoint.

**add\_site**(*name, site\_elements=None*)

Add a VPN site with site elements to this engine. VPN sites identify the sites with protected networks to be included in the VPN. Add a network and new VPN site:

```
>>> net = Network.get_or_create(name='wireless', ipv4_network='192.168.5.0/24')
>>> engine.vpn.add_site(name='wireless', site_elements=[net])
VPNSite(name=wireless)
>>> list(engine.vpn.sites)
[VPNSite(name=dingo - Primary Site), VPNSite(name=wireless)]
```

**Parameters**

- **name** (*str*) – name for VPN site
- **site\_elements** (*list(str, Element)*) – network elements for VPN site

**Raises**

- *ElementNotFound* – if site element is not found
- *UpdateElementFailed* – failed to add vpn site

**Return type**

*VPNSite*

---

**Note:** Update is immediate for this operation.

---

**property** `gateway_certificate`

A Gateway Certificate is used by the engine for securing communications such as VPN. You can also check the expiration, view the signing CA and renew the certificate from this element.

**Returns**

GatewayCertificate

**Return type**

*list*

**property** `gateway_certificate_request`

A Gateway Certificate request is a gateway certificate that need to be signed internally or externally using external CA

**Returns**

GatewayCertificateRequest

**Return type***list***property gateway\_profile**

Gateway Profile for this VPN. This is only a valid setting on layer 3 firewalls.

**Return type***GatewayProfile***property gateway\_settings**

A gateway settings profile defines VPN specific settings related to timers such as negotiation retries (min, max) and mobike settings. Gateway settings are only present on layer 3 FW types.

**Return type***GatewaySettings*

---

**Note:** This can return None on layer 3 firewalls if VPN is not enabled.

---

**generate\_certificate**(*common\_name*, *organization*='Forcepoint', *public\_key\_algorithm*='rsa', *signature\_algorithm*='rsa\_sha\_512', *key\_length*=2048, *signing\_ca*=None, *certificate*=None)

Generate an internal gateway certificate used for VPN on this engine. Certificate request should be an instance of VPNCertificate.

**Param**

str common\_name: common name for certificate

**Param**

str organization: organization for certificate

**Parameters**

- **public\_key\_algorithm** (*str*) – public key type to use. Valid values rsa, dsa, ecdsa.
- **signature\_algorithm** (*str*) – signature algorithm. Valid values dsa\_sha\_1, dsa\_sha\_224, dsa\_sha\_256, rsa\_md5, rsa\_sha\_1, rsa\_sha\_256, rsa\_sha\_384, rsa\_sha\_512, ecdsa\_sha\_1, ecdsa\_sha\_256, ecdsa\_sha\_384, ecdsa\_sha\_512. (Default: rsa\_sha\_512)
- **key\_length** (*int*) – length of key. Key length depends on the key type. For example, RSA keys can be 1024, 2048, 3072, 4096. See SMC documentation for more details.
- **signing\_ca** (*str*, *VPNCertificateCA*) – by default will use the internal RSA CA

:param str, certificate : used directly call another \_create\_from\_cert :raises CertificateError: error generating certificate :return: GatewayCertificate

**property internal\_endpoint**

Internal endpoints to enable VPN for the engine.

**Return type***SubElementCollection(InternalEndpoint)***property loopback\_endpoint**

Internal Loopback endpoints to enable VPN for the engine.

**Return type***SubElementCollection(InternalEndpoint)*

**remove()**

Rename the internal gateway.

**Parameters**

**name** (*str*) – new name for internal gateway

**Returns**

None

**rename(name)**

Rename the internal gateway.

**Parameters**

**name** (*str*) – new name for internal gateway

**Returns**

None

**property sites**

VPN sites configured for this engine. Using sub element methods simplify fetching sites of interest:

```
engine = Engine('sg_vm')
mysite = engine.vpn.sites.get_contains('inter')
print(mysite)
```

**Return type**

*CreateCollection(VPNSite)*

**property vpn\_client**

VPN Client settings for this engine.

Alias for internal\_gateway.

**Return type**

*InternalGateway*

**class** smc.core.engine.VPNMapping(*gateway\_ref, vpn\_ref, gateway\_nodes\_usage*)

Bases: *VPNMapping*

A VPN Mapping represents Policy Based VPNs associated with this engine. This simplifies finding references where an engine is used within a VPN without iterating through existing VPNs to find the engine.

**property internal\_gateway**

Return the engines internal gateway as element

**Return type**

*InternalGateway*

**property is\_central\_gateway**

Is this engine a central gateway in the VPN policy

**Return type**

*bool*

**property is\_mobile\_gateway**

Is the engine specified as a mobile gateway in the Policy VPN configuration

**Return type**

*bool*

**property is\_satellite\_gateway**

Is this engine a satellite gateway in the VPN policy

**Return type**

`bool`

**property vpn**

The VPN policy for this engine mapping

**Return type**

`PolicyVPN`

**class** `smc.core.engine.VPNMappingCollection(vpns)`

Bases: `BaseIterable`

## 14.6.1 AddOn

Engine feature add on functionality such as default NAT, Antivirus, File Reputation, etc. These are common settings that are located under the SMC AddOn or General properties.

Property features will have a common interface allowing you to *enable*, *disable* and check *status* from the engine reference. When property features are modified, they are done so against a local copy of the server instance. To commit the change, you must call `.update()` on the engine instance.

For example, to view status of antivirus, given a specific engine:

```
engine.antivirus.status
```

Then enable or disable:

```
engine.antivirus.enable()
engine.antivirus.disable()
engine.update()
```

**..note::** Engine property settings require that you call `engine.update()` after making / queuing your changes.

### 14.6.1.1 AntiVirus

**class** `smc.core.addon.AntiVirus(engine)`

Antivirus settings for the engine. In order to use AV, you must also have DNS server addresses configured on the engine.

Enable AV, use a proxy for updates and adjust update schedule:

```
engine.antivirus.enable()
engine.antivirus.update_frequency('daily')
engine.antivirus.update_day('tu')
engine.antivirus.log_level('transient')
engine.antivirus.http_proxy('10.0.0.1', proxy_port=8080, user='foo', password=
    ↪ 'password')
engine.update()
```

**Variables**

- **antivirus\_enabled** (`bool`) – is antivirus enabled

- **antivirus\_http\_proxy** (*str*) – http proxy settings
- **antivirus\_http\_proxy\_enabled** (*bool*) – is http proxy enabled
- **antivirus\_proxy\_port** (*int*) – http proxy port
- **antivirus\_proxy\_user** (*str*) – http proxy user
- **antivirus\_update** (*str*) – how often to update
- **antivirus\_update\_day** (*str*) – if update set to weekly, which day to update
- **antivirus\_update\_time** (*int*) – time to update av signatures
- **virus\_log\_level** (*str*) – antivirus logging level

---

**Note:** You must call `engine.update()` to commit any changes.

---

### **disable()**

Disable antivirus on the engine

### **enable()**

Enable antivirus on the engine

### **http\_proxy**(*proxy*, *proxy\_port*, *user=None*, *password=None*)

New in version 0.5.7: Requires SMC and engine version  $\geq 6.4$

Set http proxy settings for Antivirus updates.

#### **Parameters**

- **proxy** (*str*) – proxy IP address
- **proxy\_port** (*str*, *int*) – proxy port
- **user** (*str*) – optional user for authentication

### **log\_level**(*level*)

Set the log level for antivirus alerting.

#### **Parameters**

**log\_level** (*str*) – none,transient,stored,essential,alert

### **property status**

Status of AV on this engine

#### **Return type**

*bool*

### **update\_day**(*day*)

Update the day when updates should occur.

#### **Parameters**

**day** (*str*) – only used if ‘weekly’ is specified. Which day or week to perform update. Valid options: mo, tu, we, th, fr, sa, su.

### **update\_frequency**(*when*)

Set the update frequency. By default this is daily.

#### **Parameters**

**antivirus\_update** (*str*) – how often to check for updates. Valid options are: ‘never’, ‘1hour’, ‘startup’, ‘daily’, ‘weekly’

### 14.6.1.2 FileReputation

**class** `smc.core.addon.FileReputation(engine)`

Configure the engine to use File Reputation capabilities.

Enable file reputation and specify outbound http proxies for queries:

```
engine.file_reputation.enable(http_proxy=[HttpProxy('myproxy')])
engine.update()
```

#### Variables

**file\_reputation\_context** (*str*) – file reputation context, either `gti_cloud_only` or `disabled`

---

**Note:** You must call `engine.update()` to commit any changes.

---

**disable()**

Disable any file reputation on the engine.

**enable**(*http\_proxy=None*)

Enable GTI reputation on the engine. If proxy servers are needed, provide a list of proxy elements.

#### Parameters

**http\_proxy** (*list(str, HttpProxy)*) – list of proxies for GTI connections

**property http\_proxy**

Return any HTTP Proxies that are configured for File Reputation.

#### Returns

list of http proxy instances

#### Return type

*list(HttpProxy)*

**property status**

Return the status of File Reputation on this engine.

#### Return type

*bool*

### 14.6.1.3 SidewinderProxy

**class** `smc.core.addon.SidewinderProxy(engine)`

Sidewinder status on this engine. Sidewinder proxy can only be enabled on specific engine types and also requires SMC and engine version  $\geq 6.1$ .

Enable Sidewinder proxy:

```
engine.sidewinder_proxy.enable()
```

---

**Note:** You must call `engine.update()` to commit any changes.

---

**disable()**

Disable Sidewinder proxy on the engine

**enable()**

Enable Sidewinder proxy on the engine

**property status**

Status of Sidewinder proxy on this engine

**Return type**

`bool`

#### 14.6.1.4 UrlFiltering

**class** `smc.core.addon.UrlFiltering(engine)`

Enable URL Filtering on the engine.

Enable Url Filtering with next hop proxies:

```
engine.url_filtering.enable(http_proxy=[HttpProxy('myproxy')])
engine.update()
```

Disable Url Filtering:

```
engine.url_filtering.disable()
engine.update()
```

---

**Note:** You must call `engine.update()` to commit any changes.

---

**disable()**

Disable URL Filtering on the engine

**enable**(*http\_proxy=None*)

Enable URL Filtering on the engine. If proxy servers are needed, provide a list of HTTPProxy elements.

**Parameters**

**http\_proxy** (*list(str, HttpProxy)*) – list of proxies for GTI connections

**property http\_proxy**

Return any HTTP Proxies that are configured for Url Filtering.

**Returns**

list of http proxy instances

**Return type**

`list(HttpProxy)`

**property status**

Return the status of URL Filtering on the engine

**Return type**

`bool`

#### 14.6.1.5 ZTNAConnector

**class** `smc.core.addon.ZTNAConnector(engine)`

Enable ZTNA Connector on the engine.

`engine.ztna_connector.enable(bgkey="xxx:yyy:zzz", datacenter="ddd") engine.update()`

Disable ZTNA Connector:

`engine.ztna_connector.disable() engine.update()`

---

**Note:** You must call `engine.update()` to commit any changes.

---

**property** `auto_update`

get ztna auto\_update or None

**property** `bgkey`

get ztna installation key or None

**property** `datacenter`

get ztna datacenter or None

**disable()**

Disable ZTNA Connector on the engine.

**enable(bgkey, datacenter, auto\_update)**

Enable ZTNA Connector on the engine.

**Parameters**

- **bgkey** (*str*) – connector installer key
- **datacenter** (*str*) – datacenter accessed via this connector
- **auto\_update** (*bool*) – install automatically the latest version

**property** `status`

Return the status (enabled/disabled) of ZTNA Connector on the engine

**Return type**

*bool*

#### 14.6.1.6 ClientInspection

**class** `smc.core.addon.ClientInspection(engine)`

**property** `ca_for_signing`

Return the Client Protection Certificate Authority assigned to this engine. The CA is used to provide decryption services to outbound client connections.

**Return type**

*ClientProtectionCA* or None if not defined

**disable()**

Disable client decryption.

**Returns**

None

**enable**(*ca\_for\_signing*, *tls\_trusted\_ca\_tags*=None, *tls\_trusted\_cas*=None)

Enable client decryption. Provide a valid client protection CA that the engine will use to decrypt.

**Parameters**

- **ca\_for\_signing** – *tls\_signing\_certificate\_authority*
- **tls\_trusted\_ca\_tags** – optional *trusted\_ca\_tag* array
- **tls\_trusted\_cas** – optional *tls\_certificate\_authority* array

**Returns**

None

**property status**

Whether client based decryption is enabled or disabled.

**Return type**

bool

**property tls\_trusted\_ca\_tags**

Return the TLS trusted CA tags.

**Return type**

TrustedCATag array or [] if not defined

**property tls\_trusted\_cas**

Return the TLS trusted CAs.

**Return type**

TLSCertificateAuthority array or [] if not defined

#### 14.6.1.7 Sandbox

**class** `smc.core.addon.Sandbox`(*engine*)

Engine based sandbox settings. Sandbox can be configured for local (on prem) or cloud based sandbox. To create file filtering policies that use sandbox, you must first enable it and provide license keys on the engine.

Enable cloud sandbox on the engine, specifying a proxy for outbound connections:

```
engine.sandbox.enable(
    license_key='123',
    license_token='456',
    http_proxy=[HttpProxy('myproxy')])
```

---

**Note:** You must call `engine.update()` to commit any changes.

---

**disable**()

Disable the sandbox on this engine.

**enable**(*license\_key*, *license\_token*, *sandbox\_type*='cloud\_sandbox', *service*='Automatic', *http\_proxy*=None, *sandbox\_data\_center*='Automatic')

Enable sandbox on this engine. Provide a valid license key and license token obtained from your engine licensing. Requires SMC version >= 6.3.

---

**Note:** Cloud sandbox is a feature that requires an engine license.

---

**Parameters**

- **license\_key** (*str*) – license key for specific engine
- **license\_token** (*str*) – license token for specific engine
- **sandbox\_type** (*str*) – ‘local\_sandbox’, ‘cloud\_sandbox’, ‘forcepoint\_sandbox’ or ‘atd’
- **service** (*str*, *SandboxService*) – a sandbox service element from SMC. The service defines which location the engine is in and which data centers to use. The default is to use the ‘US Data Centers’ profile if undefined.
- **sandbox\_data\_center** (*str*, *SandboxDataCenter*) – sandbox data center to use if the service specified does not exist. Requires SMC >= 6.4.3

**Returns**

None

**property http\_proxy**

Return any HTTP Proxies that are configured for Sandbox.

**Returns**

list of http proxy instances

**Return type**

*list(HttpProxy)*

**property status**

Status of sandbox on this engine

**Return type**

*bool*

### 14.6.1.8 TLSInspection

**class** `smc.core.addon.TLSInspection(engine)`

TLS Inspection settings control settings for doing inbound TLS decryption and outbound client TLS decryption. This provides an interface to manage TLSServerCredentials and TLSClientCredentials assigned to the engine.

---

**Note:** You must call `engine.update()` to commit any changes.

---

**add\_tls\_credential(credentials)**

Add a list of TLSServerCredential to this engine. TLSServerCredentials can be in element form or can also be the href for the element.

**Parameters**

**credentials** (*list(str, TLSServerCredential)*) – list of pre-created TLSServerCredentials

**Returns**

None

**remove\_tls\_credential(credentials)**

Remove a list of TLSServerCredentials on this engine.

**Parameters**

**credentials** (*list(str, TLSServerCredential)*) – list of credentials to remove from the engine

**Returns**

None

**property server\_credentials**

Return a list of assigned (if any) TLSServerCredentials assigned to this engine.

**Return type***list(TLSServerCredential)*

## 14.6.2 Dynamic Routing

Represents classes responsible for configuring dynamic routing protocols

### 14.6.2.1 OSPF

For more information on creating OSPF elements and enabling on a layer 3 engine:

**See also:***smc.routing.ospf*

### 14.6.2.2 BGP

For more information on creating BGP elements and enabling on a layer 3 engine:

**See also:***smc.routing.bgp*

## 14.6.3 General

### 14.6.3.1 DefaultNAT

**class** `smc.core.general.DefaultNAT(engine)`

Default NAT on the engine is used to automatically create NAT configurations based on internal routing. This simplifies the need to create specific NAT rules, primarily for outbound traffic.

---

**Note:** You must call `engine.update()` to commit any changes.

---

**disable()**

Disable default NAT on this engine

**enable()**

Enable default NAT on this engine

**property status**

Status of default nat on the engine.

**Return type***bool*

### 14.6.3.2 RankedDNSAddress

**class** `smc.core.general.RankedDNSAddress(entries)`

A RankedDNSAddress represents a list of DNS entries used as a ranked list to provide an ordered way to perform DNS queries. DNS entries can be added as raw IP addresses, or as elements of type `smc.elements.network.Host`, `smc.elements.servers.DNSServer` or a `dynamic_interface_alias` (or combination of all). This is an iterable class yielding namedtuples of type `DNSEntry`.

Normal access is done through an engine reference:

```
>>> list(engine.dns)
[DNSEntry(rank=0,value=8.8.8.8,ne_ref=None),
 DNSEntry(rank=1,value=None,ne_ref=DNSServer(name=mydnsserver))]
```

```
>>> engine.dns.append(['8.8.8.8', '9.9.9.9'])
>>> engine.dns.prepend(['1.1.1.1'])
>>> engine.dns.remove(['8.8.8.8', DNSServer('mydnsserver')])
```

---

**Note:** You must call `engine.update()` to commit any changes.

---

**append(values)**

Add DNS entries to the engine at the end of the existing list (if any). A DNS entry can be either a raw IP Address, or an element of type `smc.elements.network.Host` or `smc.elements.servers.DNSServer`.

**Parameters**

**values** (*list*) – list of IP addresses, Host and/or DNSServer elements.

**Returns**

None

---

**Note:** If the DNS entry added already exists, it will not be added. It's not a valid configuration to enter the same DNS IP multiple times. This is also true if the element is assigned the same address as a raw IP address already defined.

---

**prepend(values)**

Prepend DNS entries to the engine at the beginning of the existing list (if any). A DNS entry can be either a raw IP Address, or an element of type `smc.elements.network.Host` or `smc.elements.servers.DNSServer`.

**Parameters**

**values** (*list*) – list of IP addresses, Host and/or DNSServer elements.

**Returns**

None

**remove(values)**

Remove DNS entries from this ranked DNS list. A DNS entry can be either a raw IP Address, or an element of type `smc.elements.network.Host` or `smc.elements.servers.DNSServer`.

**Parameters**

**values** (*list*) – list of IP addresses, Host and/or DNSServer elements.

**Returns**

None

**class** `smc.core.general.DNSEntry`(*rank*, *value=None*, *ne\_ref=None*)

DNSEntry represents a single DNS entry within an engine DNSAddress list.

#### Variables

- **value** (*str*) – IP address value of this entry (None if type Element is used)
- **rank** (*int*) – order rank for the entry
- **ne\_ref** (*str*) – network element href of entry. Use element property to resolve to type Element.
- **element** (*Element*) – If the DNS entry is an element type, this property will returned a resolved version of the ne\_ref field.

### 14.6.3.3 DNS Relay

**class** `smc.core.general.DNSRelay`(*engine*)

DNS Relay allows the engine to provide DNS caching or specific host, IP and domain replies to clients. It can also be used to sinkhole specific DNS requests.

See also:

`smc.elements.profiles.DNSRelayProfile`

**disable**()

Disable DNS Relay on this engine

**Returns**

None

**enable**(*interface\_id*, *dns\_relay\_profile=None*)

Enable the DNS Relay service on this engine.

**Parameters**

- **interface\_id** (*int*) – interface id to enable relay
- **dns\_relay\_profile** (*str*, *DNSRelayProfile*) – DNSRelayProfile element or str href

**Raises**

- *EngineCommandFailed* – interface not found
- *ElementNotFound* – profile not found

**Returns**

None

**property status**

Status of DNS Relay on this engine.

**Return type**

*bool*

#### 14.6.3.4 SNMP

**class** `smc.core.general.SNMP(engine)`

SNMP configuration details for applying SNMP on an engine. SNMP requires at minimum an assigned SNMP-Agent configuration which defines the SNMP specific settings (version, community string, etc). You can also define specific interfaces to enable SNMP on. By default, if no addresses are specified, SNMP will be defined on all interfaces.

**See also:**

[`smc.elements.profiles.SNMPAgent`](#)

**property agent**

The SNMP agent profile used for this engine.

**Return type**

[`SNMPAgent`](#)

**disable()**

Disable SNMP on this engine. You must call *update* on the engine for this to take effect.

**Returns**

None

**enable(snmpp\_agent, snmp\_location=None, snmp\_interface=None, snmp\_engine\_id=None)**

Enable SNMP on the engine. Specify a list of interfaces by ID to enable only on those interfaces. Only interfaces that have NDI's are supported.

Example of adding SNMP on a port group interface:

```
engine = Engine('azure')
engine.snmp.enable(SNMPAgent('myagent'), snmp_interface=['SWP_0.1'])
engine.update()
```

**Parameters**

- **snmp\_agent** (*str*, [Element](#)) – the SNMP agent reference for this engine
- **snmp\_location** (*str*) – the SNMP location identifier for the engine
- **snmp\_interface** (*list*) – list of interface IDs to enable SNMP
- **snmp\_engine\_id** (*str*) – Specify snmp engine id. If None the it will be automatic.

**Raises**

- [ElementNotFound](#) – unable to resolve snmp\_agent
- [InterfaceNotFound](#) – specified interface by ID not found

**property interface**

Return a list of physical interfaces that the SNMP agent is bound to.

**Return type**

[list\(PhysicalInterface\)](#)

**property location**

Return the SNMP location string

**Return type**

[str](#)

**property status**

Status of SNMP on this engine

**Return type**

`bool`

**update\_configuration(\*\*kwargs)**

Update the SNMP configuration using any kwargs supported in the *enable* constructor. Return whether a change was made. You must call update on the engine to commit any changes.

**Parameters**

**kwargs** (*dict*) – keyword arguments supported by enable constructor

**Return type**

`bool`

**14.6.3.5 Layer2Settings****class smc.core.general.Layer2Settings(engine)**

Layer 2 Settings are only applicable on Layer 3 Firewall engines that want to run specific interfaces in layer 2 mode. This requires that a Layer 2 Interface Policy is applied to the engine. You can also set connection tracking and bypass on overload settings for these interfaces as well.

Set policy for the engine:

```
engine.l2fw_settings.enable(InterfacePolicy('mylayer2'))
```

**Variables**

- **bypass\_overload\_traffic** (*bool*) – whether to bypass traffic on overload
- **tracking\_mode** (*str*) – connection tracking mode

---

**Note:** You must call engine.update() to commit any changes.

---

**Warning:** This feature requires SMC and engine version >= 6.3

**bypass\_on\_overload(value)**

Set the l2fw settings to bypass on overload.

**Parameters**

**value** (*bool*) – boolean to indicate bypass setting

**Returns**

None

**connection\_tracking(mode)**

Set the connection tracking mode for these layer 2 settings.

**Parameters**

**mode** (*str*) – normal, strict, loose

**Returns**

None

**disable()**

Disable the layer 2 interface policy

**enable(policy)**

Set a layer 2 interface policy.

**Parameters**

**policy** (*str*, *Element*) – an InterfacePolicy or str href

**Raises**

- *LoadPolicyFailed* – Invalid policy specified
- *ElementNotFound* – InterfacePolicy not found

**Returns**

None

**property policy**

Return the InterfacePolicy for this layer 3 firewall.

**Return type**

*InterfacePolicy*

#### 14.6.3.6 NTPSettings

**class** smc.core.general.NTPSettings(engine=None)

This represents the definition of NTP settings.

**classmethod** create(ntp\_enable, ntp\_servers)

Create a new definition of NTPSettings :param bool ntp\_enable: is NTP enabled :param list(NTPServer) ntp\_servers: define one or more NTP Servers.

**property ntp\_enable**

Is NTP enabled? (required)

**Return type**

*bool*

**property ntp\_servers**

Define one or more NTP Servers.

**Return type**

*list(NTPServer)*

#### 14.6.4 VPN

Provisioning a firewall for VPN consists of the following steps:

- Enable VPN an interface (*InternalEndpoint*)
- Optionally add VPN sites with protected networks

---

**Note:** By default Forcepoint NGFW Engine's provide a capability that allows the protected VPN networks to be identified based on the routing table.

---

It is still possible to override this setting and add your own custom VPN sites as needed.

Once the firewall has VPN enabled, you must also assign the NGFW Engine to a specified Policy VPN as a central or satellite gateway.

The entry point for enabling the VPN on an engine is under the engine resource `smc.core.engine.Engine.vpn`.

Enabling IPSEC on an interface is done by accessing the engine resource and finding the correct *InternalEndpoint* for which to enable the VPN. Internal Endpoints are not exactly interface maps, instead they identify all addresses on a given firewall capable for running VPN. It is possible for a single interface to have more than one internal endpoint if the interface has multiple IP addresses assigned.

```
>>> from smc.core.engine import Engine
>>> engine = Engine('myfirewall')
>>> for ie in engine.vpn.internal_endpoint:
...     ie
...
InternalEndpoint(name=6.6.6.6)
InternalEndpoint(name=10.10.0.1)
InternalEndpoint(name=11.11.11.11)
InternalEndpoint(name=4.4.4.4)
InternalEndpoint(name=10.10.10.1)
```

Notice that internal endpoints are referenced by their IP address and not their interface. The interface is available as an attribute on the endpoint to make it easier to find the correct interface:

```
>>> for ie in engine.vpn.internal_endpoint:
...     print(ie, ie.interface_id)
...
(InternalEndpoint(name=6.6.6.6), u'6')
(InternalEndpoint(name=10.10.0.1), u'0')
(InternalEndpoint(name=11.11.11.11), u'11')
(InternalEndpoint(name=4.4.4.4), u'2.200')
(InternalEndpoint(name=10.10.10.1), u'1')
```

If I want to enable VPN on interface 0, you can obtain the right endpoint and enable:

```
>>> for ie in engine.vpn.internal_endpoint:
...     if ie.interface_id == '0':
...         ie.ipsec_vpn = True
```

---

**Note:** Once you've enabled the interface for VPN, you must also call `engine.update()` to commit the change.

---

The second step (optional) is to add VPN sites to the firewall. VPN Sites define a group of protected networks that can be applied to the VPN.

For example, add a new VPN site called wireless with a new network element that we'll create beforehand.

```
>>> net = Network.get_or_create(name='wireless', ipv4_network='192.168.5.0/24')
>>> engine.vpn.add_site(name='wireless', site_elements=[net])
VPNSite(name=wireless)
>>> list(engine.vpn.sites)
[VPNSite(name=dingo - Primary Site), VPNSite(name=wireless)]
```

Once the engine is enabled for VPN, see `smc.vpn.policy.PolicyVPN` for information on how to create a PolicyVPN and add engines.

#### 14.6.4.1 InternalEndpoint

**class** smc.core.engine.InternalEndpoint(\*\*meta)

Bases: *SubElement*

An Internal Endpoint is an interface mapping that enables VPN on the associated interface. This also defines what type of VPN to enable such as IPSEC, SSL VPN, or SSL VPN Portal.

To see all available internal endpoint (VPN gateways) on a particular engine, use an engine reference:

```
>>> engine = Engine('sg_vm')
>>> for e in engine.vpn.internal_endpoint:
...     print(e)
...
InternalEndpoint(name=10.0.0.254)
InternalEndpoint(name=172.18.1.254)
```

You can also retrieve an internal endpoint directly and operate on it, for example, enabling it as a VPN endpoint:

```
engine = Engine('sg_vm')
my_interface = engine.vpn.internal_endpoint.get_exact('10.0.0.254')
my_interface.update(enabled=True)
```

Multiple attributes can be updated by calling *update*:

```
my_interface.update(enabled=True, ipsec_vpn=True, force_nat_t=True, ssl_vpn_
↪portal=False, ssl_vpn_tunnel=False)
```

Available attributes:

##### Variables

- **enabled** (*bool*) – enable this interface as a VPN endpoint (default: False)
- **nat\_t** (*bool*) – enable NAT-T (default: False)
- **force\_nat\_t** (*bool*) – force NAT-T (default: False)
- **ssl\_vpn\_portal** (*bool*) – enable SSL VPN portal on the interface (default: False)
- **ssl\_vpn\_tunnel** (*bool*) – enable SSL VPN tunnel on the interface (default: False)
- **ipsec\_vpn** (*bool*) – enable IPSEC VPN on the interface (default: False)
- **udp\_encapsulation** (*bool*) – Allow UDP encapsulation (default: False)
- **balancing\_mode** (*str*) – VPN load balancing mode. Valid options are: ‘standby’, ‘aggregate’, ‘active’ (default: ‘active’)

##### property interface\_id

Interface ID for this VPN endpoint

##### Returns

str interface id

##### property name

Get the name from deducted name

##### property physical\_interface

Physical interface for this endpoint.

**Return type**  
*PhysicalInterface*

#### 14.6.4.2 InternalGateway

**class** smc.core.engine.InternalGateway(\*\*meta)

Bases: *SubElement*

InternalGateway represents the VPN Client configuration endpoint on the NGFW. Settings under Internal Gateway reflect client settings such as requiring antivirus, windows firewall and setting the VPN client mode.

View settings through an engine reference:

```
>>> engine = Engine('dingo')
>>> vpn = engine.vpn
>>> vpn.name
u'dingo Primary'
>>> vpn.vpn_client.firewall
False
>>> vpn.vpn_client.antivirus
False
>>> vpn.vpn_client.vpn_client_mode
u'ipsec'
```

Introduced all\_vpns property to get list all vpn instances, Each vpn instance associated only one internal gateway to make code backward compatible.

```
>>> list_of_all_internal_gateways=engine.all_vpns
>>> first_vpn_instance= list_of_all_internal_gateways[0]
>>> first_vpn_instance.name
u'dingo Primary'
>>> first_vpn_instance.vpn_client.firewall
False
>>> first_vpn_instance.vpn_client.antivirus
False
>>> first_vpn_instance.vpn_client.vpn_client_mode
u'ipsec'
```

Enable client AV and windows FW:

```
engine.vpn.vpn_client.update(
    firewall=True, antivirus=True)
```

##### Variables

- **firewall** (*bool*) – require windows firewall
- **antivirus** (*bool*) – require client antivirus
- **vpn\_client\_mode** (*str*) –

##### property internal\_endpoint

Internal endpoints to enable VPN for the engine.

**Return type**  
*SubElementCollection(InternalEndpoint)*

`remove()`

Remove Internal Gateway from this engine.

## 14.6.5 Interfaces

Represents classes responsible for configuring interfaces on engines

### 14.6.5.1 InterfaceCollections

Changed in version 0.7.0.

Collections classes for interfaces provide searching and methods to simplify creation based on interface types.

You can iterate any interface type by specifying the type:

```
>>> for interface in engine.tunnel_interface:
...     interface
...
TunnelInterface(name=Tunnel Interface 1008)
TunnelInterface(name=Tunnel Interface 1003)
TunnelInterface(name=Tunnel Interface 1000)
```

Or iterate all interfaces which will also return their types:

```
>>> for interface in engine.interface:
...     interface
...
Layer3PhysicalInterface(name=Interface 3)
TunnelInterface(name=Tunnel Interface 1000)
Layer3PhysicalInterface(name=Interface 61)
Layer3PhysicalInterface(name=Interface 56)
Layer3PhysicalInterface(name=Interface 15)
Layer2PhysicalInterface(name=Interface 7 (Capture))
ModemInterfaceDynamic(name=Modem 0)
TunnelInterface(name=Tunnel Interface 1030)
SwitchPhysicalInterface(name=Switch 0)
...
```

Accessing interface methods for creating interfaces can also be done in multiple ways. The simplest is to use an engine reference to use this collection. The engine reference specifies the type of interface and indicates how it will be created for the engine.

For example, creating an interface on a virtual engine:

```
engine.virtual_physical_interface.add_layer3_interface(
    interface_id=1,
    address='14.14.14.119',
    network_value='14.14.14.0/24',
    comment='my comment',
    zone_ref='myzone')
```

The helper methods use the interface API to create the interface that is then submitted to the engine. You can optionally create the interface manually using the API which provides more customization capabilities.

Example of creating a VirtualPhysicalInterface for a virtual engine manually:

```
payload = {'comment': 'comment on this interface',
          'interfaces': [{ 'nodes': [{ 'address': '13.13.13.13',
                                       'network_value': '13.13.13.0/24' }]}]}

vinterface = VirtualPhysicalInterface(interface_id=1, **payload)
```

Pass this to `update_or_create` in the event that you want to potentially modify an existing interface should the same interface ID exist:

```
engine.virtual_physical_interface.update_or_create(vinterface)
```

Or create a new interface (this will fail if the interface exists):

```
engine.add_interface(vinterface)
```

Collections also provide a simple helper when you want to provide a pre-configured interface and apply an `update_or_create` logic. In the update or create case, if the interface exists any fields that have changed will be updated. If the interface does not exist it is created. Provide *with\_status* to obtain the interface and status of the operation. The update or create will return a tuple of (Interface, modified, created), where created and modified are booleans indicating the operations performed:

```
>>> from smc.core.engine import Engine
>>> from smc.core.interfaces import Layer3PhysicalInterface
>>> engine = Engine('myfw')
>>> interface = engine.interface.get(0)
>>> interface
Layer3PhysicalInterface(name=Interface 0)
>>> interface.addresses
[(u'11.11.11.11', u'11.11.11.0/24', u'0')]
>>> myinterface = Layer3PhysicalInterface(interface_id=0,
interfaces=[{'nodes': [{ 'address': '66.66.66.66', 'network_value': '66.66.66.0/24' }]}],
comment='changed today')
...
>>> interface, modified, created = engine.physical_interface.update_or_
↳ create(myinterface)
>>> interface
Layer3PhysicalInterface(name=Interface 0)
>>> modified
True
>>> created
False
>>> interface.addresses
[(u'66.66.66.66', u'66.66.66.0/24', u'0')]
>>> interface.comment
u'changed today'
```

```
class smc.core.collection.InterfaceCollection(engine, rel='interfaces')
```

Bases: *BaseIterable*

An interface collection provides top level search capabilities to iterate or get interfaces of the specified type. This also delegates all 'add' methods of an interface to the interface type specified. Collections are returned from an engine reference and not called directly.

For example, you can use this to obtain all interfaces of a given type from an engine:

```
>>> for interface in engine.interface.all():
...     print(interface.name, interface.addresses)
('Tunnel Interface 2001', [('169.254.9.22', '169.254.9.20/30', '2001')])
('Tunnel Interface 2000', [('169.254.11.6', '169.254.11.4/30', '2000')])
('Interface 2', [('192.168.1.252', '192.168.1.0/24', '2')])
('Interface 1', [('10.0.0.254', '10.0.0.0/24', '1')])
('Interface 0', [('172.18.1.254', '172.18.1.0/24', '0')])
```

Or only physical interface types:

```
for interface in engine.physical_interfaces:
    print(interface)
```

Get switch interfaces and associated port groups:

```
for interface in engine.switch_physical_interface:
    print(interface, interface.port_groups)
```

Get a specific interface directly:

```
engine.interface.get(10)
```

Switch interface direct fetching must include the 'SWP\_' prefix as well. To get switch interface 0:

```
engine.interface.get('SWP_0')
```

You can also get port groups directly similar to fetching VLANs:

```
engine.switch_physical_interface.get('SWP_0.1')
```

Or use collection helpers to create interfaces:

```
engine.physical_interface.add(2)
engine.physical_interface.add_layer3_interface(...)
...
```

---

**Note:** This can raise `UnsupportedInterfaceType` for unsupported engine types based on the interface context.

---

### **add\_layer3\_vlan\_interface(\*args, \*\*kwargs)**

Add a Layer 3 VLAN interface. Optionally specify an address and network if assigning an IP to the VLAN. This method will also assign an IP address to an existing VLAN, or add an additional address to an existing VLAN. This method may commonly be used on a Master Engine to create VLANs for virtual firewall engines.

Example of creating a VLAN and passing kwargs to define a DHCP server service on the VLAN interface:

```
engine = Engine('engine1')
engine.physical_interface.add_layer3_vlan_interface(interface_id=20, vlan_id=20,
    address='20.20.20.20', network_value='20.20.20.0/24', comment='foocomment',
    dhcp_server_on_interface={
        'default_gateway': '20.20.20.1',
        'default_lease_time': 7200,
        'dhcp_address_range': '20.20.20.101-20.20.20.120',
```

(continues on next page)

(continued from previous page)

```
'dhcp_range_per_node': [],
'primary_dns_server': '8.8.8.8'})
```

**Parameters**

- **interface\_id** (*str*, *int*) – interface identifier
- **vlan\_id** (*int*) – vlan identifier
- **address** (*str*) – optional IP address to assign to VLAN
- **network\_value** (*str*) – network cidr if address is specified. In format: 10.10.10.0/24.
- **zone\_ref** (*str*) – zone to use, by name, href, or Zone
- **comment** (*str*) – optional comment for VLAN level of interface
- **virtual\_mapping** (*int*) – virtual engine mapping id See [smc.core.engine.VirtualResource.vfw\\_id](#)
- **virtual\_resource\_name** (*str*) – name of virtual resource See [smc.core.engine.VirtualResource.name](#)
- **kw** (*dict*) – keyword arguments are passed to top level of VLAN interface, not the base level physical interface. This is useful if you want to pass in a configuration that enables the DHCP server on a VLAN for example.

**Raises**

[EngineCommandFailed](#) – failure creating interface

**Returns**

None

**get**(*interface\_id*)

Get the interface by id, if known. The interface is retrieved from the top level Physical or Tunnel Interface. If the interface is an inline interface, you can specify only one of the two inline pairs and the same interface will be returned.

If interface type is unknown, use `engine.interface` for retrieving:

```
>>> engine = Engine('sg_vm')
>>> intf = engine.interface.get(0)
>>> print(intf, intf.addresses)
(PhysicalInterface(name=Interface 0), [('172.18.1.254', '172.18.1.0/24', '0')])
```

Get an inline interface:

```
>>> intf = engine.interface.get('2-3')
```

**Note:** For the inline interface example, you could also just specify '2' or '3' and the fetch will return the pair.

**Parameters**

**interface\_id** (*str*, *int*) – interface ID to retrieve

**Raises**

[InterfaceNotFound](#) – invalid interface specified

**Returns**

interface object by type (Physical, Tunnel, VlanInterface)

**update\_or\_create**(*interface*)

Collections class update or create method that can be used as a shortcut to updating or creating an interface. The interface must first be defined and provided as the argument. The interface method must have an *update\_interface* method which resolves differences and adds as necessary.

**Parameters**

**interface** (*Interface*) – an instance of an interface type, either PhysicalInterface, TunnelInterface or SwitchPhysicalInterface

**Raises**

- *EngineCommandFailed* – Failed to create new interface
- *UpdateElementFailed* – Failure to update element with reason

**Return type**

tuple

**Returns**

A tuple with (Interface, modified, created), where created and modified are booleans indicating the operations performed

**class** smc.core.collection.**LoopbackCollection**(*engine*)

Bases: *BaseIterable*

An loopback collection provides top level search capabilities to iterate or get loopback interfaces from a given engine.

All loopback interfaces can be fetched from the engine:

```
>>> engine = Engine('dingo')
>>> for lb in engine.loopback_interface:
...     lb
...
LoopbackInterface(address=172.20.1.1, nodeid=1, rank=1)
LoopbackInterface(address=172.31.1.1, nodeid=1, rank=2)
```

Or directly from the nodes:

```
>>> for node in engine.nodes:
...     for lb in node.loopback_interface:
...         lb
...
LoopbackInterface(address=172.20.1.1, nodeid=1, rank=1)
LoopbackInterface(address=172.31.1.1, nodeid=1, rank=2)
```

**get**(*address*)

Get a loopback address by it's address. Find all loopback addresses by iterating at either the node level or the engine:

```
loopback = engine.loopback_interface.get('127.0.0.10')
```

**Parameters**

**address** (*str*) – ip address of loopback

**Raises***InterfaceNotFound* – invalid interface specified**Return type***LoopbackInterface***class** smc.core.collection.PhysicalInterfaceCollection(*engine*)Bases: *InterfaceCollection*

PhysicalInterface Collection provides an interface to retrieving existing interfaces and helper methods to shortcut the creation of an interface.

**add**(\*args, \*\*kwargs)

Add single physical interface with interface\_id. Use other methods to fully add an interface configuration based on engine type. Virtual mapping and resource are only used in Virtual Engines.

**Parameters**

- **interface\_id** (*str*, *int*) – interface identifier
- **virtual\_mapping** (*int*) – virtual firewall id mapping See *smc.core.engine.VirtualResource.vfw\_id*
- **virtual\_resource\_name** (*str*) – virtual resource name See *smc.core.engine.VirtualResource.name*
- **lldp\_mode** (*str*) – disabled, receive\_only, send\_and\_receive, send\_only

**Raises***EngineCommandFailed* – failure creating interface**Returns**

None

**add\_capture\_interface**(*interface\_id*, *logical\_interface\_ref*, *inspect\_unspecified\_vlans=True*, *zone\_ref=None*, *comment=None*)

Add a capture interface. Capture interfaces are supported on Layer 2 engine and IPS engines.

**..note::** Capture interface are supported on Layer 3 engine/clusters for NGFW engines version >= 6.3 and SMC >= 6.3.

**Parameters**

- **interface\_id** (*str*, *int*) – interface identifier
- **logical\_interface\_ref** (*str*) – logical interface name, href or LogicalInterface. If None, 'default\_eth' logical interface will be used.
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone

**Raises***EngineCommandFailed* – failure creating interface**Returns**

None

See *smc.core.sub\_interfaces.CaptureInterface* for more information

**add\_cluster\_interface\_on\_master\_engine**(*interface\_id*, *macaddress*, *nodes*, *zone\_ref=None*, *vlan\_id=None*, *comment=None*)

Add a cluster address specific to a master engine. Master engine clusters will not use “CVI” interfaces like normal layer 3 clusters, instead each node has a unique address and share a common macaddress. Adding multiple addresses to an interface is not supported with this method.

#### Parameters

- **interface\_id** (*str*, *int*) – interface id to use
- **macaddress** (*str*) – mac address to use on interface
- **nodes** (*list*) – interface node list
- **is\_mgmt** (*bool*) – is this a management interface
- **zone\_ref** – zone to use, by name, str href or Zone
- **vlan\_id** – optional VLAN id if this should be a VLAN interface

#### Raises

**EngineCommandFailed** – failure creating interface

#### Returns

None

**add\_dhcp\_interface**(*interface\_id*, *dynamic\_index*, *zone\_ref*=None, *vlan\_id*=None, *comment*=None, *pppoe\_settings*=None, *ipv6*=False, *\*\*kw*)

Add a DHCP interface on a single engine :param int interface\_id: interface id :param int vlan\_id: vlan id :param int dynamic\_index: index number for dhcp interface (IPv4/IPv6) :param bool primary\_mgt: whether to make this primary mgt :param str zone\_ref: zone reference, can be name, href or Zone :param bool ipv6: to set IPv6 dynamic interface mode :param dict pppoe\_settings: set pppoe settings with a dict

```
pppoe_dict = {"pppoe_password": "password", "pppoe_servicename": "sn", "pppoe_username": "name"}
```

#### Parameters

**comment** (*str*) – comment

#### Raises

**EngineCommandFailed** – failure creating interface

#### Returns

None

See DHCPInterface for more information

**add\_inline\_interface**(*interface\_id*, *second\_interface\_id*, *logical\_interface\_ref*=None, *vlan\_id*=None, *second\_vlan\_id*=None, *zone\_ref*=None, *second\_zone\_ref*=None, *failure\_mode*='normal', *comment*=None, *\*\*kw*)

Add an inline interface pair. This method is only for IPS or L2FW engine types.

#### Parameters

- **interface\_id** (*str*) – interface id of first interface
- **second\_interface\_id** (*str*) – second interface pair id
- **logical\_interface\_ref** (*str*, *href*) – logical interface by href or name
- **vlan\_id** (*str*) – vlan ID for first interface in pair
- **second\_vlan\_id** (*str*) – vlan ID for second interface in pair
- **zone\_ref** (*str*, *href*) – zone reference by name or href for first interface

- **second\_zone\_ref** (*str*, *href*) – zone reference by nae or href for second interface
- **failure\_mode** (*str*) – normal or bypass
- **comment** (*str*) – optional comment

**Raises**

**EngineCommandFailed** – failure creating interface

**Returns**

None

```
add_inline_ips_interface(interface_id, second_interface_id, logical_interface_ref=None,
                        vlan_id=None, failure_mode='normal', zone_ref=None,
                        second_zone_ref=None, comment=None)
```

New in version 0.5.6: Using an inline interface on a layer 3 engine requires SMC and engine version >= 6.3.

An inline IPS interface is a new interface type for Layer 3 NGFW engines version >=6.3. Traffic passing an Inline IPS interface will have a access rule default action of Allow. Inline IPS interfaces are bypass capable. When using bypass interfaces and NGFW is powered off, in an offline state or overloaded, traffic is allowed through without inspection regardless of the access rules.

If the interface does not exist and a VLAN id is specified, the logical interface and zones will be applied to the top level physical interface. If adding VLANs to an existing inline ips pair, the logical and zones will be applied to the VLAN.

**Parameters**

- **interface\_id** (*str*) – first interface in the interface pair
- **second\_interface\_id** (*str*) – second interface in the interface pair
- **logical\_interface\_ref** (*str*) – logical interface name, href or LogicalInterface. If None, 'default\_eth' logical interface will be used.
- **vlan\_id** (*str*) – optional VLAN id for first interface pair
- **failure\_mode** (*str*) – 'normal' or 'bypass' (default: normal). Bypass mode requires fail open interfaces.
- **zone\_ref** – zone for first interface in pair, can be name, str href or Zone
- **second\_zone\_ref** – zone for second interface in pair, can be name, str href or Zone
- **comment** (*str*) – comment for this interface

**Raises**

**EngineCommandFailed** – failure creating interface

**Returns**

None

---

**Note:** Only a single VLAN is supported on this inline pair type

---

```
add_inline_l2fw_interface(interface_id, second_interface_id, logical_interface_ref=None,
                        vlan_id=None, zone_ref=None, second_zone_ref=None, comment=None)
```

New in version 0.5.6: Requires NGFW engine >=6.3 and layer 3 engine or cluster

An inline L2 engine interface is a new interface type for Layer 3 NGFW engines version >=6.3. Traffic passing an Inline Layer 2 Firewall interface will have a default action in access rules of Discard. Layer 2

Firewall interfaces are not bypass capable, so when NGFW is powered off, in an offline state or overloaded, traffic is blocked on this interface.

If the interface does not exist and a VLAN id is specified, the logical interface and zones will be applied to the top level physical interface. If adding VLANs to an existing inline ips pair, the logical and zones will be applied to the VLAN.

#### Parameters

- **interface\_id** (*str*) – interface id; ‘1-2’, ‘3-4’, etc
- **logical\_interface\_ref** (*str*) – logical interface name, href or LogicalInterface. If None, ‘default\_eth’ logical interface will be used.
- **vlan\_id** (*str*) – optional VLAN id for first interface pair
- **vlan\_id2** (*str*) – optional VLAN id for second interface pair
- **zone\_ref\_intf1** – zone for first interface in pair, can be name, str href or Zone
- **zone\_ref\_intf2** – zone for second interface in pair, can be name, str href or Zone

#### Raises

*EngineCommandFailed* – failure creating interface

#### Returns

None

---

**Note:** Only a single VLAN is supported on this inline pair type

---

**add\_layer3\_cluster\_interface**(*interface\_id*, *cluster\_virtual*=None, *network\_value*=None, *macaddress*=None, *nodes*=None, *cvi\_mode*='packetdispatch', *zone\_ref*=None, *comment*=None, *lldp\_mode*=None, \*\*kw)

Add cluster virtual interface. A “CVI” interface is used as a VIP address for clustered engines. Providing ‘nodes’ will create the node specific interfaces. You can also add a cluster address with only a CVI, or only NDI’s.

Add CVI only:

```
engine.physical_interface.add_cluster_virtual_interface(  
    interface_id=30,  
    cluster_virtual='30.30.30.1',  
    network_value='30.30.30.0/24',  
    macaddress='02:02:02:02:02:06')
```

Add NDI’s only:

```
engine.physical_interface.add_cluster_virtual_interface(  
    interface_id=30,  
    nodes=nodes)
```

Add CVI and NDI’s:

```
engine.physical_interface.add_cluster_virtual_interface(  
    cluster_virtual='5.5.5.1',  
    network_value='5.5.5.0/24',  
    macaddress='02:03:03:03:03:03',
```

(continues on next page)

(continued from previous page)

```
nodes=[{'address':'5.5.5.2', 'network_value':'5.5.5.0/24', 'nodeid':1},
        {'address':'5.5.5.3', 'network_value':'5.5.5.0/24', 'nodeid':2}]])
```

Changed in version 0.6.1: Renamed from `add_cluster_virtual_interface`

#### Parameters

- **interface\_id** (*str*, *int*) – physical interface identifier
- **cluster\_virtual** (*str*) – CVI address (VIP) for this interface
- **network\_value** (*str*) – network value for VIP; format: 10.10.10.0/24
- **macaddress** (*str*) – mandatory mac address if cluster\_virtual and cluster\_mask provided
- **nodes** (*list*) – list of dictionary items identifying cluster nodes
- **cvi\_mode** (*str*) – packetdispatch is recommended setting
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone
- **lldp\_mode** (*str*) – disabled, receive\_only, send\_and\_receive, send\_only
- **kw** – key word arguments are valid NodeInterface sub-interface settings passed in during create time. For example, 'backup\_mgt=True' to enable this interface as the management backup.

#### Raises

*EngineCommandFailed* – failure creating interface

#### Returns

None

**add\_layer3\_interface**(*interface\_id*, *address*, *network\_value*, *zone\_ref*=None, *comment*=None, *lldp\_mode*=None, *\*\*kw*)

Add a layer 3 interface on a non-clustered engine. For Layer 2 engine and IPS engines, this interface type represents a layer 3 routed (node dedicated) interface. For clusters, use the cluster related methods such as *add\_layer3\_cluster\_interface()*

#### Parameters

- **interface\_id** (*str*, *int*) – interface identifier
- **address** (*str*) – ip address
- **network\_value** (*str*) – network/cidr (12.12.12.0/24)
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone
- **lldp\_mode** (*str*) – disabled, receive\_only, send\_and\_receive, send\_only
- **comment** (*str*) – optional comment
- **kw** – keyword arguments are passed to the sub-interface during create time. If it is a single engine, the sub-interface type is *smc.core.sub\_interfaces.SingleNodeInterface*. For all other engines, the type is *smc.core.sub\_interfaces.NodeInterface* For example, pass 'backup\_mgt=True' to enable this interface as the management backup.

#### Raises

*EngineCommandFailed* – failure creating interface

#### Returns

None

**Note:** If an existing ip address exists on the interface and zone\_ref is provided, this value will overwrite any previous zone definition.

---

**add\_layer3\_shared\_virtual\_interface**(*interface\_id*, *mac\_address\_prefix*, *vlan\_id*=None, *virtual\_resource\_settings*=None, *zone\_ref*=None, *comment*=None, *\*\*kw*)

New in version 0.7.0: Requires SMC >= 6.6.0

Add a single layer 3 physical interface on a Master NGFW Engine that can be shared by up to 250 Virtual Firewalls. In addition, VLAN interfaces under the physical interface can be shared.

A shared interface can have individual VLANs with single virtual resources but this interface type can provide communication without requiring an external switch (access rules and routing are still required to allow access).

If a VLAN id is not provided, then this interface will be shared by the specified virtual resource settings.

Virtual resource settings should be a list in the following format:

```
[{'qos_limit': -1,
  'virtual_mapping': '0',
  'virtual_resource_name': 've-1'}]
```

The *qos\_limit* defines the throughput limit for the given virtual resource (-1 is no limit) in Mbps. The *virtual\_mapping* defines the interface ID that will be seen within the VirtualFirewall. The resource name maps to the VirtualResource and should already exist.

**See also:**

[\*smc.core.engine.VirtualResource\*](#)

#### Parameters

- **interface\_id** – the interface ID for the shared interface
- **mac\_address\_prefix** (*str*) – a unique unicast MAC address prefix (the first five octets of a MAC address) that identifies the interface and groups the Virtual Firewalls that use this interface
- **vlan\_id** (*str*) – optional VLAN id if specifying shared VLANs
- **virtual\_resource\_settings** (*list*) – list of virtual resources by name to add to shared interface
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone
- **comment** (*str*) – optional comment

**add\_layer3\_vlan\_cluster\_interface**(*interface\_id*, *vlan\_id*, *nodes*=None, *cluster\_virtual*=None, *network\_value*=None, *macaddress*=None, *cvi\_mode*='packetdispatch', *zone\_ref*=None, *comment*=None, *\*\*kw*)

Add IP addresses to VLANs on a firewall cluster. The minimum params required are *interface\_id* and *vlan\_id*. To create a VLAN interface with a CVI, specify *cluster\_virtual*, *cluster\_mask* and *macaddress*.

To create a VLAN with only NDI, specify *nodes* parameter.

Nodes data structure is expected to be in this format:

```
nodes=[{'address':'5.5.5.2', 'network_value':'5.5.5.0/24', 'nodeid':1},
        {'address':'5.5.5.3', 'network_value':'5.5.5.0/24', 'nodeid':2}]
```

### Parameters

- **interface\_id** (*str*, *int*) – interface id to assign VLAN.
- **vlan\_id** (*str*, *int*) – vlan identifier
- **nodes** (*list*) – optional addresses for node interfaces (NDI's). For a cluster, each node will require an address specified using the nodes format.
- **cluster\_virtual** (*str*) – cluster virtual ip address (optional). If specified, cluster\_mask parameter is required
- **network\_value** (*str*) – Specifies the network address, i.e. if cluster virtual is 1.1.1.1, cluster mask could be 1.1.1.0/24.
- **macaddress** (*str*) – (optional) if used will provide the mapping from node interfaces to participate in load balancing.
- **cvi\_mode** (*str*) – cvi mode for cluster interface (default: packetdispatch)
- **zone\_ref** – zone to assign, can be name, str href or Zone
- **kw** (*dict*) – keyword arguments are passed to top level of VLAN interface, not the base level physical interface. This is useful if you want to pass in a configuration that enables the DHCP server on a VLAN for example.

### Raises

*EngineCommandFailed* – failure creating interface

### Returns

None

---

**Note:** If the interface\_id specified already exists, it is still possible to add additional VLANs and interface addresses.

---

**class** smc.core.collection.SwitchInterfaceCollection(engine)

Bases: *InterfaceCollection*

SwitchInterfaceCollection provides an interface to retrieving existing interfaces and helper methods to shortcut the creation of a switch. Note that switch interfaces are only supported on specific engine types and require that the top level switch is created and port groups are created (although you can use one single port group for the entire switch configuration).

Get specific switch interfaces assigned on the given engine:

```
for interface in engine.switch_physical_interface:
    print(interface, interface.port_groups_interface)
```

You can also retrieve a switch directly by referencing it using the switch interface id. Switch interfaces will always have a name starting with 'SWP\_'. For example, SWP\_0 specifies physical switch port 0:

```
engine.switch_physical_interface.get('SWP_0')
```

You can also get port\_group\_interfaces directly:

```
engine.switch_physical_interface.get('SWP_0.1')
```

Or iterate through the `port_group_interface` collection:

```
interface = engine.switch_physical_interface.get('SWP_0')
for port_group in interface.port_group_interface:
    ...
```

**add\_port\_group\_interface**(*interface\_id*, *port\_group\_id*, *interface\_ports*, *interfaces=None*, *zone\_ref=None*)

Add a port group to an existing switch physical interface. If the switch port should have an address assigned, use the following format:

```
engine.switch_physical_interface.add_port_group_interface('SWP_1', 1, [1],
    interfaces=[{'nodes': [{'address': '12.12.12.12',
                           'network_value': '12.12.12.0/24',
                           'nodeid': 1}]}])
```

To create a generic switch port group without IP addresses assigned with port group ID 1 and using physical port numbers 2,3,4,5:

```
engine.switch_physical_interface.add_port_group_interface('SWP_1', 1, [2,3,4,5])
```

---

**Note:** If the port group ID exists, this method will modify the existing port group with the specified settings

---

#### Parameters

- **interface\_id** (*str*) – The top level switch, naming convention should be SWP\_0, SWP\_1, etc. ( Since API 6.8: SWI\_0, SWI\_1..)
- **port\_group\_id** (*int*) – Port group number encapsulating switch port/s
- **interface\_ports** (*list*) – list of interface ports to add to this port group. If the port group.
- **interfaces** (*list*) – list of interface node definitions if the switch port should have IP address/es assigned
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone, will be created if it doesn't exist

#### Raises

**InterfaceNotFound** – invalid switch interface\_id specified

#### Returns

None

**add\_switch\_interface**(*interface\_id*, *appliance\_switch\_module='110'*, *comment=None*, *\*\*kwargs*)

In case of Switch Physical/Port Group interfaces, the interface ID must be prefixed by 'SWP\_'. For example, for switch ID 1 and Port Group ID 1.2 you must enter 'SWP\_1' for the switch and SWP\_1.2 for the Port Group. Since API 6.8 prefix is 'SWI\_'

#### Parameters

- **interface\_id** (*str*) – Name of the interface, must be prefixed with 'SWP\_'

- **appliance\_switch\_module** (*str*) – appliance switch module which specifies the hardware module (default: '110')
- **comment** (*str*) – optional comment
- **kwargs** (*dict*) – optional kwargs conforming to the port group dict format if port groups need to be created

**Raises**

*EngineCommandFailed* – failure during creation

**Returns**

None

**class** smc.core.collection.TunnelInterfaceCollection(*engine*)

Bases: *InterfaceCollection*

TunnelInterface Collection provides an interface to retrieving existing interfaces and helper methods to shortcut the creation of an interface.

**add\_cluster\_virtual\_interface**(*interface\_id*, *cluster\_virtual=None*, *network\_value=None*, *nodes=None*, *zone\_ref=None*, *comment=None*, *\*\*kw*)

Add a tunnel interface on a clustered engine. For tunnel interfaces on a cluster, you can specify a CVI only, NDI interfaces, or both. This interface type is only supported on layer 3 firewall engines.

Add a tunnel CVI **and** NDI:

```
engine.tunnel_interface.add_cluster_virtual_interface(
    interface_id=3000,
    cluster_virtual='4.4.4.1',
    network_value='4.4.4.0/24',
    nodes=nodes)
```

Add tunnel NDI's **only**:

```
engine.tunnel_interface.add_cluster_virtual_interface(
    interface_id=3000,
    nodes=nodes)
```

Add tunnel CVI **only**:

```
engine.tunnel_interface.add_cluster_virtual_interface(
    interface_id=3000,
    cluster_virtual='31.31.31.31',
    network_value='31.31.31.0/24',
    zone_ref='myzone')
```

**Parameters**

- **interface\_id** (*str*, *int*) – tunnel identifier (akin to interface\_id)
- **cluster\_virtual** (*str*) – CVI ipaddress (optional)
- **network\_value** (*str*) – CVI network; required if **cluster\_virtual** set
- **nodes** (*list*) – nodes for clustered engine with address,network\_value,nodeid
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone
- **comment** (*str*) – optional comment

**Raises**

*EngineCommandFailed* – failure during creation

**Returns**

None

```
add_layer3_interface(interface_id, address=None, network_value=None, zone_ref=None,  
                     comment=None, **kw)
```

Creates a tunnel interface with sub-type single\_node\_interface. This is to be used for single layer 3 firewall instances.

---

**Note:** If no address or network\_value is provided, an unconfigured tunnel interface will be created

---

**Parameters**

- **interface\_id** (*str*, *int*) – the tunnel id for the interface, used as nicid also
- **address** (*str*) – ip address of interface
- **network\_value** (*str*) – network cidr for interface; format: 1.1.1.0/24
- **zone\_ref** (*str*) – zone reference for interface can be name, href or Zone
- **comment** (*str*) – optional comment

**Raises**

*EngineCommandFailed* – failure during creation

**Returns**

None

```
class smc.core.collection.VPNBrokerInterfaceCollection(engine)
```

Bases: *InterfaceCollection*

VPNBrokerInterfaceCollection Collection provides an interface to retrieving existing interfaces and helper methods to shortcut the creation of an interface.

```
add_cluster_virtual_interface(interface_id, cluster_virtual=None, vpn_broker_domain_ref=None,  
                              mac_address_postfix=None, shared_secret=None,  
                              network_value=None, nodes=None, zone_ref=None, comment=None,  
                              retrieve_routes=None, adjust_antispoofing=None, **kw)
```

Add a VPN Broker interface on a clustered engine. For VPN Broker interfaces on a cluster, you can specify a CVI only, NDI interfaces, or both. This interface type is only supported on layer 3 firewall engines.

Add a VPN Broker CVI **and** NDI:

```
engine.vpn_broker_interface.add_cluster_virtual_interface(  
    interface_id=3000,  
    cluster_virtual='4.4.4.1',  
    network_value='4.4.4.0/24',  
    nodes=nodes)
```

Add VPN Broker NDI's **only**:

```
engine.vpn_broker_interface.add_cluster_virtual_interface(  
    interface_id=3000,
```

(continues on next page)

(continued from previous page)

```

nodes=nodes)

Add VPN Broker CVI only:

engine.vpn_broker_interface.add_cluster_virtual_interface(
    interface_id=3000,
    cluster_virtual='31.31.31.31',
    network_value='31.31.31.0/24',
    zone_ref='myzone')

```

**add\_layer3\_interface**(*interface\_id*, *address=None*, *network\_value=None*, *zone\_ref=None*, *comment=None*, *vpn\_broker\_domain\_ref=None*, *mac\_address\_postfix=None*, *retrieve\_routes=None*, *adjust\_antispoofing=None*, *shared\_secret=None*, *\*\*kw*)

Creates a vpn broker interface with sub-type single\_node\_interface. This is to be used for single layer 3 firewall instances.

**class** smc.core.collection.VirtualPhysicalInterfaceCollection(*engine*)

Bases: [InterfaceCollection](#)

PhysicalInterface Collection provides an interface to retrieving existing interfaces and helper methods to shortcut the creation of an interface.

**add\_layer3\_interface**(*interface\_id*, *address*, *network\_value*, *zone\_ref=None*, *comment=None*, *\*\*kw*)

Add a layer 3 interface on a virtual engine.

#### Parameters

- **interface\_id** (*str*, *int*) – interface identifier
- **address** (*str*) – ip address
- **network\_value** (*str*) – network/cidr (12.12.12.0/24)
- **zone\_ref** (*str*) – zone reference, can be name, href or Zone
- **kw** – keyword arguments are passed are any value attribute values of type [smc.core.sub\\_interfaces.NodeInterface](#)

#### Raises

[EngineCommandFailed](#) – failure creating interface

#### Returns

None

---

**Note:** If an existing ip address exists on the interface and zone\_ref is provided, this value will overwrite any previous zone definition.

---

**add\_tunnel\_interface**(*interface\_id*, *address*, *network\_value*, *zone\_ref=None*, *comment=None*)

Creates a tunnel interface for a virtual engine.

#### Parameters

- **interface\_id** (*str*, *int*) – the tunnel id for the interface, used as nicid also
- **address** (*str*) – ip address of interface
- **network\_value** (*str*) – network cidr for interface; format: 1.1.1.0/24
- **zone\_ref** (*str*) – zone reference for interface can be name, href or Zone

**Raises**

[\*EngineCommandFailed\*](#) – failure during creation

**Returns**

None

Interface module encapsulates interface types for security engines. All interface have a ‘top level’ such as Physical or Tunnel Interface. These top level interfaces have certain common settings that can be modified such as assigning a zone.

IP addresses, netmask, management settings, VLANs, etc are part of an interfaces ‘sub’ interface. Sub interfaces can be retrieved from an engine reference and call to [\*sub\\_interfaces\(\)\*](#)

The interface hierarchy resembles:

```
Interface
|
Physical/Tunnel Interface
|
| - VlanInterface (is a PhysicalInterface)
| /
Sub Interfaces (SingleNodeInterface, NodeInterface, InlineInterface, etc)
|
Attributes (address, network_value, vlan_id, etc)
```

Sub interfaces are documented in [\*smc.core.sub\\_interfaces\*](#).

VLANs are properties of specific interfaces and can also be retrieved by first getting the top level interface, and calling [\*vlan\\_interface\(\)\*](#)

to view or modify specific aspects of a VLAN, such as addresses, etc.

```
class smc.core.interfaces.Interface(**meta)
```

Interface settings common to all interface types.

```
add_ip_address(cvi=None, sni=None, nodes=None)
```

Add an ip address(ipv4/ipv6) to tunnel interface :param SingleNodeInterface sni: The single node interface.  
:param ClusterVirtualInterface cvi: The cluster virtual interface instance. :param list(NodeInterface) nodes:  
List of Node Interface instance.

**property addresses**

Return 3-tuple with (address, network, nicid)

**Returns**

address related information of interface as 3-tuple list

**Return type**

[\*list\*](#)

**property all\_interfaces**

Access to all assigned sub-interfaces on this interface. A sub interface is the node level where IP addresses are assigned, or a inline interface is defined, VLANs, etc. Example usage:

```
>>> engine = Engine('dingo')
>>> itf = engine.interface.get(0)
>>> assigned = itf.all_interfaces
>>> list(assigned)
[SingleNodeInterface(address=1.1.1.1)]
```

(continues on next page)

(continued from previous page)

```

>>> assigned.get(address='1.1.1.1')
SingleNodeInterface(address=1.1.1.1)
>>> itf = engine.interface.get(52)
>>> assigned = itf.all_interfaces
>>> list(assigned)
[Layer3PhysicalInterfaceVlan(name=VLAN 52.52),
 Layer3PhysicalInterfaceVlan(name=VLAN 52.53)]
>>> vlan = assigned.get(vlan_id='52')
>>> vlan.addresses
[(u'52.52.52.52', u'52.52.52.0/24', u'52.52')]

```

**Return type***BaseIterable*(AllInterfaces)**change\_interface\_id(interface\_id)**

Change the interface ID for this interface. This can be used on any interface type. If the interface is an Inline interface, you must provide the `interface_id` in format '1-2' to define both interfaces in the pair. The change is committed after calling this method.

```

itf = engine.interface.get(0)
itf.change_interface_id(10)

```

Or inline interface pair 10-11:

```

itf = engine.interface.get(10)
itf.change_interface_id('20-21')

```

**Parameters**

**interface\_id** (*str, int*) – new interface ID. Format can be single value for non-inline interfaces or '1-2' format for inline.

**Raises**

*UpdateElementFailed* – changing the interface failed with reason

**Returns**

None

**property comment**

Optional interface comment

**Returns**

str or None

**property contact\_addresses**

Configure an interface contact address for this interface. Note that an interface may have multiple IP addresses assigned so you may need to iterate through contact addresses. Example usage:

```

>>> itf = engine.interface.get(0)
>>> itf.contact_addresses
[ContactAddressNode(interface_id=0, interface_ip=1.1.1.10),
 ContactAddressNode(interface_id=0, interface_ip=1.1.1.25)]
>>> for ca in itf.contact_addresses:
...     print("IP: %s, addresses: %s" % (ca.interface_ip, list(ca)))

```

(continues on next page)

(continued from previous page)

```
...
IP: 1.1.1.10, addresses: []
IP: 1.1.1.25, addresses: [InterfaceContactAddress(address=172.18.1.20,
                                location=Default)]

>>> for ca in itf.contact_addresses:
...     if ca.interface_ip == '1.1.1.10':
...         ca.add_contact_address('10.5.5.5', location='remote')
```

**Returns**

list of interface contact addresses

**Return type***ContactAddressNode***See also:***smc.core.contact\_address***delete()**

Override delete in parent class, this will also delete the routing configuration referencing this interface.

```
engine = Engine('vm')
interface = engine.interface.get(2)
interface.delete()
```

**delete\_invalid\_route()**

Delete any invalid routes for this interface. An invalid route is a left over when an interface is changed to a different network.

**Returns**

None

**get\_boolean(name)**

Get the boolean value for attribute specified from the sub interface/s.

**property has\_interfaces**

Does the interface have interface have any sub interface types assigned. For example, a physical interface with no IP addresses would return False.

**Returns**

Does this interface have actual types assigned

**Return type***bool***property has\_vlan**

Does the interface have VLANs

**Returns**

Whether VLANs are configured

**Return type***bool*

**property interface\_id**

The Interface ID automatically maps to a physical network port of the same number during the initial configuration of the engine, but the mapping can be changed as necessary. Call [change\\_interface\\_id\(\)](#) to change inline, VLAN, cluster and single interface ID's.

---

**Note:** It is not possible to change an interface ID from a VlanInterface. You must call on the parent PhysicalInterface.

---

**Parameters**

**value** (*str*) – interface\_id

**Return type**

*str*

**property interfaces**

Access to assigned *sub-interfaces* on this interface. A sub interface is the node level where IP addresses are assigned, or a layer 2 interface is defined.

```
>>> itf = engine.interface.get(20)
>>> assigned = itf.interfaces
>>> list(assigned)
[SingleNodeInterface(address=20.20.20.20), SingleNodeInterface(address=21.21.21.
↳21)]
>>> assigned.get(address='20.20.20.20')
SingleNodeInterface(address=20.20.20.20)
```

**Return type**

*BaseIterable(SubInterfaceCollection)*

**property log\_moderation**

This is the definition of Log Compression for the engine or for an interface. You can also configure Log Compression to save resources on the engine. By default, each generated Antispoofing and Discard log entry is logged separately and displayed as a separate entry in the Logs view. Log Compression allows you to define the maximum number of separately logged entries. When the defined limit is reached, a single Antispoofing log entry or Discard log entry is logged. The single entry contains information on the total number of the generated Antispoofing log entries or Discard log entries. After this, logging returns to normal and all the generated entries are once more logged and displayed separately. Example of using log moderation settings:

```
>>> engine = Engine("testme")
>>> interface = engine.interface.get(1)
>>> log_moderation_obj = interface.log_moderation
>>> log_moderation_obj.get(1)["rate"]
100
>>> log_moderation_obj.get(1)["burst"]
1000
>>> log_moderation_obj.add(rate=200, burst=1100, log_event=2)
>>> interface.update()
>>> engine = Engine("testme")
>>> interface = engine.interface.get(1)
>>> log_moderation_obj=interface.log_moderation
```

(continues on next page)

(continued from previous page)

```
>>> log_moderation_obj.get(2)["rate"]
200
>>> log_moderation_obj.get(2)["burst"]
1100
```

:rtype LogModeration

**property name**

Read only name tag

**property port\_group\_interface**

The associated port group interfaces for this switch physical interface.

**Return type**

PortGroupInterfaceCollection(PortGroupInterface)

**reset\_interface()**

Reset the interface by removing all assigned addresses and VLANs. This will not delete the interface itself, only the sub interfaces that may have addresses assigned. This will not affect inline or capture interfaces. Note that if this interface is used as a primary control, auth request or outgoing interface, the update will fail. You should move that functionality to another interface before calling this. See also: `smc.core.engine.interface_options`.

**Raises**

*UpdateElementFailed* – failed to update the interfaces. This is usually caused when the interface is assigned as a control, outgoing, or auth\_request interface.

**Returns**

None

**sub\_interfaces()**

Flatten out all top level interfaces and only return sub interfaces. It is recommended to use *all\_interfaces()*, *interfaces()* or *vlan\_interfaces()* which return collections with helper methods to get sub interfaces based on index or attribute value pairs.

**Return type**

*list(SubInterface)*

**update(\*args, \*\*kw)**

Update/save this interface information back to SMC. When interface changes are made, especially to sub interfaces, call *update* on the top level interface.

Example of changing the IP address of an interface:

```
>>> engine = Engine('sg_vm')
>>> interface = engine.physical_interface.get(1)
>>> interface.zone_ref = zone_helper('mynewzone')
>>> interface.update()
```

**Raises**

*UpdateElementFailed* – failure to save changes

**Returns**

Interface

**update\_interface**(*other\_interface*, *ignore\_mgmt=True*)

Update an existing interface by comparing values between two interfaces. If a VLAN interface is defined in the other interface and it doesn't exist on the existing interface, it will be created.

**Parameters**

- **Interface** (*other\_interface*) – an instance of an interface where values in this interface will be used to as the template to determine changes. This only has to provide attributes that need to change (or not).
- **ignore\_mgmt** (*bool*) – ignore resetting management fields. These are generally better set after creation using *engine.interface\_options*

**Raises**

*UpdateElementFailed* – Failed to update the element

**Returns**

(Interface, modified, created)

**Return type**

tuple

---

**Note:** Interfaces with multiple IP addresses are ignored

---

**property** `vlan_interface`

Access VLAN interfaces for this interface, if any. Example usage:

```
>>> itf = engine.interface.get(52)
>>> assigned = itf.vlan_interface
>>> list(assigned)
[Layer3PhysicalInterfaceVlan(name=VLAN 52.52),
 Layer3PhysicalInterfaceVlan(name=VLAN 52.53)]
>>> vlan = assigned.get(vlan_id='52')
>>> vlan.addresses
[(u'52.52.52.52', u'52.52.52.0/24', u'52.52')]
>>> assigned.get(address='12.12.12.13')
SingleNodeInterface(address=12.12.12.13, vlan_id=1)
>>> assigned.get(vlan_id='1')
SingleNodeInterface(address=12.12.12.12, vlan_id=1)
>>> assigned.get(vlan_id='2')
SingleNodeInterface(address=36.35.35.37, vlan_id=2)
```

**Return type**

*BaseIterable(VlanInterface)*

**property** `zone`

Return the Zone for this interface, otherwise None

**Returns**

Zone or None

**property** `zone_ref`

Zone for this physical interface.

**Parameters**

**value** (*str*) – href of zone, set to None to remove existing zone

**Return type**`str`

### 14.6.5.2 InterfaceOptions

**class** `smc.core.interfaces.InterfaceOptions(engine)`

Interface Options allow you to define settings related to the roles of the firewall interfaces:

- Which IP addresses are used as the primary and backup Control IP address
- Which interfaces are used as the primary and backup heartbeat interface
- The default IP address for outgoing traffic

You can optionally change which interface is used for each of these purposes, and define a backup Control IP address and backup Heartbeat Interface. If calling the *set* methods, using a value of `None` will unset the option.

---

**Note:** Setting an interface option will commit the change immediately.

---

**property** `auth_request`

Return the interface for authentication requests. Can be either a `PhysicalInterface` or `LoopbackInterface`

**Returns**`interface id`**Return type**`str`**property** `backup_heartbeat`

Obtain the interface specified as the backup heartbeat interface. This may return `None` if a backup has not been specified or this is not a cluster.

**Returns**`interface id`**Return type**`str`**property** `backup_mgt`

Obtain the interface specified as the backup management interface. This can return `None` if no backup has been defined

**Returns**`interface id`**Return type**`str`**property** `outgoing`

Obtain the interface specified as the “Default IP address for outgoing traffic”. This will always return a value.

**Returns**`interface id`**Return type**`str`

**property primary\_heartbeat**

Obtain the interface specified as the primary heartbeat interface. This will return None if this is not a clustered engine.

**Returns**

interface id

**Return type**

`str`

**property primary\_mgt**

Obtain the interface specified as the primary management interface. This will always return a value as you must have at least one physical interface specified for management.

**Returns**

interface id

**Return type**

`str`

**set\_auth\_request**(*interface\_id*, *address=None*)

Set the authentication request field for the specified engine.

**set\_backup\_heartbeat**(*interface\_id*)

Set this interface as the backup heartbeat interface. Clusters and Master NGFW Engines only.

**Parameters**

**interface\_id** (`str`, `int`) – interface as backup

**Raises**

- **InterfaceNotFound** – specified interface is not found
- **UpdateElementFailed** – failure to update interface

**Returns**

None

**set\_backup\_mgt**(*interface\_id*)

Set this interface as a backup management interface.

Backup management interfaces cannot be placed on an interface with only a CVI (requires node interface/s). To ‘unset’ the specified interface address, set interface id to None

```
engine.interface_options.set_backup_mgt(2)
```

Set backup on interface 1, VLAN 201:

```
engine.interface_options.set_backup_mgt('1.201')
```

Remove management backup from engine:

```
engine.interface_options.set_backup_mgt(None)
```

**Parameters**

**interface\_id** (`str`, `int`) – interface identifier to make the backup management server.

**Raises**

- **InterfaceNotFound** – specified interface is not found

- *UpdateElementFailed* – failure to make modification

**Returns**

None

**set\_outgoing(interface\_id)**

Specifies the IP address that the engine uses to initiate connections (such as for system communications and ping) through an interface that has no Node Dedicated IP Address. In clusters, you must select an interface that has an IP address defined for all nodes. Setting primary\_mgt also sets the default outgoing address to the same interface.

**Parameters**

**interface\_id** (*str*, *int*) – interface to set outgoing

**Raises**

- *InterfaceNotFound* – specified interface is not found
- *UpdateElementFailed* – failure to make modification

**Returns**

None

**set\_primary\_heartbeat(interface\_id)**

Set this interface as the primary heartbeat for this engine. This will ‘unset’ the current primary heartbeat and move to specified interface\_id. Clusters and Master NGFW Engines only.

**Parameters**

**interface\_id** (*str*, *int*) – interface specified for primary mgmt

**Raises**

- *InterfaceNotFound* – specified interface is not found
- *UpdateElementFailed* – failed modifying interfaces

**Returns**

None

**set\_primary\_mgt(interface\_id, auth\_request=None, address=None)**

Specifies the Primary Control IP address for Management Server contact. For single engine and cluster engines, this will enable ‘Outgoing’, ‘Auth Request’ and the ‘Primary Control’ interface. For clusters, the primary heartbeat will NOT follow this change and should be set separately using *set\_primary\_heartbeat()*. For virtual engines, only auth\_request and outgoing will be set. For master engines, only primary control and outgoing will be set.

Primary management can be set on an interface with single IP’s, multiple IP’s or VLANs.

```
engine.interface_options.set_primary_mgt(1)
```

Set primary management on a VLAN interface:

```
engine.interface_options.set_primary_mgt('1.100')
```

Set primary management and different interface for auth\_request:

```
engine.interface_options.set_primary_mgt(  
    interface_id='1.100', auth_request=0)
```

Set on specific IP address of interface VLAN with multiple addresses:

```
engine.interface_options.set_primary_mgt(
    interface_id='3.100', address='50.50.50.1')
```

#### Parameters

- **interface\_id** (*str*, *int*) – interface id to make management
- **address** (*str*) – if the interface for management has more than one ip address, this specifies which IP to bind to.
- **auth\_request** (*str*, *int*) – if setting primary mgt on a cluster interface with no CVI, you must pick another interface to set the auth\_request field to (default: None)

#### Raises

- **InterfaceNotFound** – specified interface is not found
- **UpdateElementFailed** – updating management fails

#### Returns

None

---

**Note:** Setting primary management on a cluster interface with no CVI requires you to set the interface for auth\_request.

---

### 14.6.5.3 QoS

**class** smc.core.interfaces.QoS(*interface*)

QoS can be placed on physical interfaces, physical VLAN interfaces and tunnel interfaces. It is possible to have multiple QoS policies defined if using VLANs on a physical interface as QoS can be attached directly at the interface level or VLAN level. You obtain the QoS reference after retrieving the interface:

```
itf = engine.interface.get(0)
itf.qos.full_qos(1000000, QoSPolicy('testqos'))
itf.update()
```

Disable QoS:

```
itf = engine.interface.get(0)
itf.qos.disable()
itf.update()
```

On a tunnel interface:

```
itf = engine.interface.get(1000)
itf.qos.full_qos(1000000, QoSPolicy('testqos'))
itf.update()
```

Or a VLAN:

```
itf = engine.interface.get('1.100')
itf.qos.full_qos(1000000, QoSPolicy('testqos'))
itf.update()
```

---

**Note:** You must call *update* on the interface to commit the change

---

**disable()**

Disable QoS on this interface

**dscp\_marking\_and\_throttling(*qos\_policy*)**

Enable DSCP marking and throttling on the interface. This requires that you provide a QoS policy to which identifies DSCP tags and how to prioritize that traffic.

**Parameters**

**qos\_policy** (*QoSPolicy*) – the qos policy to apply to the interface

**full\_qos(*qos\_limit*, *qos\_policy*)**

Enable full QoS on the interface. Full QoS requires that you set a bandwidth limit (in Mbps) for the interface. You must also provide a QoS policy to which identifies the parameters for prioritizing traffic.

**Parameters**

- **qos\_limit** (*int*) – max bandwidth in Mbps
- **qos\_policy** (*QoSPolicy*) – the qos policy to apply to the interface

**property qos\_limit**

QoS Limit for this interface. The limit represents the number in bps. For example, 100000 represents 100Mbps.

**Return type**

*int*

**property qos\_mode**

QoS mode in string format

**Return type**

*str*

**property qos\_policy**

QoS Policy for this interface/vlan. A QoS policy will only be present if DSCP throttling or Full QoS is specified.

**Return type**

*QoSPolicy*

**statistics\_only()**

Set interface to collect QoS statistics only. No enforcement is being done but visibility will be provided in dashboards against applications tagged by QoS.

#### 14.6.5.4 LoopbackInterface

**class** `smc.core.sub_interfaces.LoopbackInterface(data, engine=None)`

Bases: *NodeInterface*

Loopback interface for a physical or virtual single firewall. To create a loopback interface, call from the engine node:

```
engine.loopback_interface.add_single(...)
```

**add\_node\_loopback**(*addresses*, *\*\*kwargs*)

Add a node loopback interface to this engine.

**Parameters**

**addresses** (*dict*) – {nodeid: {‘address’: ‘127.0.0.1’, ‘ospf\_area’: ospfArea}, ... }

**Raises**

*UpdateElementFailed* – failure to create loopback address

**Returns**

None

**add\_single**(*address*, *rank=1*, *nodeid=1*, *ospf\_area=None*, *\*\*kwargs*)

Add a single loopback interface to this engine. This is used for single or virtual engines.

**Parameters**

- **address** (*str*) – ip address for loopback
- **nodeid** (*int*) – nodeid to apply. Default to 1 for single engine
- **ospf\_area** (*str*, *Element*) – ospf area href or element

**Raises**

*UpdateElementFailed* – failure to create loopback address

**Returns**

None

**delete**()

Delete a loopback interface from this engine. Changes to the engine configuration are done immediately.

A simple way to obtain an existing loopback is to iterate the loopbacks or to get by address:

```
lb = engine.loopback_interface.get('127.0.0.10')
lb.delete()
```

**Warning:** When deleting a loopback assigned to a node on a cluster all loopbacks with the same rank will also be removed.

**Raises**

*UpdateElementFailed* – failure to delete loopback interface

**Returns**

None

#### 14.6.5.5 LoopbackClusterInterface

**class** smc.core.sub\_interfaces.**LoopbackClusterInterface**(*data*, *engine=None*)

Bases: *ClusterVirtualInterface*

This represents the CVI Loopback IP address. A CVI loopback IP address is used for loopback traffic that is sent to the whole cluster. It is shared by all the nodes in the cluster.

**add\_cvi\_loopback**(*address*, *ospf\_area=None*, *\*\*kw*)

Add a loopback interface as a cluster virtual loopback. This enables the loopback to ‘float’ between cluster members. Changes are committed immediately.

**Parameters**

- **address** (*str*) – ip address for loopback
- **rank** (*int*) – rank of this entry
- **ospf\_area** (*str*, *Element*) – optional ospf\_area to add to loopback

**Raises**

*UpdateElementFailed* – failure to save loopback address

**Returns**

None

**delete()**

Delete a loopback cluster virtual interface from this engine. Changes to the engine configuration are done immediately.

You can find cluster virtual loopbacks by iterating at the engine level:

```
for loopbacks in engine.loopback_interface:
    ...
```

**Raises**

*UpdateElementFailed* – failure to delete loopback interface

**Returns**

None

#### 14.6.5.6 PhysicalInterface

**class** `smc.core.interfaces.PhysicalInterface`(*engine=None, meta=None, \*\*interface*)

Bases: *Interface*

Physical Interfaces on NGFW. This represents the following base configuration for the following interface types:

- Single Node Interface
- Node Interface
- Capture Interface
- Inline Interface
- Cluster Virtual Interface
- Virtual Physical Interface (used on Virtual Engines)
- DHCP Interface

This should be used to add interfaces to an engine after it has been created. First get the engine context by loading the engine then get the engine property for physical interface:

```
engine = Engine('myfw')
engine.physical_interface.add_layer3_interface(...)
engine.physical_interface.add(5) #single unconfigured physical interface
engine.physical_interface.add_inline_ips_interface('5-6', ...)
....
```

When making changes, the etag used should be the top level engine etag.

**property aggregate\_mode**

LAGG configuration mode for this interface. Values are 'ha' or 'lb' (load balancing). This can return None if LAGG is not configured.

**Returns**

aggregate mode set, if any

**Return type**

`str`, `None`

**property arp\_entry**

Return any manually configured ARP entries for this physical interface

**Returns**

arp entries as dict

**Return type**

`list`

**change\_vlan\_id(*original*, *new*)**

Change VLAN ID for a single VLAN, cluster VLAN or inline interface. When changing a single or cluster engine vlan, you can specify the original VLAN and new VLAN as either single int or str value. If modifying an inline interface VLAN when the interface pair has two different VLAN identifiers per interface, use a str value in form: '10-11' (original), and '20-21' (new).

Single VLAN id:

```
>>> engine = Engine('singlefw')
>>> itf = engine.interface.get(1)
>>> itf.vlan_interfaces()
[PhysicalVlanInterface(vlan_id=11), PhysicalVlanInterface(vlan_id=10)]
>>> itf.change_vlan_id(11, 100)
>>> itf.vlan_interfaces()
[PhysicalVlanInterface(vlan_id=100), PhysicalVlanInterface(vlan_id=10)]
```

Inline interface with unique VLAN on each interface pair:

```
>>> itf = engine.interface.get(2)
>>> itf.vlan_interfaces()
[PhysicalVlanInterface(vlan_id=2-3)]
>>> itf.change_vlan_id('2-3', '20-30')
>>> itf.vlan_interfaces()
[PhysicalVlanInterface(vlan_id=20-30)]
```

**Parameters**

- **original** (`str`, `int`) – original VLAN to change.
- **new** (`str`, `int`) – new VLAN identifier/s.

**Raises**

- **InterfaceNotFound** – VLAN not found
- **UpdateElementFailed** – failed updating the VLAN id

**Returns**

`None`

**enable\_aggregate\_mode**(*mode*, *interfaces*)

Enable Aggregate (LAGG) mode on this interface. Possible LAGG types are ‘ha’ and ‘lb’ (load balancing). For HA, only one secondary interface ID is required. For load balancing mode, up to 7 additional are supported (8 max interfaces).

**Parameters**

- **mode** (*str*) – ‘lb’ or ‘ha’
- **interfaces** (*list(str, int)*) – secondary interfaces for this LAGG

**Raises**

*UpdateElementFailed* – failed adding aggregate

**Returns**

None

**property is\_auth\_request**

Is this physical interface tagged as the interface for authentication requests

**Return type**

*bool*

**property is\_backup\_heartbeat**

Is this physical interface tagged as the backup heartbeat interface for this cluster.

**Returns**

is backup heartbeat

**Return type**

*bool*

**property is\_backup\_mgt**

Is this physical interface tagged as the backup management interface for this cluster.

**Returns**

is backup heartbeat

**Return type**

*bool*

**property is\_outgoing**

Is this the default interface IP used for outgoing for system communications.

**Returns**

is dedicated outgoing IP interface

**Return type**

*bool*

**property is\_primary\_heartbeat**

Is this physical interface tagged as the primary heartbeat interface for this cluster.

**Returns**

is backup heartbeat

**Return type**

*bool*

**property is\_primary\_mgt**

Is this physical interface tagged as the backup management interface for this cluster.

**Returns**

is backup heartbeat

**Return type**

bool

**property lldp\_mode**

Represents the LLDP Mode configuration on an interface. Send is to advertise information about device itself. Receive is to receive information about the neighbouring devices. Disable is the default selection. Supported values are: disabled, receive\_only, send\_and\_receive, send\_only :rtype: str

**property mtu**

Set MTU on interface. Enter a value between 400-65535. The same MTU is automatically applied to any VLANs created under this physical interface

**Parameters**

value (*int*) – MTU

**Return type**

int

**property multicast\_ip**

Enter a multicast address, that is, an IP address from the range 224.0.0.0-239.255.255.255. The address is used for automatically calculating a MAC address. Required only if multicastigmp cvi mode is selected as the cvi\_mode.

**Parameters**

value (*str*) – address

**Return type**

str

**property ndi\_interfaces**

Return a formatted dict list of NDI interfaces on this engine. This will ignore CVI or any inline or layer 2 interface types. This can be used to identify to indicate available IP addresses for a given interface which can be used to run services such as SNMP or DNS Relay.

**Returns**

list of dict items [{ 'address':x, 'nicid':y}]

**Return type**

list(dict)

**property qos**

The QoS settings for this physical interface

**Return type**

*QoS*

**property second\_interface\_id**

Peer interfaces used in LAGG configuration.

**Parameters**

value (*str*) – comma seperated nic id's for LAGG peers

**Return type**

str

**static\_arp\_entry**(*ipaddress*, *macaddress*, *arp\_type*='static', *netmask*=32)

Add an arp entry to this physical interface.

```
interface = engine.physical_interface.get(0)
interface.static_arp_entry(
    ipaddress='23.23.23.23',
    arp_type='static',
    macaddress='02:02:02:02:04:04')
interface.save()
```

**Parameters**

- **ipaddress** (*str*) – ip address for entry
- **macaddress** (*str*) – macaddress for ip address
- **arp\_type** (*str*) – type of entry, ‘static’ or ‘proxy’ (default: static)
- **netmask** (*str*, *int*) – netmask for entry (default: 32)

**Returns**

None

**property virtual\_engine\_vlan\_ok**

Whether to allow VLAN creation on the Virtual Engine. Only valid for master engine.

**Parameters**

**value** (*bool*) – enable/disable

**Return type**

*bool*

**property virtual\_mapping**

The virtual mapping id. Required if Virtual Resource chosen. See [smc.core.engine.VirtualResource.vfw\\_id](#)

**Parameters**

**value** (*int*) – vfw\_id

**Return type**

*int*

**property virtual\_resource\_name**

Virtual Resource name used on Master Engine to map a virtual engine. See [smc.core.engine.VirtualResource.name](#)

**Parameters**

**value** (*str*) – virtual resource name

**Return type**

*str*

### 14.6.5.7 Layer3PhysicalInterface

**class** `smc.core.interfaces.Layer3PhysicalInterface(engine=None, meta=None, **interface)`

Bases: *PhysicalInterface*

Represents a routed layer 3 interface on an any engine type.

Example interface:

```
interface = {
    'comment': u'Regular interface',
    'interface_id': u'67',
    'interfaces': [{ 'nodes': [{ 'address': u'5.5.5.2',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1},
                            { 'address': u'5.5.5.3',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1}]}],
    'zone_ref': 'foozone'}
```

Layer3 VLAN interface:

```
interface = {
    'comment': u'Interface with VLAN',
    'interface_id': u'67',
    'interfaces': [{ 'nodes': [{ 'address': u'5.5.5.2',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1},
                            { 'address': u'5.5.5.3',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1}]}],
    'vlan_id': 10}],
    'zone_ref': 'foozone'}
```

DHCP interface on a VLAN (use *dynamic* and specify *dynamic\_index*):

```
interface = {
    'comment': u'Interface with VLAN',
    'interface_id': u'67',
    'interfaces': [{ 'nodes': [{ 'dynamic': True,
                                'dynamic_index': 2,
                                'nodeid': 1}]}],
    'vlan_id': 10}],
    'zone_ref': 'foozone'}
```

When an interface is created, the first key level is applied to the “top” level physical interface. The *interfaces* list specifies the node and addressing information using the *nodes* parameter. If *vlan\_id* is specified as a key/value in the interfaces dict, the list dict keys are applied to the nested physical interface VLAN.

#### Parameters

- **interface\_id** (*str*) – id for interface
- **interface** (*str*) – specifies the type of interface to create. The interface type defaults to ‘node\_interface’ and applies to all engine types except a single engine. For single engine, specify *single\_node\_interface*

- **interfaces** (*list*) – interface attributes, *cluster\_virtual*, *network\_value*, *nodes*, etc
- **nodes** (*dict*) – nodes dict should contain keys *address*, *network\_value* and *nodeid*. Overridden sub interface settings can also be set here
- **zone\_ref** (*str*) – zone reference, name or zone
- **comment** (*str*) – comment for interface

#### 14.6.5.8 Layer2PhysicalInterface

**class** smc.core.interfaces.Layer3PhysicalInterface(*engine=None, meta=None, \*\*interface*)

Bases: *PhysicalInterface*

Represents a routed layer 3 interface on an any engine type.

Example interface:

```
interface = {
    'comment': u'Regular interface',
    'interface_id': u'67',
    'interfaces': [{ 'nodes': [{ 'address': u'5.5.5.2',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1},
                            { 'address': u'5.5.5.3',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1}]}],
    'zone_ref': 'foozone'}
```

Layer3 VLAN interface:

```
interface = {
    'comment': u'Interface with VLAN',
    'interface_id': u'67',
    'interfaces': [{ 'nodes': [{ 'address': u'5.5.5.2',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1},
                            { 'address': u'5.5.5.3',
                                'network_value': u'5.5.5.0/24',
                                'nodeid': 1}]}],
    'vlan_id': 10}],
    'zone_ref': 'foozone'}
```

DHCP interface on a VLAN (use *dynamic* and specify *dynamic\_index*):

```
interface = {
    'comment': u'Interface with VLAN',
    'interface_id': u'67',
    'interfaces': [{ 'nodes': [{ 'dynamic': True,
                                'dynamic_index': 2,
                                'nodeid': 1}]}],
    'vlan_id': 10}],
    'zone_ref': 'foozone'}
```

When an interface is created, the first key level is applied to the “top” level physical interface. The *interfaces* list specifies the node and addressing information using the *nodes* parameter. If *vlan\_id* is specified as a key/value

in the interfaces dict, the list dict keys are applied to the nested physical interface VLAN.

#### Parameters

- **interface\_id** (*str*) – id for interface
- **interface** (*str*) – specifies the type of interface to create. The interface type defaults to 'node\_interface' and applies to all engine types except a single engine. For single engine, specify *single\_node\_interface*
- **interfaces** (*list*) – interface attributes, *cluster\_virtual*, *network\_value*, *nodes*, etc
- **nodes** (*dict*) – nodes dict should contain keys *address*, *network\_value* and *nodeid*. Overridden sub interface settings can also be set here
- **zone\_ref** (*str*) – zone reference, name or zone
- **comment** (*str*) – comment for interface

#### 14.6.5.9 ClusterPhysicalInterface

**class** smc.core.interfaces.ClusterPhysicalInterface(*engine=None, meta=None, \*\*interface*)

Bases: *PhysicalInterface*

A ClusterPhysicalInterface represents an interface on a cluster that is a physical interface type. A cluster interface can have a CVI, NDI's, or CVI's and NDI's.

Example interface format, with CVI and 2 nodes:

```
interface = {
    'interface_id': '23',
    'comment': 'my comment',
    'zone_ref': 'zone1',
    'cvi_mode': 'packetdispatch',
    'macaddress': '02:08:08:02:02:06',
    'interfaces': [{ 'cluster_virtual': '241.241.241.250',
                     'network_value': '241.241.241.0/24',
                     'nodes': [{ 'address': '241.241.241.2',
                                  'network_value': '241.241.241.0/24', 'nodeid': 1},
                               { 'address': '241.241.241.3',
                                  'network_value': '241.241.241.0/24', 'nodeid': 2}]
                   }
    ]
}
```

Example interface with VLAN and CVI / NDI::

```
interface = {
    'interface_id': '24',
    'cvi_mode': 'packetdispatch',
    'macaddress': '02:02:08:08:08:06',
    'interfaces': [{ 'cluster_virtual': '242.242.242.250',
                     'network_value': '242.242.242.0/24',
                     'nodes': [{ 'address': '242.242.242.2',
                                  'network_value': '242.242.242.0/24', 'nodeid': 1},
                               { 'address': '242.242.242.3',
                                  'network_value': '242.242.242.0/24', 'nodeid': 2}]
                   },
    'vlan_id': 24,
    'zone_ref': 'vlanzone',
}
```

(continues on next page)

(continued from previous page)

```
        'comment': 'comment on vlan'}]],  
'zone_ref': zone_helper('myzone'),  
'comment': 'top level interface'}
```

When an interface is created, the first key level is applied to the “top” level physical interface. The *interfaces* list specifies the node and addressing information using the *nodes* parameter. If *vlan\_id* is specified as a key/value in the interfaces dict, the list dict keys are applied to the nested physical interface VLAN.

#### Parameters

- **interface\_id** (*str*) – id for interface
- **cvi\_mode** – cvi mode type (i.e. packetdispatch), required when using CVI
- **macaddress** (*str*) – mac address for top level physical interface. Required if CVI set
- **interfaces** (*list*) – interface attributes, *cluster\_virtual*, *network\_value*, *nodes*, etc
- **nodes** (*dict*) – nodes dict should contain keys *address*, *network\_value* and *nodeid*. Overridden sub interface settings can also be set here
- **zone\_ref** (*str*, *href*) – zone reference, name or zone. If zone does not exist it will be created
- **comment** (*str*) – comment for interface

---

**Note:** Values for dict match the FirewallCluster.create constructor

---

#### property cvi\_mode

HA Cluster mode.

##### Returns

possible values: packetdispatch, unicast, multicast, multicastgmp

##### Return type

*str*

#### property macaddress

MAC Address for cluster virtual interface. Not required for NDI only interfaces.

##### Parameters

**value** (*str*) – macaddress

##### Return type

*str*

### 14.6.5.10 VirtualPhysicalInterface

**class** smc.core.interfaces.VirtualPhysicalInterface(*engine=None, meta=None, \*\*interface*)

Bases: *Layer3PhysicalInterface*

This interface type is used by virtual engines and has subtle differences to a normal interface. For a VE in layer 3 firewall, it also specifies a Single Node Interface as the physical interface sub-type. When creating the VE, one of the interfaces must be designated as the source for Auth Requests and Outgoing.

### 14.6.5.11 SwitchPhysicalInterface

**class** `smc.core.interfaces.SwitchPhysicalInterface(engine=None, meta=None, **interface)`

Bases: *Interface*

A switch physical interface is a new dedicated physical module supported on N110 appliances at the time of this document. Check the latest updated spec sheets to determine if your physical appliance currently supports this module

Represents a routed layer 3 interface on an any engine type.

Example interface:

```
{'interface_id': u'SWP_0.1',
 'interfaces': [{ 'nodes': [{ 'dynamic': True,
                               'dynamic_index': 2}]}],
 'switch_physical_interface_port': [{u'switch_physical_interface_port_comment': u'',
                                       u'switch_physical_interface_port_number': 0}],
 'zone_ref': u'External'},
{'interface_id': u'SWP_0.2',
 'interfaces': [{ 'nodes': [{ 'dynamic': True,
                               'dynamic_index': 3}]}],
 'switch_physical_interface_port': [{u'switch_physical_interface_port_comment': u'',
                                       u'switch_physical_interface_port_number': 1}],
 'zone_ref': u'External'},
{'interface_id': u'SWP_0.3',
 'interfaces': [{ 'nodes': [{ 'dynamic': True,
                               'dynamic_index': 4}]}],
 'switch_physical_interface_port': [{u'switch_physical_interface_port_comment': u'',
                                       u'switch_physical_interface_port_number': 3}],
 'zone_ref': u'External'},
{'interface_id': u'SWP_0.4',
 'switch_physical_interface_port': [{u'switch_physical_interface_port_comment': u'
→ port 2',
                                     u'switch_physical_interface_port_number': 2},
                                   {u'switch_physical_interface_port_comment': u'',
                                     u'switch_physical_interface_port_number': 4},
                                   {u'switch_physical_interface_port_comment': u'',
                                     u'switch_physical_interface_port_number': 5},
                                   {u'switch_physical_interface_port_comment': u'',
                                     u'switch_physical_interface_port_number': 6}]}
```

#### Variables

**switch\_physical\_module** (*ApplianceSwitchModule*) – appliance module type

#### property **appliance\_switch\_module**

Return the appliance module used for this switch physical interface.

#### Return type

*ApplianceSwitchModule*

#### **update\_interface**(*other\_interface*, *ignore\_mgmt=True*)

Update a switch physical interface with another interface. You can provide only partial interface data, for example, if you have an existing port group and you want to add additional ports. Or if you want to change the zone assigned to a single port group. There is nothing that can be modified on the top level switch interface itself, only the nested port groups.

If the intent is to delete a port\_group, retrieve the port group interface and call delete().

#### Parameters

- **other\_interface** (*SwitchPhysicalInterface*) – interface to use for modifications
- **ignore\_mgmt** (*bool*) – ignore management settings

#### 14.6.5.12 TunnelInterface

**class** smc.core.interfaces.TunnelInterface(engine=None, meta=None, \*\*interface)

Bases: *Interface*

This interface type represents a tunnel interface that is typically used for route based VPN traffic. Nested interface nodes can be SingleNodeInterface (for L3 NGFW), a NodeInterface (for cluster's with only NDI's) or ClusterVirtualInterface (CVI) for cluster VIP. Tunnel Interfaces are only available under layer 3 routed interfaces and do not support VLANs.

Example tunnel interface format on cluster:

```
cluster_tunnel_interface = {
    'comment': u'My Tunnel on cluster',
    'interface_id': u'1000',
    'interfaces': [{ 'cluster_virtual': u'77.77.77.70',
                     'network_value': u'77.77.77.0/24',
                     'nodes': [{ 'address': u'5.5.5.2',
                                  'network_value': u'5.5.5.0/24',
                                  'nodeid': 1},
                               { 'address': u'5.5.5.3',
                                  'network_value': u'5.5.5.0/24',
                                  'nodeid': 2}]]},
    'zone_ref': 'foozone'}
```

Tunnel interface on single engine with multiple tunnel IPs:

```
single_fw_interface = {
    'comment': u'Tunnel with two addresses on single engine',
    'interface_id': u'1000',
    'interfaces': [{ 'nodes': [{ 'address': u'5.5.5.2',
                                  'network_value': u'5.5.5.0/24',
                                  'nodeid': 1},
                               { 'address': u'5.5.5.3',
                                  'network_value': u'5.5.5.0/24',
                                  'nodeid': 1}]}],
    'zone_ref': 'foozone'}
```

#### property qos

The QoS settings for this tunnel interface

#### Return type

*QoS*

#### 14.6.5.13 VPNBrokerInterface

**class** `smc.core.interfaces.VPNBrokerInterface(engine=None, meta=None, **interface)`

Bases: [Interface](#)

This interface type represents a VPN broker interface. Not meant to be used directly: instead use:

`engine.vpn_broker_interface.add_layer3_interface` to create a `VPNBrokerInterface`

Usage:

```
itf = {'nodes': [{'address': u'10.88.0.12',
                  'network_value': u'10.88.0.0/24', 'nodeid': 1, "nicid": "VPN_0"}
      ]}

my_interface = VPNBrokerInterface(
    zone_ref="Internal", interface_id="VPN_0", comment="testing vpn broker", vpn_broker_domain_ref =
    brokdom.href, mac_address_postfix="a3:b3:c3", retrieve_routes=True, shared_secret="blabla", interfaces
    = [itf])

print(json.dumps(dict(my_interface.data)))
```

#### 14.6.5.14 VlanInterface

**class** `smc.core.interfaces.VlanInterface`

Bases: [object](#)

`VlanInterface` is a dynamic class generated by collections referencing interfaces with vlan interfaces. The inheriting class for a `VlanInterface` is dependent on the parent interface.

**change\_vlan\_id**(*vlan\_id*)

Change the VLAN id for this VLAN interface. If this is an inline interface, you can specify two interface values to create unique VLANs on both sides of the inline pair. Or provide a single to use the same VLAN id.

**Parameters**

**vlan\_id** (*str*) – string value for new VLAN id.

**Raises**

[UpdateElementFailed](#) – failed to update the VLAN id

**Returns**

None

**delete()**

Delete this Vlan interface from the parent interface. This will also remove stale routes if the interface has networks associated with it.

**Returns**

None

**property qos**

QoS on VLANs is only supported for `Layer3PhysicalInterface` types.

**Return type**

[QoS](#)

**property** `vlan_id`

Vlan id for this vlan

**Return type**`str`

### 14.6.5.15 Sub-Interfaces

Module provides an interface to sub-interfaces on an engine. A ‘top level’ interface is linked from the engine and will be `PhysicalInterface`, `TunnelInterface`, etc. Within the top level interface, there are sub-interface configurations that identify the basic settings such as ip address, network, administrative settings etc. These are not called directly but used as a reference to the top level interface. All sub interfaces are type dict.

**class** `smc.core.sub_interfaces.CaptureInterface(data)`

Capture Interface (SPAN) This is a single physical interface type that can be installed on either layer 2 or IPS engine roles. It enables the NGFW to capture traffic on the wire without actually blocking it (although blocking is possible).

**Variables**

- **inspect\_unspecified\_vlans** (*boolean*) – promiscuous SPAN on unspecified VLANs
- **(required)** (*str logical\_interface\_ref*) – logical interface to use, by href
- **reset\_interface\_nicid** (*int*) – if sending passive RST back, interface id to use
- **nicid** (*str, int*) – nicid for this capture interface

**class** `smc.core.sub_interfaces.ClusterVirtualInterface(data)`

These interfaces (CVI) are used on cluster devices and applied to layer 3 interfaces. They specify a ‘VIP’ (or shared IP) to be used for traffic load balancing or high availability. Each engine will still also have a ‘node’ interface for communication to/from the engine itself. The following getter/setter properties are available:

**Variables**

- **address** (*str*) – address of the CVI
- **auth\_request** (*boolean*) – interface for authentication requests (only 1)
- **network\_value** (*str*) – network address for interface, i.e. 1.1.1.0/24
- **nicid** (*int*) – nic interface identifier
- **relayed\_by\_dhcp** (*boolean*) – is the interface using DHCP
- **igmp\_mode** (*str*) – IGMP mode (upstream/downstream/None)

**property** `vlan_id`

VLAN ID for this interface, if any

**Returns**

VLAN identifier

**Return type**`str`**class** `smc.core.sub_interfaces.InlineIPSInterface(data)`

An Inline IPS Interface is a new interface type introduced in SMC version 6.3. This interface type is the same as a normal IPS interface except that it is applied on a Layer 3 Firewall.

New in version 0.5.6: Requires SMC 6.3.

**class** `smc.core.sub_interfaces.InlineInterface(data)`

This interface type is used on layer 2 or IPS related engines. It requires that you specify two interfaces to be part of the inline pair. These interfaces do not need to be sequential. It is also possible to add VLANs and zones to the inline interfaces. The logical interface reference needs to be unique for inline and capture interfaces when they are applied on the same engine.

#### Variables

- **inspect\_unspecified\_vlans** (*boolean*) – promiscuous SPAN on unspecified VLANs
- **(required)** (*str logical\_interface\_ref*) – logical interface to use, by href
- **failure\_mode** (*str*) – normal or bypass
- **nicid** (*str*) – interfaces for inline pair, for example, '4.50-5.55', '5-6'
- **vlan\_id** (*str*) – vlan identifier for interface
- **(optional)** (*str zone\_ref*) – zone for second interface in pair

**change\_interface\_id**(*newid*)

Change the inline interface ID. The current format is nicid='1-2', where '1' is the top level interface ID (first), and '2' is the second interface in the pair. Consider the existing nicid in case this is a VLAN.

#### Parameters

**newid** (*str*) – string defining new pair, i.e. '3-4'

#### Returns

None

**change\_vlan\_id**(*vlan\_id*)

Change a VLAN id for an inline interface.

#### Parameters

**vlan\_id** (*str*) – New VLAN id. Can be in format '1-2' or a single numerical value. If in '1-2' format, this specifies the vlan ID for the first inline interface and the rightmost for the second.

#### Returns

None

**property** `vlan_id`

VLAN ID for this interface, if any

#### Returns

VLAN identifier

#### Return type

*str*

**class** `smc.core.sub_interfaces.InlineL2FWInterface(data)`

An Inline L2FW Interface is a new interface type introduced in SMC version 6.3. This interface type is the a layer 2 engine interface on a layer 3 firewall. By default this interface type does not support bypass mode and will discard on overload.

New in version 0.5.6: Requires SMC 6.3.

**class** `smc.core.sub_interfaces.LoopbackClusterInterface(data, engine=None)`

This represents the CVI Loopback IP address. A CVI loopback IP address is used for loopback traffic that is sent to the whole cluster. It is shared by all the nodes in the cluster.

**add\_cvi\_loopback**(*address*, *ospf\_area=None*, *\*\*kw*)

Add a loopback interface as a cluster virtual loopback. This enables the loopback to ‘float’ between cluster members. Changes are committed immediately.

**Parameters**

- **address** (*str*) – ip address for loopback
- **rank** (*int*) – rank of this entry
- **ospf\_area** (*str*, *Element*) – optional ospf\_area to add to loopback

**Raises**

*UpdateElementFailed* – failure to save loopback address

**Returns**

None

**delete()**

Delete a loopback cluster virtual interface from this engine. Changes to the engine configuration are done immediately.

You can find cluster virtual loopbacks by iterating at the engine level:

```
for loopbacks in engine.loopback_interface:
    ...
```

**Raises**

*UpdateElementFailed* – failure to delete loopback interface

**Returns**

None

**class** `smc.core.sub_interfaces.LoopbackInterface`(*data*, *engine=None*)

Loopback interface for a physical or virtual single firewall. To create a loopback interface, call from the engine node:

```
engine.loopback_interface.add_single(...)
```

**add\_node\_loopback**(*addresses*, *\*\*kwargs*)

Add a node loopback interface to this engine.

**Parameters**

**addresses** (*dict*) – {nodeid: {‘address’: ‘127.0.0.1’, ‘ospf\_area’: ospfArea}, ... }

**Raises**

*UpdateElementFailed* – failure to create loopback address

**Returns**

None

**add\_single**(*address*, *rank=1*, *nodeid=1*, *ospf\_area=None*, *\*\*kwargs*)

Add a single loopback interface to this engine. This is used for single or virtual engines.

**Parameters**

- **address** (*str*) – ip address for loopback
- **nodeid** (*int*) – nodeid to apply. Default to 1 for single engine
- **ospf\_area** (*str*, *Element*) – ospf area href or element

**Raises**

*UpdateElementFailed* – failure to create loopback address

**Returns**

None

**delete()**

Delete a loopback interface from this engine. Changes to the engine configuration are done immediately.

A simple way to obtain an existing loopback is to iterate the loopbacks or to get by address:

```
lb = engine.loopback_interface.get('127.0.0.10')
lb.delete()
```

**Warning:** When deleting a loopback assigned to a node on a cluster all loopbacks with the same rank will also be removed.

**Raises**

*UpdateElementFailed* – failure to delete loopback interface

**Returns**

None

**class** smc.core.sub\_interfaces.**NodeInterface**(data)

Node Interface Node dedicated interface (NDI) is used on specific engine types and represents an interface used for management (IPS and layer 2 engines), or as normal layer 3 interfaces such as on a layer 3 firewall cluster.

For Layer 2 Firewall/IPS these are used as individual interfaces. On clusters, these are used to define the node specific address for each node member, along with a cluster virtual interface.

**Variables**

- **address** (*str*) – ip address of this interface
- **network\_value** (*str*) – network for this interface, i.e. 1.1.1.0/24
- **nicid** (*str* or *int*) – nic interface id
- **nodeid** (*int*) – node identifier for interface (in a cluster, each node will be unique)
- **outgoing** (*boolean*) – This option defines the IP address that the nodes use if they have to initiate connections (system communications, ping, etc.) through an interface that has no Node Dedicated IP Address. In Firewall Clusters, you must select an interface that has an IP address defined for all nodes.
- **primary\_heartbeat** (*boolean*) – Whether interface is the primary heartbeat interface for communications between the nodes. It is recommended that you use a Physical Interface, not a VLAN Interface. It is also recommended that you do not direct any other traffic through this interface.
- **primary\_mgt** (*boolean*) – Is it the Primary Control Interface for Management Server contact. There must be one and only one Primary Control Interface
- **auth\_request** (*boolean*) – whether to specify this interface as interface for authentication requests. Should be set on interface acting as management
- **auth\_request\_source** (*boolean*) – If the authentication requests are sent to an external authentication server over VPN, select an interface with a Node Dedicated IP address that you want use for the authentication requests

- **reverse\_connection** (*boolean*) – Reverse connection enables engine to contact SMC versus other way around
- **vlan\_id** (*str*) – VLAN id for interface if assigned
- **backup\_mgt** (*boolean*) – Whether interface is a backup control interface that is used if the primary control interface is not available
- **backup\_heartbeat** (*boolean*) – Whether the interface is a backup heartbeat. It is not mandatory to configure a backup heartbeat interface.
- **dynamic** (*boolean*) – Whether this is a DHCP interface
- **dynamic\_index** (*int*) – The dynamic index of the DHCP interface. The value is between 1 and 16. Only used when ‘dynamic’ is set to True.
- **igmp\_mode** (*str*) – IGMP mode (upstream/downstream/None)
- **vrrp** (*boolean*) – Enable VRRP
- **vrrp\_address** (*str*) – IP address if VRRP is enabled
- **vrrp\_id** (*int*) – The VRRP ID. Required only for VRRP mode
- **vrrp\_priority** (*int*) – The VRRP Priority. Required only for VRRP mode

**property** `vlan_id`

VLAN ID for this interface, if any

**Returns**

VLAN identifier

**Return type**

*str*

**class** `smc.core.sub_interfaces.SingleNodeInterface`(*data*)

This interface is used by single node Layer 3 Firewalls. This type of interface can be a management interface as well as a non-management routed interface.

**Variables**

- **dynamic** (*bool*) – is this interface a dynamic DHCP interface
- **dynamic\_index** (*int*) – dynamic interfaces index value
- **automatic\_default\_route** (*boolean*) – Flag to know if the dynamic default route will be automatically created for this dynamic interface. Used in DHCP interfaces only

**class** `smc.core.sub_interfaces.SubInterface`(*data*)

**change\_interface\_id**(*interface\_id*)

Generic change interface ID for VLAN interfaces that are not Inline Interfaces (non-VLAN sub interfaces do not have an `interface_id` field).

**Parameters**

**interface\_id** (*str*, *int*) – interface ID value

**change\_vlan\_id**(*vlan\_id*)

Change a VLAN id

**Parameters**

**vlan\_id** (*str*) – new vlan

**class** `smc.core.sub_interfaces.SubInterfaceCollection`(*interface*)

A Sub Interface collection for non-VLAN interfaces.

#### 14.6.5.16 InterfaceContactAddress

A ContactAddress is used by elements to provide an alternate address for communication between engine and management/log server. This is typically used when the SMC sits behind a NAT address and the SMC needs to contact the engine directly (this is a default behavior). In this case, you would add the public IP in front of the engine as a contact address to the engine interface.

Obtain all eligible interfaces for contact addresses:

```
>>> engine = Engine('dingo')
>>> for ca in engine.contact_addresses:
...     ca
...
ContactAddressNode(interface_id=11, interface_ip=10.10.10.20)
ContactAddressNode(interface_id=120, interface_ip=120.120.120.100)
ContactAddressNode(interface_id=0, interface_ip=1.1.1.1)
ContactAddressNode(interface_id=12, interface_ip=3.3.3.3)
ContactAddressNode(interface_id=12, interface_ip=17.17.17.17)
```

Retrieve a specific contact address interface for modification:

```
>>> ca = engine.contact_addresses.get(interface_id=12, interface_ip='3.3.3.3')
>>> ca
ContactAddressNode(interface_id=12, interface_ip=3.3.3.3)
>>> list(ca)
[InterfaceContactAddress(location=Default,address=4.4.4.4),
 InterfaceContactAddress(location=Foo,address=3.4.5.6)]
```

Add a new contact address to the fetched interface:

```
>>> ca.add_contact_address('23.23.23.23', location='mynewlocation')
>>> list(ca)
[InterfaceContactAddress(location=Default,address=4.4.4.4),
 InterfaceContactAddress(location=Foo,address=3.4.5.6),
 InterfaceContactAddress(location=mynewlocation,address=23.23.23.
↪23)]
```

Remove a contact address:

```
>>> ca.remove_contact_address('23.23.23.23')
>>> list(ca)
[InterfaceContactAddress(location=Default,address=4.4.4.4),
 InterfaceContactAddress(location=Foo,address=3.4.5.6)]
```

---

**Note:** Contact Addresses for servers (Management/Log Server) do not use this same object definition

---

**class** smc.core.contact\_address.ContactAddressCollection(resource)

Bases: *SubElementCollection*

A contact address collection provides all available interfaces that can be used to configure a contact address. An eligible interface is one that is a layer 3 interface with an address assigned (including VLANs):

```
for ca in engine.contact_addresses:
    ...
```

---

**Note:** All eligible interfaces are returned, regardless of whether a contact address is assigned or not.

---

**get**(*interface\_id*, *interface\_ip=None*)

Get will return a list of interface references based on the specified interface id. Multiple references can be returned if a single interface has multiple IP addresses assigned.

**Returns**

If *interface\_ip* is provided, a single `ContactAddressNode` element is returned if found. Otherwise a list will be returned with all contact address nodes for the given *interface\_id*.

**class** `smc.core.contact_address.ContactAddressNode`(*\*\*meta*)

Bases: `SubElement`

A mapping of contact address to interface. This is specific to assigning the contact address on the engine.

**add\_contact\_address**(*contact\_address*, *location='Default'*, *dynamic=False*)

Add a contact address to this specified interface. A contact address is an alternative address which is typically applied when NAT is used between the NGFW and another component (such as management server). Adding a contact address operation is committed immediately.

**Parameters**

- **contact\_address** (*str*) – IP address for this contact address.
- **location** (*str*) – name of the location, the location will be added if it doesn't exist
- **dynamic** (*bool*) – flag to specify the dynamic FQDN contact address.

**Raises**

`EngineCommandFailed` – invalid contact address

**Returns**

`ContactAddressNode`

As example for dynamic FQDN:

```
add_contact_address(contract_address="your.fqdn",
                    location=Location("SpecificLocation"), dynamic=True)
```

As example for dynamic:

```
update_or_create(contract_address="dynamic",
                 location=Location("SpecificLocation"))
```

**delete**(*location\_name*)

Remove a given location by location name. This operation is performed only if the given location is valid, and if so, *update* is called automatically.

**Parameters**

**location** (*str*) – location name or location ref

**Raises**

`UpdateElementFailed` – failed to update element with reason

**Return type**

`bool`

**property** `interface_id`

The interface ID for this contact address interface

**Return type**`str`**property interface\_ip**

The IP address for this contact address interface

**Return type**`str`**remove\_contact\_address**(*location*)

Remove a contact address from an interface by the location name. There is a one to one relationship between a contact address and

**Parameters**

**contact\_address** (`str`) – ip for contact address

**Raises**

*EngineCommandFailed* – problem removing address

**Returns**

status of delete as boolean

**Return type**`bool`**update\_or\_create**(*location*, *contact\_address*, *dynamic=False*, *with\_status=False*, *\*\*kw*)

Update an existing contact address or create if the location does not exist.

**Parameters**

- **location** (`str`) – name of the location, the location will be added if it doesn't exist
- **contact\_address** (`str`) – contact address IP. Can be the string 'dynamic' if this should be a dynamic contact address (i.e. on DHCP interface)
- **dynamic** (`bool`) – flag to specify the dynamic FQDN contact address.
- **with\_status** (`bool`) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

**Raises**

*UpdateElementFailed* – failed to update element with reason

**Return type***ContactAddressNode*

As example for dynamic FQDN:

```
update_or_create(location=Location("SpecificLocation"),
                 contract_address="your.fqdn", dynamic=True)
```

As example for dynamic:

```
update_or_create(location=Location("SpecificLocation"),
                 contract_address="dynamic")
```

```
class smc.core.contact_address.InterfaceContactAddress(data=None, **kwargs)
```

Bases: *ContactAddress*

An interface contact address is used on engine interfaces to provide an alternative location to address mapping. This is frequently used when the engine sits behind a NAT and you need a public NAT mapping, as might be the case with site to site VPN.

**property addresses**

List of addresses set as contact address

**Return type**

`list`

## 14.6.6 Node

Node level actions for an engine. Once an engine is loaded, all methods and resources are available to that particular engine.

For example, to load an engine and run node level commands:

```
engine = Engine('myfw')
for node in engine.nodes:
    node.reboot()
    node.bind_license()
    node.go_online()
    node.go_offline()
    ...
    ...
```

**class** `smc.core.node.MasterNode(**meta)`

Bases: `Node`

This represents an individual Master NGFW Engine node in the Security Management Client, representing a device that runs firewall software as part of a Master NGFW Engine.

**class** `smc.core.node.Node(**meta)`

Bases: `SubElement`

Node settings to make each engine node controllable individually. Obtain a reference to a Node by loading an Engine resource. Engine will have a ‘has-a’ relationship with node and stored as the nodes attribute.

```
>>> for node in engine.nodes:
...     node
...
Node(name=fwcluster node 1)
Node(name=fwcluster node 2)
```

**appliance\_info()**

New in version 0.5.7: Requires SMC version >= 6.3

Retrieve appliance info for this engine.

**Raises**

`NodeCommandFailed` – Appliance info not supported on this node

**Return type**

`ApplianceInfo`

**bind\_license**(`license_item_id=None`, `**kwargs`)

Auto bind license, uses dynamic if POS is not found

**Parameters**

**license\_item\_id**(`str`) – license id

**Raises**

[\*LicenseError\*](#) – binding license failed, possibly no licenses

**Returns**

None

**cancel\_unbind\_license**(\*\*kwargs)

Cancel unbind for license

**Raises**

[\*LicenseError\*](#) – unbind failed with reason

**Returns**

None

**certificate\_info**()

Get the certificate info of this node. This can return None if the engine type does not directly have a certificate, like a virtual engine where the master engine manages certificates.

**Returns**

dict with links to cert info

**change\_ssh\_pwd**(pwd=None, comment=None)

Executes a change SSH password operation on the specified node

**Parameters**

- **pwd** (*str*) – changed password value
- **comment** (*str*) – optional comment for audit log

**Raises**

[\*NodeCommandFailed\*](#) – cannot change ssh password

**Returns**

None

**debug**(filter\_enabled=False)

View all debug settings for this node. This will return a debug object. View the debug object repr to identify settings to enable or disable and submit the object to [\*set\\_debug\(\)\*](#) to enable settings.

Add filter\_enabled=True argument to see only enabled settings

**Parameters**

**filter\_enabled** (*bool*) – returns all enabled diagnostics

**Raises**

[\*NodeCommandFailed\*](#) – failure getting diagnostics

**Return type**

[\*Debug\*](#)

**See also:**

[\*Debug\*](#) for example usage

**fetch\_license**(\*\*kwargs)

Fetch the node level license

**Raises**

[\*LicenseError\*](#) – fetching license failure with reason

**Returns**

None

**go\_offline**(*comment=None*)

Executes a Go-Offline operation on the specified node

**Parameters**

**comment** (*str*) – optional comment to audit

**Raises**

[\*NodeCommandFailed\*](#) – offline not available

**Returns**

None

**go\_online**(*comment=None*)

Executes a Go-Online operation on the specified node typically done when the node has already been forced offline via [\*go\\_offline\(\)\*](#)

**Parameters**

**comment** (*str*) – (optional) comment to audit

**Raises**

[\*NodeCommandFailed\*](#) – online not available

**Returns**

None

**go\_standby**(*comment=None*)

Executes a Go-Standby operation on the specified node. To get the status of the current node/s, run [\*status\(\)\*](#)

**Parameters**

**comment** (*str*) – optional comment to audit

**Raises**

[\*NodeCommandFailed\*](#) – engine cannot go standby

**Returns**

None

**property hardware\_status**

Obtain hardware statistics for various areas of this node.

See [\*HardwareStatus\*](#) for usage.

**Raises**

[\*NodeCommandFailed\*](#) – failure to retrieve current status

**Return type**

[\*HardwareStatus\*](#)

**property health**

Basic status for individual node. Specific information such as node name dynamic package version, configuration status, platform and version.

**Return type**

[\*ApplianceStatus\*](#)

**initial\_contact**(*enable\_ssh=True, time\_zone=None, keyboard=None, install\_on\_server=None, filename=None, as\_base64=False*)

Allows to save the initial contact for for the specified node

**Parameters**

- **enable\_ssh** (*bool*) – flag to know if we allow the ssh daemon on the specified node
- **time\_zone** (*str*) – optional time zone to set on the specified node
- **keyboard** (*str*) – optional keyboard to set on the specified node
- **install\_on\_server** (*bool*) – optional flag to know if the generated configuration needs to be installed on SMC Install server (POS is needed)
- **filename** (*str*) – filename to save initial\_contact to
- **as\_base64** (*bool*) – return the initial config in base 64 format. Useful for cloud based engine deployments as userdata

**Raises**

*NodeCommandFailed* – IOError handling initial configuration data

**Returns**

initial contact text information

**Return type**

*str*

**property interface\_status**

Obtain the interface status for this node. This will return an iterable that provides information about the existing interfaces. Retrieve a single interface status:

```
>>> node = engine.nodes[0]
>>> node
Node(name=ngf-1065)
>>> node.interface_status
<smc.core.node.InterfaceStatus object at 0x103b2f310>
>>> node.interface_status.get(0)
InterfaceStatus(aggregate_is_active=False, capability=u'Normal Interface',
                flow_control=u'AutoNeg: off Rx: off Tx: off',
                interface_id=0, mtu=1500, name=u'eth0_0', port=u'Copper',
                speed_duplex=u'1000 Mb/s / Full / Automatic', status=u'Up')
```

Or iterate and get all interfaces:

```
>>> for stat in node.interface_status:
...     stat
...
InterfaceStatus(aggregate_is_active=False, capability=u'Normal Interface', ...
...)
```

**Raises**

*NodeCommandFailed* – failure to retrieve current status

**Return type**

*InterfaceStatus*

**lock\_offline**(*comment=None*)

Executes a Lock-Offline operation on the specified node Bring back online by running *go\_online()*.

**Parameters**

**comment** (*str*) – comment for audit

**Raises**

*NodeCommandFailed* – lock offline failed

**Returns**

None

**lock\_online**(*comment=None*)

Executes a Lock-Online operation on the specified node

**Parameters**

**comment** (*str*) – comment for audit

**Raises**

*NodeCommandFailed* – cannot lock online

**Returns**

None

**property loopback\_interface**

Loopback interfaces for this node. This will return empty if the engine is not a layer 3 firewall type:

```
>>> engine = Engine('dingo')
>>> for node in engine.nodes:
...     for loopback in node.loopback_interface:
...         loopback
...
LoopbackInterface(address=172.20.1.1, nodeid=1, rank=1)
LoopbackInterface(address=172.31.1.1, nodeid=1, rank=2)
LoopbackInterface(address=2.2.2.2, nodeid=1, rank=3)
```

**Return type**

*list(LoopbackInterface)*

**property nodeid**

ID of this node

**pki\_certificate\_info()**

Get the certificate info of this component when working with External PKI. This can return None if the component does not directly have a certificate, like a virtual engine where the master engine manages certificates.

**Raises**

*UnsupportedEngineFeature* – requires layer 3 engine

**Return type**

PkiCertificateInfo

**pki\_certificate\_settings()**

Get the certificate info of this node when working with External PKI. This can return None if the engine type does not directly have a certificate,

like a virtual engine where the master engine manages certificates.

**Raises**

*UnsupportedEngineFeature* – requires engine version 6.10 and external PKI

installation :rtype: PkiCertificateSettings

**pki\_delete\_certificate\_request()**

Delete the certificate request if any is defined for this component.

**pki\_export\_certificate\_request(filename=None)**

Export the certificate request for the node when working with an External PKI. This can return None if the engine type does not have a certificate request.

**Raises**

***CertificateExportError*** – error exporting certificate

**Return type**

*str* or None

**pki\_import\_certificate(certificate)**

Import a valid certificate. Certificate can be either a file path or a string of the certificate. If string certificate, it must include the —BEGIN CERTIFICATE— string.

**Parameters**

**certificate** (*str*) – fully qualified path or string

**Raises**

- ***CertificateImportError*** – failure to import cert with reason
- ***IOError*** – file not found, permissions, etc.

**pki\_renew\_certificate()**

Start renewal process on component when using external PKI mode. It generates new private key and prepares a new certificate request.

**power\_off()**

New in version 0.5.6: Requires engine version >=6.3

Power off engine.

**Raises**

***NodeCommandFailed*** – online not available

**Returns**

None

**reboot(comment=None)**

Send reboot command to this node.

**Parameters**

**comment** (*str*) – comment to audit

**Raises**

***NodeCommandFailed*** – reboot failed with reason

**Returns**

None

**rename(name)**

Rename this node

**Parameters**

**name** (*str*) – new name for node

**reset\_to\_factory()**

New in version 0.5.6: Requires engine version >=6.3

Reset the engine to factory defaults.

**Raises**

[\*NodeCommandFailed\*](#) – online not available

**Returns**

None

**reset\_user\_db**(*comment=None*)

Executes a Send Reset LDAP User DB Request operation on this node.

**Parameters**

**comment** (*str*) – comment to audit

**Raises**

[\*NodeCommandFailed\*](#) – failure resetting db

**Returns**

None

**set\_debug**(*debug*)

Set the debug settings for this node. This should be a modified [\*Debug\*](#) instance. This will take effect immediately on the specified node.

**Parameters**

**debug** ([\*Debug\*](#)) – debug object with specified settings

**Raises**

[\*NodeCommandFailed\*](#) – fail to communicate with node

**Returns**

None

**See also:**

[\*Debug\*](#) for example usage

**sginfo**(*include\_core\_files=False, include\_slapcat\_output=False, filename='sginfo.gz'*)

Get the SG Info of the specified node. Optionally provide a filename, otherwise default to 'sginfo.gz'. Once you run `gzip -d <filename>`, the inner contents will be in .tar format.

**Parameters**

- **include\_core\_files** – flag to include or not core files
- **include\_slapcat\_output** – flag to include or not slapcat output

**Raises**

[\*NodeCommandFailed\*](#) – failed getting sginfo with reason

**Returns**

string path of download location

**Return type**

*str*

**ssh**(*enable=True, comment=None*)

Enable or disable SSH

**Parameters**

- **enable** (*bool*) – enable or disable SSH daemon
- **comment** (*str*) – optional comment for audit

**Raises**

[\*NodeCommandFailed\*](#) – cannot enable SSH daemon

**Returns**

None

**status()**

Basic status for individual node. Specific information such as node name dynamic package version, configuration status, platform and version.

**Return type***ApplianceStatus***time\_sync()**

Send a time sync command to this node.

**Raises***NodeCommandFailed* – time sync not supported on node**Returns**

None

**property type**

Node type

**unbind\_license(\*\*kwargs)**

Unbind a bound license on this node.

**Raises***LicenseError* – failure with reason**Returns**

None

**property version**

Engine version. If the node is not yet initialized, this will return None.

**Returns**

str or None

### 14.6.6.1 Appliance Info

```
class smc.core.node.ApplianceInfo(cloud_id, cloud_type, first_upload_time, hardware_version,
                                   initial_contact_time, product_name, initial_license_remaining_days,
                                   proof_of_serial, software_features, software_version)
```

Bases: *tuple*

Appliance specific information about the given engine node. Appliance info is specific to the engine itself and will provide additional details about the hardware model, applied license features, if the engine has made initial contact and when initial policy upload was made.

Retrieve appliance info engine nodes:

```
engine = Engine('dingo')
for node in engine.nodes:
    node.appliance_info()
```

**Variables**

- **cloud\_id** (*str*) – N/A
- **cloud\_type** (*str*) – N/A

- **first\_upload\_time** (*long*) – policy first upload time in ms
- **hardware\_version** (*float*) – hardware version of appliance
- **initial\_contact\_time** (*long*) – when node contacted SMC, in ms
- **intial\_license\_remaining\_days** (*int*) – validity in days of current license
- **product\_name** (*str*) – name of hardware model
- **proof\_of\_serial** (*str*) – proof of serial for this hardware
- **software\_features** (*str*) – feature string
- **software\_version** (*str*) – initial software version on base image

#### 14.6.6.2 Appliance Status

**class** smc.core.node.ApplianceStatus(*data*)

Bases: `NestedDict`

Appliance status attributes define specifics about the hardware platform itself, including version, dynamic package, current configuration status and installed policy. Retrieve appliance status for engine nodes:

```
for node in engine.nodes:
    node.health
```

Changed in version 1.0.1: added master\_node since SMC version >= 6.10 API6.10, 6.9, 6.8

**property** configuration\_status

**Valid values:**

- Initial (no initial configuration file is yet generated)
- Declared (initial configuration file is generated)
- Configured (initial configuration is done with the engine)
- Installed (policy is installed on the engine)

**Returns**

str configuration\_status: configuration status

**property** dyn\_up

**Returns**

str dyn\_up: Dynamic update package version

**property** installed\_policy

**Returns**

str installed\_policy: Installed policy by name

**property** master\_node

The master engine node for a virtual engine

**Returns**

MasterNode: the master node or None

**property name****Returns**

str name: Name of engine

**property platform****Returns**

str platform: Underlying platform, x86, etc

**property state****Valid values:**

INITIAL/READY/ERROR/SERVER\_ERROR/NO\_STATUS/TIMEOUT/ DELETED/DUMMY

**Returns**

str state: state of the Node

**property status**

str status:

**Valid values:**

Not Monitored/Unknown/Online/Going Online/Locked Online/ Going Locked Online/Offline/Going Offline/Locked Offline/ Going Locked Offline/Standby/Going Standby/No Policy Installed

**Returns****property version****Returns**

str version: Version of software installed

### 14.6.6.3 ApplianceSwitchModule

**class** smc.core.hardware.ApplianceSwitchModule(name=None, \*\*meta)Bases: *Element*

Read only class specifying hardware switch modules used in smaller appliance form factors. This is referenced when creating switch interfaces

### 14.6.6.4 Hardware Status

**class** smc.core.node.HardwareStatus(\*args, \*\*kwargs)Bases: *SerializedIterable*

Provides an interface to methods that simplify viewing hardware statuses on this node. Example of usage:

```

>>> engine = Engine('sg_vm')
>>> node = engine.nodes[0]
>>> node
Node(name=ngf-1065)
>>> node.hardware_status
HardwareStatus(Anti-Malware, File Systems, GTI Cloud, Sandbox, Logging subsystem,
                MLC Connection, Web Filtering)
>>> node.hardware_status.filesystem

```

(continues on next page)

(continued from previous page)

```

HardwareCollection(File Systems, items: 5)
>>> for stats in node.hardware_status.filesystem:
...     stats
...
Status(label=u'Root', param=u'Partition Size', status=-1, sub_system=u'File Systems
↪',
        value=u'600 MB')
Status(label=u'Data', param=u'Usage', status=-1, sub_system=u'File Systems',
        value=u'6.3%')
Status(label=u'Data', param=u'Size', status=-1, sub_system=u'File Systems',
        value=u'1937 MB')
Status(label=u'Spool', param=u'Usage', status=-1, sub_system=u'File Systems',
        value=u'4.9%')
Status(label=u'Spool', param=u'Size', status=-1, sub_system=u'File Systems',
        value=u'9729 MB')
Status(label=u'Tmp', param=u'Usage', status=-1, sub_system=u'File Systems',
        value=u'0.0%')
Status(label=u'Tmp', param=u'Size', status=-1, sub_system=u'File Systems',
        value=u'3941 MB')
Status(label=u'Swap', param=u'Usage', status=-1, sub_system=u'File Systems',
        value=u'0.0%')
Status(label=u'Swap', param=u'Size', status=-1, sub_system=u'File Systems',
        value=u'1887 MB')

Since SMC-API >= v6.7
('OK', Status(label='Swap', param='Size', sub_system='File Systems', value='494 MB
↪'))
('WARNING', Status(label='Tmp', param='Usage', sub_system='File Systems', value='96.
↪7%'))
('WARNING', Status(label='Tmp', param='Size', sub_system='File Systems', value='997_
↪MB'))

>>> for stats in node.hardware_status.sandbox_subsystem:
...     stats
...
('WARNING', Status(label='Cloud connection', param='status', sub_system='Sandbox',
                    value='1'))

```

**property cloud\_sync**

A collection of Connection Synchronisation related statuses

**Return type**

Status

**property filesystem**

A collection of filesystem related statuses

**Return type**

Status

**property logging\_subsystem**

A collection of logging subsystem statuses

**Return type**

Status

**property sandbox\_subsystem**

A collection of sandbox subsystem statuses

**Return type**

Status

**property single\_ip\_ha**

A collection of Single IP High Availability related statuses

**Return type**

Status

**14.6.6.5 Interface Status**

**class** `smc.core.node.InterfaceStatus(status)`

Bases: *SerializedIterable*

An iterable that provides a collections interface to interfaces and current status on the specified node.

Interface status fields:

**Variables**

- **aggregate\_is\_active** (*bool*) – Is link aggregation enabled on this interface
- **capability** (*str*) – What type of interface this is, i.e. “Normal Interface”
- **flow\_control** (*str*) – Autonegotiation, etc
- **interface\_id** (*int*) – Physical interface id
- **mtu** (*int*) – Max transmission unit
- **name** (*str*) – Name of the interface, i.e. eth0\_0, etc
- **port** (*str*) – Type of physical port, i.e. Copper, Fiber
- **speed\_duplex** (*str*) – Negotiated speed on the interface
- **status** (*str*) – Status of interface, Up, Down, etc.

**get**(*interface\_id*)

Get a specific interface by the interface id

**Parameters**

**interface\_id** (*int*) – interface ID

**Return type**

*InterfaceStatus*

**14.6.6.6 Debug**

**class** `smc.core.node.Debug(diag)`

Debug settings that can be enabled on the engine. To view available options, print the repr of this object. All diagnostic values can be set as an attribute of this class instance. Set the values to either True or False and submit this object back to the node to change settings. Setting changes are in effect immediately and does not require a policy push. Example usage:

```
>>> node = engine.nodes[0]
>>> node
Node(name=ngf-1065)
>>> debug = node.debug()
>>> debug
Debug(access_guardian=False, accounting=False, anti_malware=False,
↪ authentication=False,
    blacklisting=False, browser_based_user_authentication=False, cluster_
↪ daemon=False,
    cluster_protocol=False, connection_tracking=False, data_synchronization=False,
    dhcp_client=False, dhcp_relay=False, dhcp_service=False, dns_resolution=False,
    dynamic_routing=False, endpoint_integration=False, file_reputation=False,
    inspection=False, invalid=False, ipsec_vpn=False, licensing=False,
    load_balancing_filter=False, log_server=False, logging_system=False,
↪ management=False,
    mcafee_logon_collector=False, monitoring=False, multicast_routing=False,
↪ nat=False,
    netlink_incoming_ha=False, packet_filtering=False, protocol_agent=False,
    radius_forwarder=False, sandbox=False, server_pool_load_balancing=False,
↪ snmp=False,
    ssl_vpn=False, ssl_vpn_portal=False, ssl_vpn_session_manager=False,
    state_synchronisation=False, syslog=False, system_utilities=False, tester=False,
    user_agent=False, wireless_access_point=False)
>>> debug.management=True
>>> debug.sandbox=True
>>> node.set_debug(debug)
```

### 14.6.7 Pending Changes

```
class smc.core.resource.ChangeRecord(approved_on=None, changed_on=None, element=None,
                                     element_name=None, event_type=None, modifier=None,
                                     approver=None)
```

Bases: [ChangeRecord](#)

Change record details for any pending changes.

#### Parameters

- **approved\_on** – approved on datetime, may be empty if not approved
- **change\_on** – changed on datetime
- **element** – element affected
- **event\_type** – type of change, update, delete, etc.
- **modifier** – account making the modification
- **element\_name** – name of the element (only present in SMC >= 6.5)
- **approver** – name of who has done the change.

```
class smc.core.resource.PendingChanges(engine)
```

Bases: [SerializedIterable](#)

Pending changes apply to the engine having changes that have not yet been committed. Retrieve from the engine level:

```
>>> for changes in engine.pending_changes.all():
...     print(changes, changes.resolve_element)
...
(ChangeRecord(approved_on=u'', changed_on=u'2017-07-12 15:24:40 (GMT)',
element=u'http://172.18.1.150:8082/6.2/elements/fw_cluster/116',
event_type=u'stonegate.object.update', modifier=u'admin'),
FirewallCluster(name=sg_vm))
```

Approve all changes:

```
>>> engine.pending_changes.approve_all()
```

Conversely, reject all pending changes:

```
>>> engine.pending_changes.disapprove_all()
```

#### Raises

*ActionCommandFailed* – failure to retrieve pending changes

#### Return type

*ChangeRecord*

#### approve\_all()

Approve all pending changes

#### Raises

*ActionCommandFailed* – possible permissions issue

#### Returns

None

#### disapprove\_all()

Disapprove all pending changes

#### Raises

*ActionCommandFailed* – possible permissions issue

#### Returns

None

## 14.6.8 Routing

Route module encapsulates functions related to static routing and related configurations on NGFW. When retrieving routing, it is done from the engine context.

For example, retrieve all routing for an engine in context:

```
>>> engine = Engine('sg_vm')
>>> for route_node in engine.routing:
...     print(route_node)
...
Routing(name=Interface 0,level=interface)
Routing(name=Interface 1,level=interface)
Routing(name=Interface 2,level=interface)
```

(continues on next page)

(continued from previous page)

```
Routing(name=Tunnel Interface 2000,level=interface)
Routing(name=Tunnel Interface 2001,level=interface)
```

Routing nodes are nested, starting with the engine level. Routing node nesting is made up of ‘levels’ and can be represented as a tree:

```
engine (root)
|
--> interface
    |
    --> network
        |
        --> gateway
            |
            --> any
```

You can get a representation of the routing or antispoofing tree nodes by calling `as_tree`:

```
>>> print(engine.routing.as_tree())
Routing(name=myfw,level=engine_cluster)
--Routing(name=Interface 0,level=interface)
----Routing(name=network-1.1.1.0/24,level=network)
-----Routing(name=mypeering,level=gateway)
-----Routing(name=mynetlink,level=gateway)
-----Routing(name=router-1.1.1.1,level=any)
-----Routing(name=mystatic,level=gateway)
--Routing(name=Interface 1,level=interface)
----Routing(name=network-10.10.10.0/24,level=network)
-----Routing(name=anotherpeering,level=gateway)
--Routing(name=Tunnel Interface 1000,level=interface)
----Routing(name=network-2.2.2.0/24,level=network)
--Routing(name=Tunnel Interface 1001,level=interface)
--Routing(name=Interface 2,level=interface)
----Routing(name=Network (IPv4),level=network)
-----Routing(name=dynamic_netlink-myfw-Interface 2,level=gateway)
-----Routing(name=Any network,level=any)
```

If nested routes exist, you can iterate a given node to get specific information:

```
>>> interface = engine.routing.get(1)
>>> for routes in interface:
...     print(routes)
...
Routing(name=network-10.0.0.0/24,level=network)
...
>>> for networks in interface:
...     networks
...     for gateways in networks:
...         print gateways, gateways.ip
...
Routing(name=network-172.18.1.0/24,level=network)
Routing(name=asus-wireless,level=gateway) 172.18.1.200
```

If BGP, OSPF or a Traffic Handler (netlink) need to be added to an interface that has multiple IP addresses assigned and you want to bind to only one, you can provide the `network` parameter to `add_` methods. The network can be obtained for an interface:

```
>>> engine = Engine('sg_vm')
>>> interface0 = engine.routing.get(0)
>>> for network in interface0:
...     network, network.ip
...
(Routing(name=network-172.18.1.0/24, level=network), '172.18.1.0/24')
```

Then add using:

```
>>> engine = Engine('sg_vm')
>>> interface0 = engine.routing.get(0)
>>> interface0.add_traffic_handler(StaticNetlink('foo'), network='172.18.1.0/24')
```

**Note:** If the `network` keyword is omitted and the interface has multiple IP addresses assigned, this will bind OSPF, BGP or the Traffic Handler to all address assigned.

Adding a basic static route can be done from the engine directly if it is a simple source network to destination route:

```
engine.add_route(gateway='192.168.1.254/32', network='172.18.1.0/24')
```

The route gateway will be mapped to an interface with an address range in the 192.168.1.x network automatically.

For more complex static routes such as ones that may use group elements, use the routing node:

```
>>> engine = Engine('ve-1')
>>> interface0 = engine.routing.get(0)
>>> interface0.add_static_route(Router('tmprouter'), destination=[Group('routegroup')])
```

When a routing gateway is added to an IPv6 network, the gateway is validated before adding. For example, if you have a single interface that has both an IPv4 and IPv6 address assigned, a static route using a Router gateway with only an IPv4 address will only bind to the IPv4 network. In this case, you can optionally add both an IPv4 and IPv6 to the router element, or run this operation for each network respectively.

**See also:**

[`Routing.add\_static\_route\(\)`](#)

**Note:** When changing are made to a routing node, i.e. adding OSPF, BGP, Netlink's, the configuration is updated immediately without calling `.update()`

**class** `smc.core.route.RoutingTree`(*data=None, \*\*meta*)

`RoutingTree` is the base class for both `Routing` and `Antispoofing` nodes. This provides a common API for operations that affect how routing table and antispoofing operate.

**all()**

Return all routes for this engine.

**Returns**

current route entries as [`Routing`](#) element

**Return type**`list`**as\_tree**(*level=0*)

Display the routing tree representation in string format

**Return type**`str`**delete**()

Delete the element

**Raises***DeleteElementFailed* – possible dependencies, record locked, etc**Returns**

None

**property** `dynamic_nicid`

NIC id for this dynamic interface

**Returns**

nic identifier, if this is a DHCP interface

**Return type**`str` or None**get**(*interface\_id*)

Obtain routing configuration for a specific interface by ID.

---

**Note:** If interface is a VLAN, you must use a str to specify the interface id, such as '3.13' (interface 3, VLAN 13)

---

**Parameters****interface\_id** (*str*, *int*) – interface identifier**Raises***InterfaceNotFound* – invalid interface for engine**Returns**

Routing element, or None if not found

**Return type***Routing***property** `ip`

IP network / host for this route

**Returns**

IP address of this routing level

**Return type**`str`**property** `level`

Routing nodes have multiple 'levels' where routes can be nested. Most routes are placed at the interface level. This setting can mostly be ignored, but provides an informative view of how the route is nested.

**Returns**

routing node level (interface,network,gateway,any)

**Return type**

str

**property name**

Interface name / ID for routing level

**Returns**

name of routing node

**Return type**

str

**property nicid**

NIC id for this interface

**Returns**

nic identifier

**Return type**

str

**property probe\_ecmp**

the ECMP for multi path routing.

**Return type**

int

**property probe\_interval**

the probe interval.

**Return type**

int

**property probe\_ipaddress**

the probe ipaddress, when the probe test is Ping.

**Return type**

str

**property probe\_metric**

the probe metric.

**Return type**

int

**property probe\_retry\_count**

the probe retry counter.

**Return type**

int

**property probe\_test**

probe test for a route. possible values: ping, next\_hop\_reachability, not\_enabled

**Returns**

probe test value

**Return type**

str

**property related\_element\_type**

New in version 0.6.0: Requires SMC version  $\geq 6.4$

Related element type defines the ‘type’ of element at this routing or antispoofing node level.

**Return type**

`str`

**update()**

Update the existing element and clear the instance cache. Removing the cache will ensure subsequent calls requiring element attributes will force a new fetch to obtain the latest copy.

Calling `update()` with no args will assume the element has already been modified directly and the *data* cache will be used to update. You can also override the following attributes: `href`, `etag`, `json` and `params`. If `json` is sent, it is expected to be a complete payload to satisfy the update.

For kwargs, if attribute values are a list, you can pass ‘`append_lists=True`’ to add to an existing list, otherwise overwrite the existing (default: overwrite)

**See also:**

To see different ways to utilize this method for updating, see: [Update](#).

**Parameters**

- **exception** – pass a custom exception to throw if failure
- **kwargs** – optional kwargs to update request data to server.

**Raises**

- [ModificationFailed](#) – raised if element is tagged as System element
- [UpdateElementFailed](#) – failed to update element with reason

**Returns**

href of the element modified

**Return type**

`str`

### 14.6.8.1 Routing

**class** `smc.core.route.Routing`(*data=None, \*\*meta*)

Bases: [RoutingTree](#)

Routing represents the Engine routing configuration and provides the ability to view and add features to routing nodes such as OSPF.

**add\_bgp\_peering**(*bgp\_peering, external\_bgp\_peer=None, network=None*)

Add a BGP configuration to this routing interface. If the interface has multiple ip addresses, all networks will receive the BGP peering by default unless the `network` parameter is specified.

Example of adding BGP to an interface by ID:

```
interface = engine.routing.get(0)
interface.add_bgp_peering(
    BGPPeering('mypeer'),
    ExternalBGPPeer('neighbor'))
```

**Parameters**

- **bgp\_peering** ([BGPPeering](#)) – BGP Peer element
- **external\_bgp\_peer** ([ExternalBGPPeer](#), [Engine](#)) – peer element or href
- **network** (*str*) – if network specified, only add OSPF to this network on interface

**Raises**

- [ModificationAborted](#) – Change must be made at the interface level
- [UpdateElementFailed](#) – failed to add BGP

**Returns**

Status of whether the route table was updated

**Return type**

[bool](#)

**add\_dynamic\_gateway**(*networks*)

A dynamic gateway object creates a router object that is attached to a DHCP interface. You can associate networks with this gateway address to identify networks for routing on this interface.

```
route = engine.routing.get(0)
route.add_dynamic_gateway([Network('mynetwork')])
```

**Parameters**

**Network** (*list*) – list of network elements to add to this DHCP gateway

**Raises**

- [ModificationAborted](#) – Change must be made at the interface level
- [UpdateElementFailed](#) – failure to update routing table

**Returns**

Status of whether the route table was updated

**Return type**

[bool](#)

**add\_ospf\_area**(*ospf\_area*, *ospf\_interface\_setting=None*, *network=None*, *communication\_mode='not\_forced'*, *unicast\_ref=None*)

Add OSPF Area to this routing node.

Communication mode specifies how the interface will interact with the adjacent OSPF environment. Please see SMC API documentation for more in depth information on each option.

If the interface has multiple networks nested below, all networks will receive the OSPF area by default unless the **network** parameter is specified. OSPF cannot be applied to IPv6 networks.

Example of adding an area to interface routing node:

```
area = OSPFArea('area0') #obtain area resource

#Set on routing interface 0
interface = engine.routing.get(0)
interface.add_ospf_area(area)
```

---

**Note:** If unicast is specified, you must also provide a `unicast_ref` of element type `Host` to identify the remote host. If no `unicast_ref` is provided, this is skipped

---

#### Parameters

- **ospf\_area** (`OSPFArea`) – OSPF area instance or href
- **ospf\_interface\_setting** (`OSPFInterfaceSetting`) – used to override the OSPF settings for this interface (optional)
- **network** (`str`) – if network specified, only add OSPF to this network on interface
- **communication\_mode** (`str`) – `not_forced|point_to_point|passive|unicast`
- **unicast\_ref** (`Element`) – Element used as unicast gw (required for unicast)

#### Raises

- **ModificationAborted** – Change must be made at the interface level
- **UpdateElementFailed** – failure updating routing
- **ElementNotFound** – ospf area not found

#### Returns

Status of whether the route table was updated

#### Return type

`bool`

#### `add_static_route(gateway, destination, network=None)`

Add a static route to this route table. Destination can be any element type supported in the routing table such as a Group of network members. Since a static route gateway needs to be on the same network as the interface, provide a value for *network* if an interface has multiple addresses on different networks.

```
>>> engine = Engine('ve-1')
>>> itf = engine.routing.get(0)
>>> itf.add_static_route(
    gateway=Router('tmprouter'),
    destination=[Group('routegroup')])
```

#### Parameters

- **gateway** (`Element`) – gateway for this route (Router, Host)
- **destination** (`list(Host, Router, ...)`) – destination network/s for this route.

#### Raises

- **ModificationAborted** – Change must be made at the interface level
- **UpdateElementFailed** – failure to update routing table

#### Returns

Status of whether the route table was updated

#### Return type

`bool`

**add\_traffic\_handler**(*netlink*, *netlink\_gw=None*, *network=None*)

Add a traffic handler to a routing node. A traffic handler can be either a static netlink or a multilink traffic handler. If *network* is not specified and the interface has multiple IP addresses, the traffic handler will be added to all ipv4 addresses.

Add a pre-defined netlink to the route table of interface 0:

```
engine = Engine('vm')
rnode = engine.routing.get(0)
rnode.add_traffic_handler(StaticNetlink('mynetlink'))
```

Add a pre-defined netlink only to a specific network on an interface with multiple addresses. Specify a *netlink\_gw* for the netlink:

```
rnode = engine.routing.get(0)
rnode.add_traffic_handler(
    StaticNetlink('mynetlink'),
    netlink_gw=[Router('myrtr'), Host('myhost')],
    network='172.18.1.0/24')
```

#### Parameters

- **netlink** (*StaticNetlink*, *Multilink*) – netlink element
- **netlink\_gw** (*list*(*Element*)) – list of elements that should be destinations for this netlink. Typically these may be of type host, router, group, server, network or engine.
- **network** (*str*) – if network specified, only add OSPF to this network on interface

#### Raises

- *UpdateElementFailed* – failure updating routing
- *ModificationAborted* – Change must be made at the interface level
- *ElementNotFound* – ospf area not found

#### Returns

Status of whether the route table was updated

#### Return type

*bool*

#### property **bgp\_peerings**

BGP Peerings applied to a routing node. This can be called from the engine, interface or network level. Return is a tuple of (interface, network, bgp\_peering). This simplifies viewing and removing BGP Peers from the routing table:

```
>>> for bgp in engine.routing.bgp_peerings:
...     bgp
...
(Routing(name=Interface 0,level=interface,type=physical_interface),
 Routing(name=network-1.1.1.0/24,level=network,type=network),
 Routing(name=mypeering,level=gateway,type=bgp_peering))
(Routing(name=Interface 1,level=interface,type=physical_interface),
 Routing(name=network-2.2.2.0/24,level=network,type=network),
 Routing(name=mypeering,level=gateway,type=bgp_peering))
```

See also:

[`netlinks\(\)`](#) and [`ospf\_areas\(\)`](#) for obtaining other routing element types

**Return type**

`tuple(Routing)`

#### property `netlinks`

Netlinks applied to a routing node. This can be called from the engine, interface or network level. Return is a tuple of (interface, network, netlink). This simplifies viewing and removing Netlinks from the routing table:

```
>>> interface = engine.routing.get(1)
>>> for static_netlink in interface.netlinks:
...     interface, network, netlink = static_netlink
...     netlink
...     netlink.delete()
...
Routing(name=mylink, level=gateway, type=netlink)
```

See also:

[`bgp\_peerings\(\)`](#) and [`ospf\_areas\(\)`](#) for obtaining other routing element types

**Return type**

`tuple(Routing)`

#### property `ospf_areas`

OSPFv2 areas applied to a routing node. This can be called from the engine, interface or network level. Return is a tuple of (interface, network, bgp\_peering). This simplifies viewing and removing BGP Peers from the routing table:

```
>>> for ospf in engine.routing.ospf_areas:
...     ospf
...
(Routing(name=Interface 0, level=interface, type=physical_interface),
 Routing(name=network-1.1.1.0/24, level=network, type=network),
 Routing(name=area10, level=gateway, type=ospfv2_area))
```

See also:

[`bgp\_peerings\(\)`](#) and [`netlinks\(\)`](#) for obtaining other routing element types

**Return type**

`tuple(Routing)`

#### `remove_route_gateway(element, network=None)`

Remove a route element by href or Element. Use this if you want to remove a netlink or a routing element such as BGP or OSPF. Removing is done from within the routing interface context.

```
interface0 = engine.routing.get(0)
interface0.remove_route_gateway(StaticNetlink('mynetlink'))
```

Only from a specific network on a multi-address interface:

```
interface0.remove_route_gateway(
    StaticNetlink('mynetlink'),
    network='172.18.1.0/24')
```

**Parameters**

- **element** (*str*, *Element*) – element to remove from this routing node
- **network** (*str*) – if network specified, only add OSPF to this network on interface

**Raises**

- *ModificationAborted* – Change must be made at the interface level
- *UpdateElementFailed* – failure to update routing table

**Returns**

Status of whether the entry was removed (i.e. or not found)

**Return type**

*bool*

**property routing\_node\_element**

A routing node element will reference the element used to represent the node (i.e. router, host, network, netlink, bgp peering, etc). Although the routing node already resolves the element and provides the *ip* property to obtain the address/network, use this property to obtain access to modifying the element itself:

```
>>> interface0 = engine.routing.get(0)
>>> for networks in interface0:
...     for gateway in networks:
...         gateway.routing_node_element
...
Router(name=router-1.1.1.1)
StaticNetlink(name=mystatic)
BGPPeering(name=anotherpeering)
BGPPeering(name=mypeering)
>>>
```

**14.6.8.2 Antispoofing**

**class** `smc.core.route.Antispoofing`(*data=None*, *\*\*meta*)

Bases: *RoutingTree*

Anti-spoofing is configured by default based on interface networks directly attached. It is possible to override these settings by adding additional networks as valid source networks on a given interface.

Antispoofing is nested similar to routes. Iterate the antispoofing configuration:

```
for entry in engine.antispoofing.all():
    print(entry)
```

**add**(*element*)

Add an entry to this antispoofing node level. Entry can be either href or network elements specified in *smc.elements.network*

```
if0 = engine.antispoofing.get(0)
if0.add(Network('foonet'))
```

**Parameters**

**element** ([Element](#)) – entry to add, i.e. Network('mynetwork'), Host(..)

**Raises**

- [CreateElementFailed](#) – failed adding entry
- [ElementNotFound](#) – element entry specified not in SMC

**Returns**

whether entry was added

**Return type**

[bool](#)

**property autogenerated**

Was the entry auto generated by a route entry or added manually as an override

**Return type**

[bool](#)

**remove(element)**

Remove a specific user added element from the antispoofing tables of a given interface. This will not remove autogenerated or system level entries.

**Parameters**

**element** ([Element](#)) – element to remove

**Returns**

remove element if it exists and return bool

**Return type**

[bool](#)

**property validity**

Enabled or disabled antispoofing entry

**Returns**

validity of this entry (enable,disable,absolute)

**Return type**

[str](#)

### 14.6.8.3 Route Table

```
class smc.core.route.Route(data)
```

Active routes obtained from a running engine. Obtain routes from an engine reference:

```
>>> engine = Engine('sg_vm')
>>> for route in engine.routing_monitoring:
...     route
```

**Variables**

- **route\_network** ([str](#)) – network for this route

- **route\_netmask** (*int*) – netmask for the route
- **route\_gateway** (*str*) – route gateway, may be None if it's a local network only
- **route\_type** (*str*) – status of the route
- **dst\_if** (*int*) – destination interface index
- **src\_if** (*int*) – source interface index

#### 14.6.8.4 Policy Routing

**class** `smc.core.route.PolicyRoute(engine)`

An iterable providing an interface to policy based routing on the engine. You must call `engine.udpate()` after performing an add or delete:

```
>>> engine = Engine('myfw')
>>> engine.policy_route
PolicyRoute(items: 1)
>>> for rt in engine.policy_route:
...     rt
...
PolicyRoute(source=u'172.18.1.0/24', destination=u'172.18.1.0/24',
            gateway_ip=u'172.18.1.1', comment=None)
>>> engine.policy_route.create(source='172.18.2.0/24',
                               destination='192.168.3.0/24', gateway_ip='172.18.2.1
→')
>>> engine.update()
'http://172.18.1.151:8082/6.4/elements/single_fw/746'
>>> for rt in engine.policy_route:
...     rt
...
PolicyRoute(source=u'172.18.1.0/24', destination=u'172.18.1.0/24',
            gateway_ip=u'172.18.1.1', comment=None)
PolicyRoute(source=u'172.18.2.0/24', destination=u'192.168.3.0/24',
            gateway_ip=u'172.18.2.1', comment=None)
>>> engine.policy_route.delete(source='172.18.2.0/24')
>>> engine.update()
'http://172.18.1.151:8082/6.4/elements/single_fw/746'
>>> for rt in engine.policy_route:
...     rt
...
PolicyRoute(source=u'172.18.1.0/24', destination=u'172.18.1.0/24',
            gateway_ip=u'172.18.1.1', comment=None)
```

#### Variables

- **source** (*str*) – source network/cidr for the route
- **destination** (*str*) – destination network/cidr for the route
- **gateway\_ip** (*str*) – gateway IP address, must be on source network
- **comment** (*str*) – optional comment

**create**(*source, destination, gateway\_ip, comment=None*)

Add a new policy route to the engine.

**Parameters**

- **source** (*str*) – network address with /cidr
- **destination** (*str*) – network address with /cidr
- **gateway** (*str*) – IP address, must be on source network
- **comment** (*str*) – optional comment

**delete**(*\*\*kw*)

Delete a policy route from the engine. You can delete using a single field or multiple fields for a more exact match. Use a keyword argument to delete a route by any valid attribute.

**Parameters**

**kw** – use valid Route keyword values to delete by exact match

#### 14.6.8.5 TunnelMonitoringGroup

### 14.6.9 Snapshot

**class** `smc.core.resource.Snapshot`(*\*\*meta*)

Bases: `SubElement`

Policy snapshots currently held on the SMC. You can retrieve all snapshots at the engine level and view details of each:

```
for snapshot in engine.snapshots:
    print(snapshot)
```

Snapshots can be generated manually, but also will be generated automatically when a policy is pushed:

```
engine.generate_snapshot(filename='mysnapshot.zip')
```

Snapshots can also be downloaded:

```
for snapshot in engine.snapshots:
    if snapshot.name == 'blah snapshot':
        snapshot.download()
```

Snapshot filename will be <snapshot\_name>.zip if not specified.

**compare\_to\_latest**(*filename=None*)

Download comparison between this snapshot and the current version of the configuration as HTML to filename

**Parameters**

**filename** (*str*) – fully qualified path including filename .zip

**Raises**

`EngineCommandFailed` – IOError occurred downloading snapshot

**Returns**

None

**download**(filename=None)

Download snapshot to filename

**Parameters**

**filename** (*str*) – fully qualified path including filename .zip

**Raises**

*EngineCommandFailed* – IOError occurred downloading snapshot

**Returns**

None

**download\_html**(filename=None)

Download snapshot as HTML to filename

**Parameters**

**filename** (*str*) – fully qualified path including filename .zip

**Raises**

*EngineCommandFailed* – IOError occurred downloading snapshot

**Returns**

None

## 14.6.10 VirtualResource

**class** smc.core.engine.VirtualResource(\*\*meta)

Bases: *SubElement*

A Virtual Resource is a container placeholder for a virtual engine within a Master Engine. When creating a virtual engine, each virtual engine must have a unique virtual resource for mapping. The virtual resource has an identifier (vfw\_id) that specifies the engine ID for that instance.

This is called as a resource of an engine. To view all virtual resources:

```
list(engine.virtual_resource.all())
```

Available attributes:

**Variables**

- **connection\_limit** (*int*) – Maximum number of connections for this virtual engine. 0 means unlimited (default: 0)
- **show\_master\_nic** (*bool*) – Show the master engine NIC id's in the virtual engine.

When updating this element, make modifications and call update()

**property allocated\_domain\_ref**

Domain that this virtual engine is allocated in. 'Shared Domain' is the default if no domain is specified.

```
>>> for resource in engine.virtual_resource:
...     resource, resource.allocated_domain_ref
...
(VirtualResource(name=ve-1), AdminDomain(name=Shared Domain))
(VirtualResource(name=ve-8), AdminDomain(name=Shared Domain))
```

**Returns**

AdminDomain element

**Return type***AdminDomain*

**create**(*name*, *vfw\_id*, *domain*='Shared Domain', *show\_master\_nic*=False, *connection\_limit*=0, *comment*=None)

Create a new virtual resource. Called through engine reference:

```
engine.virtual_resource.create(...)
```

**Parameters**

- **name** (*str*) – name of virtual resource
- **vfw\_id** (*int*) – virtual fw identifier
- **domain** (*str*) – name of domain to install, (default Shared)
- **show\_master\_nic** (*bool*) – whether to show the master engine NIC ID's in the virtual instance
- **connection\_limit** (*int*) – whether to limit number of connections for this instance

**Returns**

href of new virtual resource

**Return type***str*

**set\_admin\_domain**(*admin\_domain*)

Virtual Resources can be members of an Admin Domain to provide delegated administration features. Assign an admin domain to this resource. Admin Domains must already exist.

**Parameters**

**admin\_domain** (*str*, *AdminDomain*) – Admin Domain to add

**Returns**

None

**property vfw\_id**

Read-Only virtual engine identifier. This is unique per virtual engine and is set when the virtual resource is created.

**Returns**

vfw id

**Return type***int*

## 14.6.11 AdvanceSettings

Engine advanced setting functionality such as LogModeration. These are common settings that are located under the SMC Advanced Settings.

**class** `smc.core.advanced_settings.LogModeration`(*engine*)

This is the definition of Log Compression for the engine or for an interface. You can also configure Log Compression to save resources on the engine. By default, each generated Antispoofing and Discard log entry is logged separately and displayed as a separate entry in the Logs view. Log Compression allows you to define the maximum number of separately logged entries. When the defined limit is reached, a single Antispoofing log entry

or Discard log entry is logged. The single entry contains information on the total number of the generated Antispoofing log entries or Discard log entries. After this, logging returns to normal and all the generated entries are once more logged and displayed separately.

**add**(*burst=None, log\_event=None, rate=None*)

Add log\_moderation setting entry on interface or engine. :param str burst: The maximum number of matching entries in a single burst. The default value for Antispoofing entries is 1000. By default, Discard log entries are not compressed. :param int log\_event: Log Moderation Event Type:

1. antispoofing: Antispoofing entry (L3 engine only)
2. discard: discard entry.

#### Parameters

**rate** (*int*) – The maximum number of entries per second. The default value for

Antispoofing entries is 100 entries /s. By default, Discard log entries are not compressed.

**contains**(*log\_event*)

Check if specific log\_moderation settings are present. :param int log\_event: Log Moderation Event Type:

1. antispoofing: Antispoofing entry (L3 engine only)
2. discard: discard entry.

**get**(*log\_event*)

Return log\_moderation entry with specific log\_event :param str log\_event: Log Moderation Event Type:

1. antispoofing: Antispoofing entry (L3 engine only)
2. discard: discard entry.

:rtype dict

**remove**(*log\_event*)

Remove the specific log\_moderation setting. :param str log\_event: Log Moderation Event Type:

1. antispoofing: Antispoofing entry (L3 engine only)
2. discard: discard entry.

## 14.7 Engine Types

### 14.7.1 IPS

**class** smc.core.engines.IPS(*name=None, \*\*meta*)

Creates an IPS engine with a default inline interface pair

**classmethod** create(*name, mgmt\_ip, mgmt\_network, mgmt\_interface=0, inline\_interface='1-2', logical\_interface='default\_eth', log\_server\_ref=None, domain\_server\_address=None, zone\_ref=None, enable\_antivirus=False, enable\_gti=False, comment=None, extra\_opts=None, lldp\_profile=None, discard\_quic\_if\_cant\_inspect=True, node\_definition=None, \*\*kw*)

Create a single IPS engine with management interface and inline pair

#### Parameters

- **name** (*str*) – name of ips engine

- **mgmt\_ip** (*str*) – ip address of management interface
- **mgmt\_network** (*str*) – management network in cidr format
- **mgmt\_interface** (*int*) – (optional) interface for management from SMC to engine
- **inline\_interface** (*str*) – interfaces to use for first inline pair
- **logical\_interface** (*str*) – name, str href or LogicalInterface (created if it doesn't exist)
- **log\_server\_ref** (*str*) – (optional) href to log\_server instance
- **domain\_server\_address** (*list*) – (optional) DNS server addresses
- **zone\_ref** (*str*) – zone name, str href or Zone for management interface (created if not found)
- **enable\_antivirus** (*bool*) – (optional) Enable antivirus (required DNS) :param bool enable\_gti: (optional) Enable GTI
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for configuring LLDP :param bool discard\_quic\_if\_cant\_inspect: (optional) discard or allow QUIC if inspection is not possible

#### Parameters

**extra\_opts** (*dict*) – extra options as a dict to be passed to the top level engine

:param node\_definition information for the node itself :raises CreateEngineFailed: Failure to create with reason :return: [smc.core.engine.Engine](#)

## 14.7.2 Layer3Firewall

**class** `smc.core.engines.Layer3Firewall`(*name=None, \*\*meta*)

Changed in version 0.7.0: extra\_opts can be passed to the top level engine dict to customize input

Represents a Layer 3 Firewall configuration. A layer 3 single engine is a standalone engine instance (not a cluster). You can use the *create* constructor and add interfaces after the engine exists, or use *create\_bulk* to fully create the engine and interfaces in a single operation.

You can also pass arbitrary kwargs passed in to the engine dict by providing the *extra\_opts* value as a dict. Therefore it can support any custom configurations as long as the format is valid. For example, enabling file reputation on a SMC >= 6.6:

```
extra_opts= {'file_reputation_settings':{'file_reputation_context': 'gti_cloud_only'
→ '}}
```

```
classmethod create(name, mgmt_ip, mgmt_network, mgmt_interface=0, log_server_ref=None,
                    default_nat=False, reverse_connection=False, domain_server_address=None,
                    zone_ref=None, enable_antivirus=False, enable_gti=False, location_ref=None,
                    enable_ospf=False, sidewinder_proxy_enabled=False, known_host_lists=[],
                    ospf_profile=None, snmp=None, ntp_settings=None, timezone=None,
                    comment=None, extra_opts=None, engine_type=None, node_type='firewall_node',
                    lldp_profile=None, link_usage_profile=None, quic_enabled=True,
                    discard_quic_if_cant_inspect=True, node_definition=None, **kw)
```

Create a single layer 3 firewall with management interface and DNS. Provide the *interfaces* keyword argument if adding multiple additional interfaces. Interfaces can be one of any valid interface for a layer 3 firewall. Unless the interface type is specified, physical\_interface is assumed.

If providing the *interfaces* keyword during creation, the valid interface types are:

- `physical_interface` (default if not specified)
- `tunnel_interface`
- `switch_physical_interface`

If providing all engine interfaces in a single operation, see `create_bulk()` for additional examples.

#### Parameters

- **name** (*str*) – name of firewall engine
- **mgmt\_ip** (*str*) – ip address of management interface
- **mgmt\_network** (*str*) – management network in cidr format
- **log\_server\_ref** (*str*) – (optional) href to log\_server instance for engine
- **mgmt\_interface** (*int*) – (optional) interface for management from SMC to engine
- **domain\_server\_address** (*list*) – (optional) DNS server addresses
- **zone\_ref** (*str*) – zone name, str href or zone name for management interface (created if not found)
- **reverse\_connection** (*bool*) – should the NGFW be the mgmt initiator (used when behind NAT)
- **default\_nat** (*bool*) – (optional) Whether to enable default NAT for outbound
- **enable\_antivirus** (*bool*) – (optional) Enable antivirus (required DNS)
- **enable\_gti** (*bool*) – (optional) Enable GTI
- **sidewinder\_proxy\_enabled** (*bool*) – Enable Sidewinder proxy functionality
- **known\_host\_lists** (*list*) – hrefs of ssh known host list objects (comma separated)
- **location\_ref** (*str*) – location href or not for engine if needed to contact SMC behind NAT (created if not found)
- **enable\_ospf** (*bool*) – whether to turn OSPF on within engine
- **ospf\_profile** (*str*) – optional OSPF profile to use on engine, by ref
- **ntp\_settings** (*NTPSettings*) – NTP settings
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for

configuring LLDP :param LinkUsageProfile link\_usage\_profile :param dict extra\_opts: extra options as a dict to be passed to the top level engine :param kw: optional keyword arguments specifying additional interfaces :param bool quic\_enabled: (optional) include QUIC ports for web traffic :param bool discard\_quic\_if\_cant\_inspect: (optional) discard or allow QUIC :param node\_definition information for the node itself

if inspection is not possible

#### Raises

**CreateEngineFailed** – Failure to create with reason

#### Returns

`smc.core.engine.Engine`

```
classmethod create_bulk(name, interfaces=None, primary_mgt=None, backup_mgt=None,
                        auth_request=None, log_server_ref=None, domain_server_address=None,
                        nodes=1, nodes_definition=None, node_type='firewall_node',
                        location_ref=None, default_nat=False, enable_antivirus=False,
                        enable_gti=False, sidewinder_proxy_enabled=False, known_host_lists=[],
                        enable_ospf=False, ospf_profile=None, comment=None, snmp=None,
                        ntp_settings=None, timezone=None, extra_opts=None, engine_type=None,
                        lldp_profile=None, link_usage_profile=None, quic_enabled=True,
                        discard_quic_if_cant_inspect=True, **kw)
```

Create a Layer 3 Firewall providing all of the interface configuration. This method provides a way to fully create the engine and all interfaces at once versus using `create()` and creating each individual interface after the engine exists.

Example interfaces format:

```
interfaces=[
    {'interface_id': 1},
    {'interface_id': 2,
     'interfaces': [{'nodes': [{'address': '2.2.2.2', 'network_value': '2.2.2.0/
↪24'}]}]},
    {'zone_ref': 'myzone'},
    {'interface_id': 3,
     'interfaces': [{'nodes': [{'address': '3.3.3.3', 'network_value': '3.3.3.0/
↪24'}]},
                    {'vlan_id': 3,
                     'zone_ref': 'myzone'},
                    {'nodes': [{'address': '4.4.4.4', 'network_value': '4.4.4.
↪0/24'}]}]},
    {'vlan_id': 4}}],
    {'interface_id': 4,
     'interfaces': [{'vlan_id': 4,
                     'zone_ref': 'myzone'}]}],
    {'interface_id': 5,
     'interfaces': [{'vlan_id': 5}]},
    {'interface_id': 1000,
     'interfaces': [{'nodes': [{'address': '10.10.10.1',
                               'network_value': '10.10.10.0/24'}]}]},
    {'type': 'tunnel_interface'}
```

Sample of creating a simple two interface firewall:

```
firewall_def = {
    'name': 'firewall',
    'domain_server_address': ['192.168.122.1'],
    'primary_mgt': 0,
    'interfaces': [
        {'interface_id': 0,
         'interfaces': [{'nodes': [{'address': '192.168.122.100',
                               'network_value': '192.168.122.0/24', 'auth_request': ↪
↪False}]}]}
    ],
    {'interface_id': 1,
     'interfaces': [{'nodes': [{'address': '10.0.0.254',
```

(continues on next page)

(continued from previous page)

```

        'network_value': '10.0.0.0/24', 'auth_request': True}}]
    }
]
}
fw = Layer3Firewall.create_bulk(**firewall_def)

```

**Note:** You can set `primary_mgt`, `backup_mgt`, `outgoing`, and `auth_request` within the interface definition itself to specify interface options. If provided in the constructor, this will be passed to the interface creation factory. You should use one or the other method, not both.

See `smc.core.interfaces.Layer3PhysicalInterface` for more advanced examples

```

classmethod create_dynamic(name, interface_id, dynamic_index=1, reverse_connection=True,
                           automatic_default_route=True, domain_server_address=None,
                           loopback_ndi='127.0.0.1', location_ref=None, log_server_ref=None,
                           zone_ref=None, enable_gti=False, enable_antivirus=False,
                           sidewinder_proxy_enabled=False, known_host_lists=[],
                           default_nat=False, comment=None, extra_opts=None, engine_type=None,
                           node_type='firewall_node', lldp_profile=None, link_usage_profile=None,
                           quic_enabled=True, discard_quic_if_cant_inspect=True,
                           node_definition=None, **kw)

```

Create a single layer 3 firewall with only a single DHCP interface. Useful when creating virtualized engine's such as in Microsoft Azure.

#### Parameters

- **name** (*str*) – name of engine
- **interface\_id** (*str*, *int*) – interface ID used for dynamic interface and management
- **reverse\_connection** (*bool*) – specifies the dynamic interface should initiate connections to management (default: True)
- **automatic\_default\_route** (*bool*) – allow SMC to create a dynamic netlink for the default route (default: True)
- **domain\_server\_address** (*list*) – list of IP addresses for engine DNS
- **loopback\_ndi** (*str*) – IP address for a loopback NDI. When creating a dynamic engine, the `auth_request` must be set to a different interface, so loopback is created
- **location\_ref** (*str*) – location by name for the engine
- **log\_server\_ref** (*str*) – log server reference, will use the default or first retrieved if not specified
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for

configuring LLDP :param dict extra\_opts: extra options as a dict to be passed to the top level engine :param bool quic\_enabled: (optional) include QUIC ports for web traffic :param bool discard\_quic\_if\_cant\_inspect: (optional) discard or allow QUIC :param node\_definition information for the node itself

if inspection is not possible

**Raises**

*CreateElementFailed* – failed to create engine

**Returns**

*smc.core.engine.Engine*

**property ha\_settings**

HA settings for the engine

**Return type**

*HAForSingleEngine* or None

**property quic\_enabled**

include QUIC ports for web traffic

**Return type**

bool

### 14.7.3 Layer2Firewall

**class** `smc.core.engines.Layer2Firewall`(*name=None, \*\*meta*)

Creates a Layer 2 Firewall with a default inline interface pair To instantiate and create, call ‘create’ classmethod as follows:

```
engine = Layer2Firewall.create(name='myinline',
                               mgmt_ip='1.1.1.1',
                               mgmt_network='1.1.1.0/24')
```

**classmethod** `create`(*name, mgmt\_ip, mgmt\_network, mgmt\_interface=0, inline\_interface='1-2', logical\_interface='default\_eth', log\_server\_ref=None, domain\_server\_address=None, zone\_ref=None, enable\_antivirus=False, enable\_gti=False, comment=None, extra\_opts=None, lldp\_profile=None, discard\_quic\_if\_cant\_inspect=True, node\_definition=None, \*\*kw*)

Create a single layer 2 firewall with management interface and inline pair

**Parameters**

- **name** (*str*) – name of firewall engine
- **mgmt\_ip** (*str*) – ip address of management interface
- **mgmt\_network** (*str*) – management network in cidr format
- **mgmt\_interface** (*int*) – (optional) interface for management from SMC to engine
- **inline\_interface** (*str*) – interfaces to use for first inline pair
- **logical\_interface** (*str*) – name, str href or LogicalInterface (created if it doesn’t exist)
- **log\_server\_ref** (*str*) – (optional) href to log\_server instance
- **domain\_server\_address** (*list*) – (optional) DNS server addresses
- **zone\_ref** (*str*) – zone name, str href or Zone for management interface (created if not found)
- **enable\_antivirus** (*bool*) – (optional) Enable antivirus (required DNS) :param bool enable\_gti: (optional) Enable GTI
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for

configuring LLDP :param bool discard\_quic\_if\_cant\_inspect: (optional) discard or allow QUIC

if inspection is not possible

:param node\_definition information for the node itself :param dict extra\_opts: extra options as a dict to be passed to the top level engine :raises CreateEngineFailed: Failure to create with reason :return: `smc.core.engine.Engine`

### 14.7.4 Layer3VirtualEngine

**class** `smc.core.engines.Layer3VirtualEngine(name=None, **meta)`

Create a layer3 virtual engine and map to specified Master Engine Each layer 3 virtual firewall will use the same virtual resource that should be pre-created.

To instantiate and create, call 'create' as follows:

```
engine = Layer3VirtualEngine.create(
    name='myips',
    master_engine='mymaster_engine',
    virtual_engine='ve-3',
    interfaces=[{'interface_id': 0,
                  'address': '5.5.5.5',
                  'network_value': '5.5.5.5/30',
                  'zone_ref': ''}]
```

**classmethod** `create(name, master_engine, virtual_resource, interfaces, default_nat=False, outgoing_intf=0, domain_server_address=None, enable_ospf=False, ospf_profile=None, comment=None, extra_opts=None, quic_enabled=True, discard_quic_if_cant_inspect=True, **kw)`

Create a Layer3Virtual engine for a Master Engine. Provide interfaces as a list of dict items specifying the interface details in format:

```
{'interface_id': 1, 'address': '1.1.1.1', 'network_value': '1.1.1.0/24',
 'zone_ref': zone_by_name, href, 'comment': 'my interface comment'}
```

#### Parameters

- **name** (*str*) – Name of this layer 3 virtual engine
- **master\_engine** (*str*) – Name of existing master engine
- **virtual\_resource** (*str*) – name of pre-created virtual resource
- **interfaces** (*list*) – dict of interface details
- **default\_nat** (*bool*) – Whether to enable default NAT for outbound
- **outgoing\_intf** (*int*) – outgoing interface for VE. Specifies interface number
- **interfaces** – interfaces mappings passed in
- **enable\_ospf** (*bool*) – whether to turn OSPF on within engine
- **ospf\_profile** (*str*) – optional OSPF profile to use on engine, by ref
- **quic\_enabled** (*bool*) – (optional) include QUIC ports for web traffic
- **discard\_quic\_if\_cant\_inspect** (*bool*) – (optional) discard or allow QUIC if inspection is not possible

- **extra\_opts** (*dict*) – extra options as a dict to be passed to the top level engine

**Raises**

- **CreateEngineFailed** – Failure to create with reason
- **LoadEngineFailed** – master engine not found

**Returns**

*smc.core.engine.Engine*

**property quic\_enabled**

include QUIC ports for web traffic

**Return type**

*bool*

## 14.7.5 FirewallCluster

**class** *smc.core.engines.FirewallCluster*(*name=None, \*\*meta*)

Firewall Cluster Creates a layer 3 firewall cluster engine with CVI and NDI's. Once engine is created, you can later add additional interfaces using the *engine.physical\_interface* reference.

**See also:**

*smc.core.physical\_interface.add\_layer3\_cluster\_interface()*

**classmethod create**(*name, cluster\_virtual, network\_value, macaddress, interface\_id, nodes, nodes\_definition=[], vlan\_id=None, cluster\_mode='balancing', backup\_mgt=None, primary\_heartbeat=None, log\_server\_ref=None, domain\_server\_address=None, location\_ref=None, zone\_ref=None, default\_nat=False, enable\_antivirus=False, enable\_gti=False, comment=None, snmp=None, ntp\_settings=None, timezone=None, extra\_opts=None, lldp\_profile=None, link\_usage\_profile=None, quic\_enabled=True, discard\_quic\_if\_cant\_inspect=True, \*\*kw*)

Create a layer 3 firewall cluster with management interface and any number of nodes. If providing keyword arguments to create additional interfaces, use the same constructor arguments and pass an *interfaces* keyword argument. The constructor defined interface will be assigned as the primary management interface by default. Otherwise the engine will be created with a single interface and interfaces can be added after.

Changed in version 0.6.1: Chngned *cluster\_nic* to *interface\_id*, and *cluster\_mask* to *network\_value*

**Parameters**

- **name** (*str*) – name of firewall engine
- **cluster\_virtual** (*str*) – ip of cluster CVI
- **network\_value** (*str*) – ip netmask of cluster CVI
- **macaddress** (*str*) – macaddress for packet dispatch clustering
- **interface\_id** (*str*) – nic id to use for primary interface
- **nodes** (*list*) – address/network\_value/nodeid combination for cluster nodes

:param list nodes\_definition : list of node info (name, comment..) :param str vlan\_id: optional VLAN id for the management interface, i.e. '15'. :param str cluster\_mode: 'balancing' or 'standby' mode (default: balancing) :param str,int primary\_heartbeat: optionally set the primary\_heartbeat. This is

automatically set to the management interface but can be overridden to use another interface if defining additional interfaces using *interfaces*.

### Parameters

- **backup\_mgt** (*str*, *int*) – optionally set the backup management interface. This is unset unless you define additional interfaces using *interfaces*.
- **log\_server\_ref** (*str*) – (optional) href to log\_server instance
- **domain\_server\_address** (*list*) – (optional) DNS server addresses
- **location\_ref** (*str*) – location href or not for engine if needed to contact SMC behind NAT (created if not found)
- **zone\_ref** (*str*) – zone name, str href or Zone for management interface (created if not found)
- **enable\_antivirus** (*bool*) – (optional) Enable antivirus (required DNS)
- **enable\_gti** (*bool*) – (optional) Enable GTI
- **interfaces** (*list*) – optional keyword to supply additional interfaces
- **snmp** (*dict*) – SNMP dict should have keys *snmp\_agent* str defining name of SNMPAgent, *snmp\_interface* which is a list of interface IDs, and optionally *snmp\_location* which is a string with the SNMP location name.
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for

configuring LLDP :param LinkUsageProfile link\_usage\_profile: Link usage profile :param bool quic\_enabled: (optional) include QUIC ports for web traffic :param bool discard\_quic\_if\_cant\_inspect: (optional) discard or allow QUIC

if inspection is not possible

### Parameters

**extra\_opts** (*dict*) – extra options as a dict to be passed to the top level engine

### Raises

**CreateEngineFailed** – Failure to create with reason

### Returns

*smc.core.engine.Engine*

Example nodes parameter input:

```
[{'address': '5.5.5.2', 'network_value': '5.5.5.0/24', 'nodeid': 1},
 {'address': '5.5.5.3', 'network_value': '5.5.5.0/24', 'nodeid': 2},
 {'address': '5.5.5.4', 'network_value': '5.5.5.0/24', 'nodeid': 3}]
```

You can also create additional CVI+NDI, or NDI only interfaces by providing the keyword argument interfaces using the same keyword values from the constructor:

```
interfaces=[
    {'interface_id': 1,
      'macaddress': '02:02:02:02:02:03',
      'interfaces': [{'cluster_virtual': '2.2.2.1',
                      'network_value': '2.2.2.0/24',
                      'nodes': [{'address': '2.2.2.2', 'network_value': '2.2.2.0/24
→',
                                'nodeid': 1},
                          {'address': '2.2.2.3', 'network_value': '2.2.2.0/24
```

(continues on next page)

(continued from previous page)

```

↪',
                                'nodeid': 2}]
                                }],
                                },
                                {'interface_id': 2,
                                'interfaces': [{'nodes': [{'address': '3.3.3.2', 'network_value': '3.3.3.0/24
↪',
                                'nodeid': 1},
                                {'address': '3.3.3.3', 'network_value': '3.3.3.0/24
↪',
                                'nodeid': 2}]
                                }]}
                                ]}]
                                ]}]

```

It is also possible to define VLAN interfaces by providing the *vlan\_id* keyword. Example VLAN with NDI only interfaces. If nesting the *zone\_ref* within the interfaces list, the zone will be applied to the VLAN versus the top level interface:

```

interfaces=[
    {'interface_id': 2,
    'interfaces': [{'nodes': [{'address': '3.3.3.2', 'network_value': '3.3.3.0/24
↪',
                                'nodeid': 1},
                                {'address': '3.3.3.3', 'network_value': '3.3.3.0/24
↪',
                                'nodeid': 2}],
                                'vlan_id': 22,
                                'zone_ref': 'private-network'
                                },
                                {'nodes': [{'address': '4.4.4.1', 'network_value': '4.4.4.0/24
↪',
                                'nodeid': 1},
                                {'address': '4.4.4.2', 'network_value': '4.4.4.0/24
↪',
                                'nodeid': 2}],
                                'vlan_id': 23,
                                'zone_ref': 'other_vlan'
                                }]}
    ]}]

```

Tunnel interfaces can also be created. As all interfaces defined are assumed to be a physical interface type, you must specify the *type* parameter to indicate the interface is a tunnel interface. Tunnel interfaces do not have a macaddress or VLANs. They be configured with NDI interfaces by omitting the *cluster\_virtual* and *network\_value* top level attributes:

```

interfaces=[
    {'interface_id': 1000,
    'interfaces': [{'cluster_virtual': '100.100.100.1',
                                'network_value': '100.100.100.0/24',
                                'nodes': [{'address': '100.100.100.2', 'network_value':
                                '100.100.100.0/24', 'nodeid': 1},
                                {'address': '100.100.100.3', 'network_value':

```

(continues on next page)

(continued from previous page)

```

        '100.100.100.0/24', 'nodeid': 2}]
    }],
    'zone_ref': 'AWStunnel',
    'type': 'tunnel_interface'
}]

```

If setting `primary_heartbeat` or `backup_mgt` to a specific interface (the primary interface configured in the constructor will have these roles by default), you must define the interfaces in the `interfaces` keyword argument list.

**Note:** If creating additional interfaces, you must at minimum provide the `interface_id` and `nodes` to create an NDI only interface.

```

classmethod create_bulk(name, interfaces=None, nodes=2, nodes_definition=[],
                        cluster_mode='balancing', primary_mgt=None, backup_mgt=None,
                        primary_heartbeat=None, log_server_ref=None,
                        domain_server_address=None, location_ref=None, default_nat=False,
                        enable_antivirus=False, enable_gti=False, comment=None, snmp=None,
                        ntp_settings=None, timezone=None, extra_opts=None, lldp_profile=None,
                        link_usage_profile=None, quic_enabled=True,
                        discard_quic_if_cant_inspect=True, **kw)

```

Create bulk is called by the `create` constructor when creating a cluster engine. This allows for multiple interfaces to be defined and passed in during element creation.

#### Parameters

**snmp** (*dict*) – SNMP dict should have keys `snmp_agent` str defining name of SNMPAgent, `snmp_interface` which is a list of interface IDs, and optionally `snmp_location` which is a string with the SNMP location name.

#### property quic\_enabled

include QUIC ports for web traffic

#### Return type

*bool*

## 14.7.6 MasterEngine

```

class smc.core.engines.MasterEngine(name=None, **meta)

```

Creates a master engine in a firewall role. Layer3VirtualEngine should be used to add each individual instance to the Master Engine.

```

classmethod create(name, master_type, mgmt_ip, mgmt_network, mgmt_interface=0,
                    log_server_ref=None, zone_ref=None, domain_server_address=None,
                    enable_gti=False, enable_antivirus=False, comment=None, extra_opts=None,
                    lldp_profile=None, cluster_mode='standby', reverse_connection=False,
                    node_definition=None, **kw)

```

Create a Master Engine with management interface

#### Parameters

- **name** (*str*) – name of master engine engine
- **master\_type** (*str*) – firewall|

- **mgmt\_ip** (*str*) – ip address for management interface
- **mgmt\_network** (*str*) – full netmask for management
- **mgmt\_interface** (*str*) – interface to use for mgmt (default: 0)
- **log\_server\_ref** (*str*) – (optional) href to log\_server instance
- **domain\_server\_address** (*list*) – (optional) DNS server addresses
- **enable\_antivirus** (*bool*) – (optional) Enable antivirus (required DNS)
- **enable\_gti** (*bool*) – (optional) Enable GTI
- **extra\_opts** (*dict*) – extra options as a dict to be passed to the top level engine
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for

configuring LLDP :param str cluster\_mode: Defines whether the clustered engines are all online balancing the

traffic or whether one node is online at a time and the other engines are used as backup

#### Parameters

**reverse\_connection** (*boolean*) – Reverse connection.

:param node\_definition information for the node itself :raises CreateEngineFailed: Failure to create with reason :return: *smc.core.engine.Engine*

## 14.7.7 MasterEngineCluster

**class** *smc.core.engines.MasterEngineCluster*(*name=None, \*\*meta*)

Master Engine Cluster Clusters are currently supported in an active/standby configuration only.

**classmethod** **create**(*name, master\_type, macaddress, nodes, nodes\_definition=None, mgmt\_interface=0, log\_server\_ref=None, domain\_server\_address=None, enable\_gti=False, enable\_antivirus=False, comment=None, extra\_opts=None, lldp\_profile=None, cluster\_mode='standby', reverse\_connection=False, \*\*kw*)

Create Master Engine Cluster

#### Parameters

- **name** (*str*) – name of master engine engine
- **master\_type** (*str*) – firewall|
- **mgmt\_ip** (*str*) – ip address for management interface
- **mgmt\_netmask** (*str*) – full netmask for management
- **mgmt\_interface** (*str*) – interface to use for mgmt (default: 0)
- **nodes** (*list*) – address/network\_value/nodeid combination for cluster nodes
- **log\_server\_ref** (*str*) – (optional) href to log\_server instance
- **domain\_server\_address** (*list*) – (optional) DNS server addresses
- **enable\_antivirus** (*bool*) – (optional) Enable antivirus (required DNS)
- **enable\_gti** (*bool*) – (optional) Enable GTI
- **extra\_opts** (*dict*) – extra options as a dict to be passed to the top level engine

- **lldp\_profile** ([LLDPProfile](#)) – LLDP Profile represents a set of attributes used for configuring LLDP :param str cluster\_mode: Defines whether the clustered engines are all online balancing the

traffic or whether one node is online at a time and the other engines are used as backup

#### Parameters

**reverse\_connection** (*boolean*) – Reverse connection.

:param list nodes\_definition : list of node info (name, comment..) :raises CreateEngineFailed: Failure to create with reason :return: [smc.core.engine.Engine](#)

Example nodes parameter input:

```
[{'address': '5.5.5.2',
  'network_value': '5.5.5.0/24',
  'nodeid': 1},
 {'address': '5.5.5.3',
  'network_value': '5.5.5.0/24',
  'nodeid': 2},
 {'address': '5.5.5.4',
  'network_value': '5.5.5.0/24',
  'nodeid': 3}]
```

### 14.7.8 CloudSGSingleFW

**class** `smc.core.engines.CloudSGSingleFW`(*name=None, \*\*meta*)

Creates a Cloud Firewall with a default dynamic interface To instantiate and create, call ‘create\_dynamic’ class-method as follows:

```
engine = CloudSGSingleFW.create_dynamic(interface_id=0,
                                         name='Cloud Single firewall 1')
```

**classmethod** `create_dynamic`(*name, interface\_id, dynamic\_index=1, reverse\_connection=True, automatic\_default\_route=True, domain\_server\_address=None, loopback\_ndi='127.0.0.1', location\_ref=None, log\_server\_ref=None, zone\_ref=None, enable\_gti=False, enable\_antivirus=False, sidewinder\_proxy\_enabled=False, known\_host\_lists=[], default\_nat=False, comment=None, extra\_opts=None, \*\*kw*)

Create a single layer 3 firewall with only a single DHCP interface. Useful when creating virtualized engine’s such as in Microsoft Azure.

#### Parameters

- **name** (*str*) – name of engine
- **interface\_id** (*str, int*) – interface ID used for dynamic interface and management
- **reverse\_connection** (*bool*) – specifies the dynamic interface should initiate connections to management (default: True)
- **automatic\_default\_route** (*bool*) – allow SMC to create a dynamic netlink for the default route (default: True)
- **domain\_server\_address** (*list*) – list of IP addresses for engine DNS

- **loopback\_ndi** (*str*) – IP address for a loopback NDI. When creating a dynamic engine, the *auth\_request* must be set to a different interface, so loopback is created
- **location\_ref** (*str*) – location by name for the engine
- **log\_server\_ref** (*str*) – log server reference, will use the default or first retrieved if not specified
- **lldp\_profile** (*LLDPProfile*) – LLDP Profile represents a set of attributes used for

configuring LLDP :param dict extra\_opts: extra options as a dict to be passed to the top level engine :param bool quic\_enabled: (optional) include QUIC ports for web traffic :param bool discard\_quic\_if\_cant\_inspect: (optional) discard or allow QUIC :param node\_definition information for the node itself

if inspection is not possible

#### Raises

*CreateElementFailed* – failed to create engine

#### Returns

*smc.core.engine.Engine*

## 14.8 Dynamic Routing Elements

### 14.8.1 RouteMap

Route map rules and match condition elements for dynamic routing policies.

A RouteMap can be created and subsequent rules can be inserted within the route map policy.

A MatchCondition is the subject of the rule providing criteria to specify how a match is made. Elements used in match conditions are *next\_hop*, *peer\_address*, *access\_list* and type *metric*.

#### See also:

*MatchCondition* for more details on how to add match conditions to a rule or modify an existing rule.

Example of creating a RouteMap and subsequent rule, specifying match condition options as keyword arguments:

```
>>> from smc.routing.route_map import RouteMap
>>> from smc.routing.access_list import IPAccessList
>>> from smc.routing.bgp import ExternalBGPPeer
...
>>> rm = RouteMap.create(name='myroutemap')
>>> rm
RouteMap(name=myroutemap)
>>> rm.route_map_rules.create(name='rule1', action='permit',
                             next_hop=IPAccessList('myacl'), peer_address=ExternalBGPPeer('bgppeer'),
                             metric=20)
RouteMapRule(name=rule1)
...
>>> rule1 = rm.route_map_rules.get(0) # retrieve rule 1 from the route map
>>> for condition in rule1.match_condition:
...     condition
...
Condition(rank=1, element=ExternalBGPPeer(name=bgppeer), type=u'peer_address')
```

(continues on next page)

(continued from previous page)

```
Condition(rank=2, element=IPAccessList(name=myacl), type='access_list')
Condition(rank=3, element=Metric(value=20), type=u'metric')
```

Instead of providing singular match condition keywords to the *create* constructor, you can also optionally provide a MatchCondition instance when creating a rule:

```
>>> from smc.routing.route_map import MatchCondition
>>> condition = MatchCondition()
>>> condition.add_access_list(IPAccessList('myacl'))
>>> condition.add_peer_address(ExternalBGPPeer('bgppeer'))
>>> condition.add_metric(20)
>>> condition
MatchCondition(entries=3)
>>> rm.route_map_rules.create(
...     name='foo2',
...     finish=False,
...     match_condition=condition)
RouteMapRule(name=foo2)
```

To remove a match condition, first obtain its rank. After making the modification be sure to call update on the rule element:

```
>>> rule = rm.route_map_rules.get(0)
>>> rule.match_condition.remove_condition(rank=2)
>>> rule.update()
```

You can also delete a rule by obtaining the rule, either through the route\_map\_rules collection reference or by iteration:

```
rule = rm.route_map_rules.get(1)
rule.delete()
```

Or by the name:

```
rule = rm.route_map_rules.get_exact('foo')
rule.delete()
```

See also:

[smc.base.collection.rule\\_collection](#)

**class** smc.routing.route\_map.MatchCondition(rule=None)

Bases: [object](#)

MatchCondition is an iterable container class that holds the match conditions for the route map rule. The list of conditions are ranked in order. You can add, remove and view conditions currently configured in this rule. After making modifications, call update on the rule to commit back to SMC.

When iterating over a match condition, a namedtuple is returned that provides the rank and element type for the condition. It is then possible to add by rank (ie: insert conditions in between others), or remove based on rank. If not rank is provided when adding new conditions, the condition is added to the bottom of the rank list.

**Return type**

[list\(Condition\)](#)

**add\_access\_list**(*accesslist*, *rank=None*)

Add an access list to the match condition. Valid access list types are IPAccessList (v4 and v6), IPPrefixList (v4 and v6), AS Path, CommunityAccessList, ExtendedCommunityAccessList.

**add\_metric**(*value*, *rank=None*)

Add a metric to this match condition

**Parameters**

**value** (*int*) – metric value

**add\_next\_hop**(*access\_or\_prefix\_list*, *rank=None*)

Add a next hop condition. Next hop elements must be of type IPAccessList or IPPrefixList.

**Raises**

*ElementNotFound* – If element specified does not exist

**add\_peer\_address**(*ext\_bgp\_peer\_or\_fw*, *rank=None*)

Add a peer address. Peer address types are ExternalBGPPeer or Engine.

**Raises**

*ElementNotFound* – If element specified does not exist

**remove\_condition**(*rank*)

Remove a condition element using it's rank. You can find the rank and element for a match condition by iterating the match condition:

```
>>> rule1 = rm.route_map_rules.get(0)
>>> for condition in rule1.match_condition:
...     condition
...
Condition(rank=1, element=ExternalBGPPeer(name=bgppeer))
Condition(rank=2, element=IPAccessList(name=myacl))
Condition(rank=3, element=Metric(value=20))
```

Then delete by rank. Call update on the rule after making the modification.

**Parameters**

**rank** (*int*) – rank of the condition to remove

**Raises**

*UpdateElementFailed* – failed to update rule

**Returns**

None

**class** smc.routing.route\_map.RouteMap(*name=None*, *\*\*meta*)

Bases: *Element*

Use Route Map elements in more complex networks to control or manipulate routes. You can use Access List elements as a Matching Condition in a Route Map rule. RouteMaps are rule lists similar to normal policies and can be iterated:

```
>>> from smc.routing.route_map import RouteMap
>>> rm = RouteMap('myroutemap')
>>> for rule in rm.route_map_rules:
...     rule
...
RouteMapRule(name=Rule @115.13)
RouteMapRule(name=Rule @117.0)
```

**classmethod** `create(name, comment=None)`

Create a new route map. After creation, you can add a rule and subsequent match conditions.

**Parameters**

- **name** (*str*) – name of route map
- **comment** (*str*) – optional comment

**Raises**

*CreateElementFailed* – failed creating route map

**Return type**

*RouteMap*

**property** `route_map_rules`

IPv6NAT Rule entry point

**Return type**

*rule\_collection(IPv6NATRule)*

**search\_rule**(*search*)

Search the RouteMap policy using a search string

**Parameters**

**search** (*str*) – search string for a contains match against the rule name and comments field

**Return type**

*list(RouteMapRule)*

**class** `smc.routing.route_map.RouteMapRule(**meta)`

Bases: *RuleCommon*, *SubElement*

A route map rule represents the rules to be processed for a route map assigned to a specific BGP network. A match condition can be provided which encapsulates using dynamic routing element types such as *IPAccessList*, *IPPrefixList*, etc.

**property** `action`

Action for this route map rule. Valid actions are ‘permit’ and ‘deny’.

**Return type**

*str*

**call\_route\_map**(*route\_map*)

Call another route map after match of this rule. Call update on the rule to save after modification.

**Parameters**

**route\_map** (*RouteMap*) – Pass the route map element

**Raises**

*ElementNotFound* – invalid RouteMap reference passed

**Returns**

*None*

**property** `comment`

Get and set the comment for this rule.

**Parameters**

**value** (*str*) – string comment

**Return type**

*str*

```
create(name, action='permit', goto=None, finish=False, call=None, comment=None, add_pos=None,
        after=None, before=None, **match_condition)
```

Create a route map rule. You can provide match conditions by using keyword arguments specifying the required types. You can also create the route map rule and add match conditions after.

#### Parameters

- **name** (*str*) – name for this rule
- **action** (*str*) – permit or deny
- **goto** (*str*) – specify a rule section to goto after if there is a match condition. This will override the finish parameter
- **finish** (*bool*) – finish stops the processing after a match condition. If finish is False, processing will continue to the next rule.
- **call** (*RouteMap*) – call another route map after matching.
- **comment** (*str*) – optional comment for the rule
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with **after** and **before** params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with **add\_pos** and **before** params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with **add\_pos** and **after** params.
- **match\_condition** – keyword values identifying initial values for the match condition. Valid keyword arguments are 'access\_list', 'next\_hop', 'metric' and 'peer\_address'. You can also optionally pass the keyword 'match\_condition' with an instance of MatchCondition.

#### Raises

- **CreateRuleFailed** – failure to insert rule with reason
- **ElementNotFound** – if references elements in a match condition this can be raised when the element specified is not found.

#### See also:

[\*MatchCondition\*](#) for valid elements and expected values for each type.

#### property finish

Is rule action goto set to finish on this rule match. If finish is False, then the policy will proceed to the next rule.

#### Return type

*bool*

#### property goto

If the rule is set to goto a rule section, return the rule section, otherwise it will return None. Check the value of finish to determine if the rule is set to finish on match.

#### Returns

RouteMap or None

**goto\_rule\_section**(*rule\_section*)

Set this rule to goto a specific rule section after match. If goto is None, then check value of finish.

**Parameters**

**rule\_section** (*RouteMapRule*) – pass rule section

**Returns**

None

**property is\_disabled**

Is the rule disabled

**Return type**

bool

**property match\_condition**

Return the match condition for this rule. This can then be modified in place. Be sure to call update on the rule to save.

**Return type**

*MatchCondition*

**class** smc.routing.route\_map.**Metric**(*value*)

Bases: *tuple*

A metric is a simple namedtuple for returning a Metric route map element

**Variables**

**value** (*int*) – metric value for this BGP route

**class** smc.routing.route\_map.**Condition**(*rank, element, type*)

Bases: *tuple*

A condition defines the type of dynamic element that is used in the match condition field of a route map.

**Variables**

- **rank** (*str*) – the rank in the match condition list
- **element** (*str*) – the dynamic element type for this condition
- **type** (*str*) – type defines the type of entry, i.e. metric, peer\_address, next\_hop, access\_list

## 14.8.2 IPAccessList

AccessList module represents functionality that support dynamic routing filters based on IPv4 or IPv6 access lists such as OSPF and BGP.

**class** smc.routing.access\_list.**AccessList**

Bases: *object*

AccessList provides an iterable container that allows simple iteration over existing IPAccessList (v4 and v6), IPPrefixList (v4 and v6), CommunityAccessList and ExtendedCommunityAccessList entries. When using the *create* constructor, validate the keyword arguments based on the specific access list requirements.

**Returns**

namedtuple based on access list type

**add\_entry**(\*\*kw)

Add an entry to an AccessList. Use the supported arguments for the inheriting class for keyword arguments.

**Raises**

*UpdateElementFailed* – failure to modify with reason

**Returns**

None

**classmethod create**(name, entries=None, comment=None, \*\*kw)

Create an Access List Entry.

Depending on the access list type you are creating (IPAccessList, IPv6AccessList, IPPrefixList, IPv6PrefixList, CommunityAccessList, ExtendedCommunityAccessList), entries will define a dict of the valid attributes for that ACL type. Each class has a defined list of attributes documented in it's class.

You can optionally leave entries blank and use the *add\_entry()* method after creating the list container.

**Parameters**

- **name** (*str*) – name of IP Access List
- **entries** (*list*) – access control entry
- **kw** – optional keywords that might be necessary to create the ACL (see specific Access Control List documentation for options)

**Raises**

*CreateElementFailed* – cannot create element

**Returns**

The access list based on type

**remove\_entry**(\*\*field\_value)

Remove an AccessList entry by field specified. Use the supported arguments for the inheriting class for keyword arguments.

**Raises**

*UpdateElementFailed* – failed to modify with reason

**Returns**

None

**classmethod update\_or\_create**(with\_status=False, overwrite\_existing=False, \*\*kw)

Update or create the Access List. This method will not attempt to evaluate whether the access list has differences, instead it will update with the contents of the payload entirely. If the intent is to only add or remove a single entry, use *~add\_entry* and *~remove\_entry* methods.

**Parameters**

- **with\_status** (*bool*) – return with 3-tuple of (Element, modified, created) holding status
- **overwrite\_existing** (*bool*) – if the access list exists but instead of an incremental update you want to overwrite with the newly defined entries, set this to True (default: False)

**Returns**

Element or 3-tuple with element and status

**class** smc.routing.access\_list.**IPAccessList**(name=None, \*\*meta)

Bases: *AccessList*, *Element*

IPAccessList is used by dynamic routing protocols to allow filtering of routes. Protocols like OSPF and BGP allow inbound and outbound filters using these.

Create an IPAccessList. When providing values for *entries* to the create constructor, use valid attributes as defined in [AccessListEntry](#):

```
>>> ip = IPAccessList.create(name='mylist', entries=[
    {'subnet': '1.1.1.0/24', 'action': 'permit'},
    {'subnet': '2.2.2.0/24', 'action': 'deny'}])
...
>>> ip.add_entry(subnet='3.3.3.0/24', action='permit')
>>> ip.remove_entry(subnet='1.1.1.0/24')
>>> ip.update()
'https://172.18.1.151:8082/6.4/elements/ip_access_list/13'
>>> for entry in ip:
...     entry
...
AccessListEntry(subnet=u'2.2.2.0/24', action=u'deny', comment=None)
AccessListEntry(subnet=u'3.3.3.0/24', action=u'permit', comment=None)
...
>>> ip.delete()
```

This is an iterable container yielding [AccessListEntry](#)

See also:

[AccessListEntry](#) for valid *create* and add/remove parameters

**class** smc.routing.access\_list.IPv6AccessList(name=None, \*\*meta)

Bases: [AccessList](#), [Element](#)

IPv6AccessList is used by dynamic routing protocols to allow filtering of routes. Protocols like OSPF and BGP allow inbound and outbound filters using these.

```
>>> acl6 = IPv6AccessList.create(name='aclv6', entries=[
...     {'subnet': '2001:db8:1::1/128', 'action': 'permit'}])
>>> acl6
IPv6AccessList(name=aclv6)
>>> for entry in acl6:
...     entry
...
AccessListEntry(subnet=u'2001:db8:1::1/128', action=u'permit', comment=None)
```

This is an iterable container yielding [AccessListEntry](#)

See also:

[IPAccessList](#) for using this element.

**class** smc.routing.access\_list.AccessListEntry(subnet, action, comment)

Bases: [tuple](#)

An AccessListEntry defines a simple entry for an IPAccessList used in dynamic routing configurations.

#### Variables

- **subnet** (*str*) – subnet associated with this entry
- **action** (*str*) – action for the entry
- **comment** (*str*) – optional comment for the entry

### 14.8.3 IPPrefixList

IP Prefix module represents prefix lists that can be used to filter networks for OSPF routing.

**class** `smc.routing.prefix_list.IPPrefixList(name=None, **meta)`

Bases: [AccessList](#), [Element](#)

An IP prefix list specifies a list of networks. When you apply an IP prefix list to a neighbor, the device sends or receives only a route whose destination is in the IP prefix list.

Creating and modifying an IPAccessList is similar to other access list methods:

```
>>> prefix = IPPrefixList.create(name='mylist', entries=[
...     {'subnet': '10.0.0.0/8', 'min_prefix_length': 16,
...       'max_prefix_length': 32, 'action': 'deny'},
...     {'subnet': '192.16.1.0/24', 'min_prefix_length': 25,
...       'max_prefix_length': 32, 'action': 'permit'}])
>>> prefix
IPPrefixList(name=mylist)
...
>>> prefix.add_entry(subnet='192.17.1.0/24', min_prefix_length=25,
...                   max_prefix_length=32, action='deny')
>>> prefix.update()
'https://172.18.1.151:8082/6.4/elements/ip_prefix_list/16'
>>> prefix.remove_entry(subnet='192.16.1.0/24')
>>> prefix.update()
'https://172.18.1.151:8082/6.4/elements/ip_prefix_list/16'
>>> for entry in prefix:
...     entry
...
PrefixListEntry(subnet=u'10.0.0.0/8', action=u'deny', min_prefix_length=16,
                 max_prefix_length=32, comment=None)
PrefixListEntry(subnet=u'192.17.1.0/24', action=u'deny', min_prefix_length=25,
                 max_prefix_length=32, comment=None)
```

**You can also create a PrefixList without using the min\_prefix\_length and max\_prefix\_length fields:**

```
>>> prefix = IPPrefixList.create(name='mylist', entries=[
...     {'subnet': '10.0.0.0/8', 'action': 'deny'}, ... {'subnet': '192.16.1.0/24', 'action': 'permit'}])
```

This is an iterable container yielding [PrefixListEntry](#)

**See also:**

[PrefixListEntry](#) for valid *create* and *add/remove* parameters

**class** `smc.routing.prefix_list.Ipv6PrefixList(name=None, **meta)`

Bases: [AccessList](#), [Element](#)

An IP prefix list specifies a list of networks. When you apply an IP prefix list to a neighbor, the device sends or receives only a route whose destination is in the IP prefix list.

```
>>> prefix6 = Ipv6PrefixList.create(name='myipv6', entries=[
...     {'subnet': 'ab00::/64', 'min_prefix_length': 65, 'max_prefix_length': 128,
```

(continues on next page)

(continued from previous page)

```

        'action': 'deny']])
>>> prefix6
IPv6PrefixList(name=myipv6)
>>> for entry in prefix6:
...     entry
...
PrefixListEntry(subnet=u'ab00::/64', action=u'deny', min_prefix_length=65,
                 max_prefix_length=128, comment=None)

```

You can also create a `PrefixList` without using the `min_prefix_length` and `max_prefix_length` fields:

```

>>> prefix = IPPrefixList.create(name='mylist', entries=[
...     {'subnet': 'ab00::/64', 'action': 'deny'}

```

This is an iterable container yielding `PrefixListEntry`

See also:

`IPPrefixList` for other common operations

**class** `smc.routing.prefix_list.PrefixListEntry`(*subnet*, *action*, *min\_prefix\_length*, *max\_prefix\_length*, *comment*)

Bases: `tuple`

A `PrefixListEntry` defines a simple entry for an `PrefixList` used in dynamic routing configurations.

**Variables**

- **subnet** (*str*) – subnet associated with this entry
- **action** (*str*) – action for the entry
- **min\_prefix\_length** (*int*) – minimum mask bits
- **max\_prefix\_length** (*int*) – maximum mask bits
- **comment** (*str*) – optional comment for the entry

## 14.8.4 BGP Elements

BGP Module representing BGP settings for Forcepoint NGFW layer 3 engines. BGP can be enabled and run on either single/cluster layer 3 firewalls or virtual engines.

For adding BGP configurations, several steps are required:

- Enable BGP on the engine and specify the BGP Profile
- Create or use an existing OSPF Area to be used
- Modify the routing interface and add the BGP Peering

Enable BGP on an existing engine using the default BGP system profile:

```

engine.bgp.enable(
    autonomous_system=AutonomousSystem('myAS')
    announced_networks=[Network('172.18.1.0/24'), Network('1.1.1.0/24')])

```

Create a BGP Peering using the default BGP Connection Profile:

```
BGPPeering.create(name='mypeer')
```

Add the BGP Peering to the routing interface:

```
interface = engine.routing.get(0)
interface.add_bgp_peering(
    BGPPeering('mypeer'),
    ExternalBGPPeer('neighbor'))
```

Disable BGP on an engine:

```
engine.bgp.disable()
```

Finding profiles or elements can also be done through collections:

```
>>> list(BGPProfile.objects.all())
[BGPProfile(name=Default BGP Profile)]

>>> list(ExternalBGPPeer.objects.all())
[ExternalBGPPeer(name=bgp-02), ExternalBGPPeer(name=Amazon AWS),
↪ ExternalBGPPeer(name=bgp-01)]
```

The BGP relationship can be represented as:

```
Engine --uses an--> (BGP Profile --and--> Autonomous System --and--> Announced Networks)
Engine Routing --uses-an--> BGP Peering --has-a--> External BGP Peer
```

Only Layer3Firewall and Layer3VirtualEngine types can support running BGP.

See also:

[\*smc.core.engines.Layer3Firewall\*](#) and [\*smc.core.engines.Layer3VirtualEngine\*](#)

**class** `smc.routing.bgp.BGP`(data=None)

BGP represents the BGP configuration on a given engine. An instance is returned from an engine reference:

```
engine = Engine('myengine')
engine.dynamic_routing.bgp.status
engine.dynamic_routing.bgp.announced_networks
...
```

When making changes to the BGP configuration, any methods called that change the configuration also require that `engine.update()` is called once changes are complete. This way you can make multiple changes without refreshing the engine cache.

For example, adding advertised networks to the configuration:

```
engine.dynamic_routing.bgp.update_configuration(announced_networks=[Network('foo')])
engine.update()
```

### Variables

- **autonomous\_system** ([`AutonomousSystem`](#)) – AS reference for this BGP configuration
- **profile** ([`BGPProfile`](#)) – BGP profile reference for this configuration

**property announced\_networks**

Show all announced networks for the BGP configuration. Returns tuple of advertised network, routemap. Route map may be None.

```
for advertised in engine.bgp.advertisements:
    net, route_map = advertised
```

**Returns**

list of tuples (advertised\_network, route\_map).

**property bmp\_router\_id**

Get the BMP router ID for this BGP configuration. Directly linked to 'bmp\_router\_id\_type' attribute:

- [0-255]:[0-65535]: AS Number : dedicated number
- V.X.Y.Z:[0-255]: IPv4 Address : AS Number
- [0-65535]:[0-255]: AS Number : dedicated number

**Returns**

str or None

**property bmp\_router\_id\_type**

Get the BMP router ID type for this BGP configuration. Accepted values:

- 0: [0-255]:[0-65535] format
- 1: V.X.Y.Z:[0-255] format
- 2: [0-65535]:[0-255] format

**Returns**

str or None

**disable()**

Disable BGP on this engine.

**Returns**

None

**enable(autonomous\_system, announced\_networks, router\_id=None, bgp\_profile=None)**

Enable BGP on this engine. On master engine, enable BGP on the virtual firewall. When adding networks to *announced\_networks*, the element types can be of type *smc.elements.network.Host*, *smc.elements.network.Network* or *smc.elements.group.Group*. If passing a Group, it must have element types of host or network.

Within *announced\_networks*, you can pass a 2-tuple that provides an optional *smc.routing.route\_map.RouteMap* if additional policy is required for a given network.

```
engine.dynamic_routing.bgp.enable(
    autonomous_system=AutonomousSystem('aws_as'),
    announced_networks=[Network('bgpnet'), Network('inside')],
    router_id='10.10.10.10')
```

**Parameters**

- **autonomous\_system** (*str*, [AutonomousSystem](#)) – provide the AS element or str href for the element
- **bgp\_profile** (*str*, [BGPProfile](#)) – provide the BGPProfile element or str href for the element; if None, use system default
- **announced\_networks** (*list*) – list of networks to advertise via BGP Announced networks can be single networks, host or group elements or a 2-tuple with the second tuple item being a routemap element
- **router\_id** (*str*) – router id for BGP, should be an IP address. If not set, automatic discovery will use default bound interface as ID.

**Raises**

[ElementNotFound](#) – OSPF, AS or Networks not found

**Returns**

None

---

**Note:** For arguments that take str or Element, the str value should be the href of the element.

---

**property router\_id**

Get the router ID for this BGP configuration. If None, then the ID will use the interface IP.

**Returns**

str or None

**property status**

Is BGP enabled on this engine.

**Return type**

bool

**update\_configuration(\*\*kwargs)**

Update configuration using valid kwargs as defined in the enable constructor.

**Parameters**

**kwargs** (*dict*) – kwargs to satisfy valid args from *enable*

**Return type**

bool

**class** smc.routing.bgp.BGPAggregationEntry(*data*)

**classmethod** create(*mode*, *subnet*)

**Parameters**

- **mode** (*str*) – The Aggregation value: 1.aggregate: Aggregate mode. 2.aggregate\_as\_set: Aggregate with AS Set mode. 3.summary\_only: Summary Only mode. 4.as\_set\_and\_summary: Aggregate with AS Set and Summary mode.
- **subnet** (*str*) – The Subnet Network element.

**Return type**

dict

**class** smc.routing.bgp.BGPBMPSettings(*bmp\_address*, *bmp\_port*, *bmp\_connect\_through\_master*)

A BGP BMP Entry representing the BMP listening address (FQDN/IPv4/IPv6), the BMP listening port and the flag to know if the connection must be done through master.

```
class smc.routing.bgp.RedistributionEntry(data)
```

```
classmethod create(redistribution_type, enabled=False, filter_type=None, **kwargs)
```

#### Parameters

- **redistribution\_type** (*str*) – The redistribution type: 1.kernel: Redistribution from Kernel. 2.static: Redistribution from Static. 3.connected: Redistribution from Connected. 4.bgp: Redistribution from BGP. Available for any dynamic routing protocols Profile element but BGP one. 5.ospfv2: Redistribution from OSPFv2. Available for any dynamic routing protocols Profile element but OSPFv2 one. 6.default\_originate: Default Originate injection. Available just for OSPFv2 element.
- **enabled** (*bool*) – This redistribution is enabled or not.
- **filter\_type** (*str*) – The filter type.

#### Return type

dict

### 14.8.4.1 AutonomousSystem

```
class smc.routing.bgp.AutonomousSystem(name=None, **meta)
```

Bases: *Element*

Autonomous System for BGP routing. AS is a required setting when enabling BGP on an engine and specifies a unique identifier for routing communications.

#### property as\_number

The AS Number for this autonomous system

#### Returns

AS number

#### Return type

int

```
classmethod create(name, as_number, comment=None)
```

Create an AS to be applied on the engine BGP configuration. An AS is a required parameter when creating an ExternalBGPPeer. You can also provide an AS number using an 'asdot' syntax:

```
AutonomousSystem.create(name='myas', as_number='200.600')
```

#### Parameters

- **name** (*str*) – name of this AS
- **as\_number** (*int*) – AS number preferred
- **comment** (*str*) – optional string comment

#### Raises

- *CreateElementFailed* – unable to create AS
- *ValueError* – If providing AS number in dotted format and low/high order bytes are > 65535.

#### Returns

instance with meta

**Return type***AutonomousSystem***classmethod** `update_or_create`(*with\_status=False*, *\*\*kwargs*)

Update or create the element. If the element exists, update it using the kwargs provided if the provided kwargs after resolving differences from existing values. When comparing values, strings and ints are compared directly. If a list is provided and is a list of strings, it will be compared and updated if different. If the list contains unhashable elements, it is skipped. To handle complex comparisons, override this method on the subclass and process the comparison separately. If an element does not have a *create* classmethod, then it is considered read-only and the request will be redirected to `get()`. Provide a *filter\_key* dict key/value if you want to match the element by a specific attribute and value. If no *filter\_key* is provided, the name field will be used to find the element.

```
>>> host = Host('kali')
>>> print(host.address)
12.12.12.12
>>> host = Host.update_or_create(name='kali', address='10.10.10.10')
>>> print(host, host.address)
Host(name=kali) 10.10.10.10
```

**Parameters**

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements *create* method.
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

**Raises**

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

**Returns**

element instance by type

**Return type***Element*

#### 14.8.4.2 ExternalBGPPeer

**class** `smc.routing.bgp.ExternalBGPPeer`(*name=None*, *\*\*meta*)Bases: *Element*

An External BGP represents the AS and IP settings for a remote BGP peer. Creating a BGP peer requires that you also pre-create an *AutonomousSystem* element:

```
AutonomousSystem.create(name='neighborA', as_number=500)
ExternalBGPPeer.create(name='name',
                       neighbor_as_ref=AutonomousSystem('neighborA'),
                       neighbor_ip='1.1.1.1')
```

**Variables****neighbor\_as** (*AutonomousSystem*) – AS for this external BGP peer

**classmethod** `create(name, neighbor_as, neighbor_ip, neighbor_port=179, comment=None)`

Create an external BGP Peer.

**Parameters**

- **name** (*str*) – name of peer
- **neighbor\_as\_ref** (*str*, *AutonomousSystem*) – AutonomousSystem element or href.
- **neighbor\_ip** (*str*) – ip address of BGP peer
- **neighbor\_port** (*int*) – port for BGP, default 179.

**Raises**

*CreateElementFailed* – failed creating

**Returns**

instance with meta

**Return type**

*ExternalBGPPeer*

**property neighbor\_ip**

IP address of the external BGP Peer

**Returns**

ipaddress of external bgp peer

**Return type**

*str*

**property neighbor\_port**

Port used for neighbor AS

**Returns**

neighbor port

**Return type**

*int*

### 14.8.4.3 BGPPeering

**class** `smc.routing.bgp.BGPPeering(name=None, **meta)`

Bases: *Element*

BGP Peering is applied directly to an interface and defines basic connection settings. A BGPConnectionProfile is required to create a BGPPeering and if not provided, the default profile will be used.

The most basic peering can simply specify the name of the peering and leverage the default BGPConnectionProfile:

```
BGPPeering.create(name='my-aws-peer')
```

**Variables**

**connection\_profile** (*BGPConnectionProfile*) – BGP connection profile for this peering

```
classmethod create(name, connection_profile_ref=None, md5_password=None,
                    local_as_option='not_set', local_as_value=None, max_prefix_option='not_enabled',
                    max_prefix_value=None, send_community='no', connected_check='disabled',
                    orf_option='disabled', next_hop_self=True, override_capability=False,
                    dont_capability_negotiate=False, remote_private_as=False,
                    route_reflector_client=False, soft_reconfiguration=True, ttl_option='disabled',
                    ttl_value=None, inbound_rm_filter=None, inbound_ip_filter=None,
                    inbound_ipv6_filter=None, inbound_ipprefix_filter=None,
                    inbound_ipv6prefix_filter=None, inbound_aspath_filter=None,
                    outbound_rm_filter=None, outbound_ip_filter=None, outbound_ipv6_filter=None,
                    outbound_ipprefix_filter=None, outbound_ipv6prefix_filter=None,
                    outbound_aspath_filter=None, default_originate=False, bfd_enabled=False,
                    bfd_interval=750, bfd_min_rx=500, bfd_multiplier=3, bfd_passive_mode=False,
                    comment=None)
```

Create a new BGPPeering configuration.

#### Parameters

- **name** (*str*) – name of peering
- **connection\_profile\_ref** (*str*, [BGPPeeringConnectionProfile](#)) – required BGP connection profile. System default used if not provided.
- **md5\_password** (*str*) – optional md5\_password
- **local\_as\_option** (*str*) – The local AS mode. Valid options are: 'not\_set', 'prepend', 'no\_prepend', 'replace\_as'
- **local\_as\_value** (*str*) – The Local AS value. Optional, depending on Local AS mode. Not required.
- **max\_prefix\_option** (*str*) – The max prefix mode. Valid options are: 'not\_enabled', 'enabled', 'warning\_only'
- **max\_prefix\_value** (*int*) – The Max Prefix value. Optional, depending on Max Prefix mode. Not required
- **send\_community** (*str*) – the send community mode. Valid options are: 'no', 'standard', 'extended', 'standard\_and\_extended'
- **connected\_check** (*str*) – the connected check mode. Valid options are: 'disabled', 'enabled', 'automatic'
- **orf\_option** (*str*) – outbound route filtering mode. Valid options are: 'disabled', 'send', 'receive', 'both'
- **next\_hop\_self** (*bool*) – next hop self setting
- **override\_capability** (*bool*) – is override received capabilities
- **dont\_capability\_negotiate** (*bool*) – do not send capabilities
- **remote\_private\_as** (*bool*) – is remote a private AS
- **route\_reflector\_client** (*bool*) – Route Reflector Client (iBGP only)
- **soft\_reconfiguration** (*bool*) – do soft reconfiguration inbound
- **ttl\_option** (*str*) – ttl check mode. Valid options are: 'disabled', 'ttl-security'
- **ttl\_value** (*int*) – The Hops value for TTL Check Mechanism. If not set, it corresponds to Automatic. Not required
- **inbound\_rm\_filter** ([RouteMap](#)) – The Route Map inbound filter. Not required

- **inbound\_ip\_filter** ([IPAccessList](#)) – The IP Access List inbound filter. Not required
- **inbound\_ipv6\_filter** ([IPv6AccessList](#)) – The IPv6 Access List inbound filter. Not required
- **inbound\_ipprefix\_filter** ([IPPrefixList](#)) – The IP Prefix List inbound filter. Not required
- **inbound\_ipv6prefix\_filter** ([IPv6PrefixList](#)) – The IPv6 Prefix List inbound filter. Not required.
- **inbound\_aspath\_filter** ([ASPathAccessList](#)) – The AS Path Access List inbound filter. Not required.
- **outbound\_rm\_filter** ([RouteMap](#)) – The Route Map outbound filter. Not required
- **outbound\_ip\_filter** ([IPAccessList](#)) – The IP Access List outbound filter. Not required
- **outbound\_ipv6\_filter** ([IPv6AccessList](#)) – The IPv6 Access List outbound filter. Not required.
- **outbound\_ipprefix\_filter** ([IPPrefixList](#)) – The IP Prefix List outbound filter. Not required.
- **outbound\_ipv6prefix\_filter** ([IPv6PrefixList](#)) – The IPv6 Prefix List outbound filter. Not required.
- **outbound\_aspath\_filter** ([ASPathAccessList](#)) – The AS Path Access List outbound filter. Not required.
- **default\_originate** ([bool](#)) – Default originate. Default value is false. Not required.
- **bfd\_enabled** ([bool](#)) – Bidirectional Forwarding Detection flag. Default value is false. Not required
- **bfd\_interval** ([int](#)) – Bidirectional Forwarding Detection interval in ms. Not required.
- **bfd\_min\_rx** ([int](#)) – Bidirectional Forwarding Detection min RX in ms. Not required.
- **bfd\_multiplier** ([int](#)) – Bidirectional Forwarding Detection multiplier. Not required.
- **bfd\_passive\_mode** ([bool](#)) – Bidirectional Forwarding Detection passive mode flag. Not required.

**Raises**

[CreateElementFailed](#) – failed creating profile

**Returns**

instance with meta

**Return type**

[BGPPeering](#)

#### 14.8.4.4 BGPProfile

**class** `smc.routing.bgp.BGPProfile`(*name=None*, *\*\*meta*)

Bases: *Element*

A BGP Profile specifies settings specific to an engine level BGP configuration. A profile specifies engine specific settings such as distance, redistribution, and aggregation and port.

These settings are always in effect:

- BGP version 4/4+
- No autosummary
- No synchronization
- Graceful restart

Example of creating a custom BGP Profile with default administrative distances and custom subnet distances:

```
Network.create(name='inside', ipv4_network='1.1.1.0/24')
BGPProfile.create(
    name='bar',
    internal_distance=100,
    external_distance=200,
    local_distance=50,
    subnet_distance=[(Network('inside'), 100)])
```

##### **property** `aggregation_entry`

Specific subnet with mode :return: list of BGPAggregationEntry

##### **property** `bmp_settings`

BMP settings: list of address/port/connect\_through\_master

**classmethod** `create`(*name*, *port=179*, *external\_distance=20*, *internal\_distance=200*, *local\_distance=200*, *subnet\_distance=None*, *bmp\_settings=None*, *aggregation\_entry=None*, *redistribution\_entry=None*)

Create a custom BGP Profile

##### **Parameters**

- **name** (*str*) – name of profile
- **port** (*int*) – port for BGP process
- **external\_distance** (*int*) – external administrative distance; (1-255)
- **internal\_distance** (*int*) – internal administrative distance (1-255)
- **local\_distance** (*int*) – local administrative distance (aggregation) (1-255)
- **subnet\_distance** (*list*) – configure specific subnet's with respective distances
- **BGPBMPSettings** (*list*) – configure the BMP listing settings (address/port/flag)

**:param** `List(BGPAggregationEntry) aggregation_entry`: This represents the BGP aggregation entry

**with:**

subnet: link to a network. mode: the aggregation mode.

**:param** `List(RedistributionEntry) redistribution_entry`: This represents an BGP or OSPF Profile

Redistribution Entry. There is one entry by BGP or OSPF Redistribution type (static, connected, kernel, ospfv2)

**Raises**

*CreateElementFailed* – reason for failure

**Returns**

instance with meta

**Return type**

*BGPProfile*

**property external\_distance**

External administrative distance (eBGP)

**Returns**

distance setting

**Return type**

*int*

**property internal\_distance**

Internal administrative distance (iBGP)

**Returns**

internal distance setting

**Return type**

*int*

**property local\_distance**

Local administrative distance (aggregation)

**Returns**

local distance setting

**Return type**

*int*

**property port**

Specified port for BGP

**Returns**

value of BGP port

**Return type**

*int*

**property redistribution\_entry**

This represents an BGP or OSPF Profile Redistribution Entry. :return: list of RedistributionEntry

**property subnet\_distance**

Specific subnet administrative distances

**Returns**

list of tuple (subnet, distance)

#### 14.8.4.5 BGPConnectionProfile

**class** `smc.routing.bgp.BGPConnectionProfile`(*name=None*, *\*\*meta*)

Bases: *Element*

A BGP Connection Profile will specify timer based settings and is used by a BGPPeering configuration.

Create a custom profile:

```
BGPConnectionProfile.create(
    name='fooprofile',
    md5_password='12345',
    connect_retry=200,
    session_hold_timer=100,
    session_keep_alive=150)
```

**property** `connect_retry`

The connect retry timer, in seconds

**Returns**

connect retry in seconds

**Return type**

*int*

**classmethod** `create`(*name*, *md5\_password=None*, *connect\_retry=120*, *session\_hold\_timer=180*,  
*session\_keep\_alive=60*)

Create a new BGP Connection Profile.

**Parameters**

- **name** (*str*) – name of profile
- **md5\_password** (*str*) – optional md5 password
- **connect\_retry** (*int*) – The connect retry timer, in seconds
- **session\_hold\_timer** (*int*) – The session hold timer, in seconds
- **session\_keep\_alive** (*int*) – The session keep alive timer, in seconds

**Raises**

*CreateElementFailed* – failed creating profile

**Returns**

instance with meta

**Return type**

*BGPConnectionProfile*

**property** `session_hold_timer`

The session hold timer, in seconds

**Returns**

in seconds

**Return type**

*int*

**property** `session_keep_alive`

The session keep alive, in seconds

**Returns**

in seconds

**Return type**

int

**14.8.4.6 ASPathAccessList**

**class** smc.routing.bgp\_access\_list.**ASPathAccessList**(name=None, \*\*meta)

Bases: [AccessList](#), [Element](#)

An AS path is the autonomous systems that routing information passed through to get to a specified router. It indicates the origin of this route. The AS path is used to prevent routing loops in BGP.

ASPathAccessLists can be used as a MatchCondition in a RouteMap:

```
>>> aspath = ASPathAccessList.create(name='aspath', entries=[
...     {'expression': '123-456', 'action': 'permit'},
...     {'expression': '1234-567', 'action': 'deny'}])
>>> aspath
ASPathAccessList(name=aspath)
>>> aspath.add_entry(expression='897', action='permit')
>>> aspath.update()
'https://172.18.1.151:8082/6.4/elements/as_path_access_list/28'
...
>>> aspath.remove_entry(expression='123-456')
>>> aspath.update()
'https://172.18.1.151:8082/6.4/elements/as_path_access_list/28'
>>> for entry in aspath:
...     entry
...
ASPathListEntry(expression=u'1234-567', action=u'deny', comment=None)
ASPathListEntry(expression=u'897', action=u'permit', comment=None)
```

This is an iterable container yielding [ASPathListEntry](#).

**See also:**

[ASPathListEntry](#) for valid *create* and *add/remove* parameters

**class** smc.routing.bgp\_access\_list.**ASPathListEntry**(expression, action, comment)

Bases: [tuple](#)

The ASPathAccessList is an iterable container and will return instances of [ASPathListEntry](#).

**Variables**

- **expression** ([str](#)) – string expression identifying the AS path
- **action** ([str](#)) – ‘permit’ or ‘deny’
- **comment** ([str](#)) – optional comment

#### 14.8.4.7 CommunityAccessList

**class** `smc.routing.bgp_access_list.CommunityAccessList` (*name=None, \*\*meta*)

Bases: [`AccessList`](#), [`Element`](#)

A `CommunityAccessList` is used to provide specific rules for BGP configurations providing and permit/deny capability based on the community defined. `CommunityAccessLists` can be used in a `RouteMap` match condition to refine the policy for a specific announced network.

When creating a new community ACL, *entries* is expecting a list of dict items using the valid field and values of this class. For example:

```
>>> from smc.routing.community_list import CommunityAccessList
>>> comm = CommunityAccessList.create(name='commacl',
    entries=[{'community': 123, 'action': 'permit'},
             {'community': 456, 'action': 'deny'}],
    type='standard')
>>> comm
CommunityAccessList(name=commacl)
```

You can optionally also create an empty access list and use `add_entry()` to insert entries after:

```
>>> comm.add_entry(community=789, action='permit')
>>> comm.update()
```

Iterating the access list will return [`CommunityListEntry`](#):

```
>>> for entries in comm:
...     entries
...
CommunityListEntry(community=u'789', action=u'permit', comment=None)
CommunityListEntry(community=u'456', action=u'deny', comment=None)
CommunityListEntry(community=u'123', action=u'permit', comment=None)
```

The *type* parameter for the *create* constructor can have values *standard* or *expanded*. If using *expanded*, the access list can then use a regex for matching the community string.

This is an iterable container yielding [`CommunityListEntry`](#).

**See also:**

[`CommunityListEntry`](#) for valid *create* and add/remove parameters

##### Variables

**type** (*str*) – ‘standard’ or ‘expanded’ (specify as kw when in *create* constructor when creating the top level access list.

**class** `smc.routing.bgp_access_list.CommunityListEntry` (*community, action, comment*)

Bases: [`tuple`](#)

The `CommunityAccessList` represents the entries for the community access lists.

##### Variables

- **community** (*str*) – community id
- **action** (*str*) – ‘permit’ or ‘deny’
- **comment** (*str*) – optional comment

#### 14.8.4.8 ExtendedCommunityAccessList

**class** smc.routing.bgp\_access\_list.**ExtendedCommunityAccessList**(name=None, \*\*meta)

Bases: [AccessList](#), [Element](#)

Extended community access lists with the ability to specify route target or start of origin for an entry.

ExtendedCommunityAccessLists can be used in a RouteMap match condition to refine the policy for a specific announced network:

```
>>> comm = ExtendedCommunityAccessList.create(name='comm', entries=[
...     {'community': 123, 'action': 'permit', 'type': 'rt'},
...     {'community': 456, 'action': 'deny', 'type': 'soo'}],
...     type='standard')
>>> comm
ExtendedCommunityAccessList(name=comm)
>>> comm.add_entry(community=789, action='permit', type='rt')
>>> comm.update()
...
>>> comm.remove_entry(community=123)
>>> comm.update()
'https://172.18.1.151:8082/6.4/elements/extended_community_access_list/25'
>>> for entry in comm:
...     entry
...
ExtCommunityListEntry(community=u'456', action=u'deny', comment=None, type=u'soo
↪)
ExtCommunityListEntry(community=u'789', action=u'permit', comment=None, type=u
↪'rt')
```

This is an iterable container yielding [ExtCommunityListEntry](#).

**See also:**

[ExtCommunityListEntry](#) for valid *create* and add/remove parameters

##### Variables

**type** (*str*) – ‘standard’ or ‘expanded’ (specify as kw when in *create* constructor when creating the top level access list.

**class** smc.routing.bgp\_access\_list.**ExtCommunityListEntry**(community, action, type)

Bases: [tuple](#)

The ExtCommunityListEntry represents the entries for the extended community access lists.

##### Variables

- **community** (*str*) – community id
- **action** (*str*) – ‘permit’ or ‘deny’
- **type** (*str*) – ‘rt’ (Route Target) or ‘soo’ (Site of Origin) (required)

## 14.8.5 OSPF Elements

Dynamic Routing can be enabled on devices configured in engine/VPN mode. Configuring dynamic routing consists of enabling the routing protocol on the engine and adding the routing elements on the interfaces at the engine routing level.

For adding OSPF configurations, several steps are required:

- Enable OSPF on the engine and specific the OSPF Profile
- Create or locate an existing OSPFArea to be used
- Modify the routing interface and add the OSPFArea

Enable OSPF on an existing engine using the default OSPF system profile:

```
engine.ospf.enable()
```

Create an OSPFArea using the default OSPF Interface Setting profile:

```
OSPFArea.create(name='customOSPFArea')
```

Add OSPF area to an interface routing configuration (add to nicid '0'):

```
interface = engine.routing.get(0)
interface.add_ospf_area(area)
```

Disable OSPF on an engine:

```
engine.ospf.disable()
```

Finding profiles or elements can also be done through collections:

```
>>> list(OSPFProfile.objects.all())
[OSPFProfile(name=Default OSPFv2 Profile)]

>>> list(OSPFArea.objects.all())
[OSPFArea(name=area0)]
```

The OSPF relationship can be represented as:

```
Engine --uses an--> OSPF Profile --has-a--> OSPF Domain Setting
Engine Routing --uses-an--> OSPF Area --has-a--> OSPF Interface Setting
```

Only Layer3Firewall and Layer3VirtualEngine types can support running OSPF.

**See also:**

[\*smc.core.engines.Layer3Firewall\*](#) and [\*smc.core.engines.Layer3VirtualEngine\*](#)

**class** `smc.routing.ospf.OSPF`(data=None)

OSPF configuration on the engine. Access through an engine reference:

```
engine.dynamic_routing.ospf.status
engine.dynamic_rotuing.ospf.enable(...)
```

When making changes to the OSPF configuration, any methods called that change the configuration also require that `engine.update()` is called once changes are complete. This way you can make multiple changes without refreshing the engine cache.

**Variables**

**profile** ([OSPFProfile](#)) – OSPFProfile reference for this engine

**disable()**

Disable OSPF on this engine.

**Returns**

None

**enable**(*ospf\_profile=None, router\_id=None*)

Enable OSPF on this engine. For master engines, enable OSPF on the virtual firewall.

Once enabled on the engine, add an OSPF area to an interface:

```
engine.dynamic_routing.ospf.enable()
interface = engine.routing.get(0)
interface.add_ospf_area(OSPFArea('myarea'))
```

**Parameters**

- **ospf\_profile** (*str*, [OSPFProfile](#)) – OSPFProfile element or str href; if None, use default profile
- **router\_id** (*str*) – single IP address router ID

**Raises**

[ElementNotFound](#) – OSPF profile not found

**Returns**

None

**property router\_id**

Get the router ID for this OSPF configuration. If None, then the ID will use the interface IP.

**Returns**

str or None

**property status**

Is OSPF enabled on this engine.

**Return type**

bool

**update\_configuration**(*\*\*kwargs*)

Update the OSPF configuration using kwargs that match the *enable* constructor.

**Parameters**

**kwargs** (*dict*) – keyword arguments matching enable constructor.

**Returns**

whether change was made

**Return type**

bool

### 14.8.5.1 OSPFArea

**class** `smc.routing.ospf.OSPFArea`(*name=None, \*\*meta*)

Bases: *Element*

OSPF Area is an element that identifies general settings for an OSPF configuration applied to an engine routing node. The OSPFArea has a reference to an OSPFInterfaceSetting and is required when creating.

Create a basic OSPFArea with just area id:

```
OSPFArea.create(name='myarea', area_id=0)
```

Create an OSPFArea and use a custom OSPFInterfaceSetting element:

```
OSPFArea.create(
    name='customOSPFArea',
    interface_settings_ref=OSPFInterfaceSetting('myospf'),
    area_id=3)
```

#### Advanced example:

Adding ospf\_virtual\_links\_endpoints:

```
OSPFArea.create(
    name='ospf',
    interface_settings_ref=intf,
    area_id=3,
    ospfv2_virtual_links_endpoints_container=[
        {'interface_settings_ref':
            'http://172.18.1.150:8082/6.1/elements/ospfv2_interface_settings/8',
            'router_id_endpoint_A': '192.168.1.1',
            'router_id_endpoint_B': '192.168.1.254'},
        {'router_id_endpoint_A': '172.18.1.254',
            'router_id_endpoint_B': '172.18.1.200'}]])
```

When using ABR substitute rules, there are 3 actions, 'aggregate', 'not\_advertise' and 'substitute\_with'. All references required are of type *smc.elements.network.Network*. These elements can either be created or retrieved using collections, or by getting the resource directly.

Example of creating an OSPF area and using ABR settings:

```
OSPFArea.create(
    name='area_with_abr',
    interface_settings_ref=intf,
    area_id=1,
    ospf_abr_substitute_container=[
        {'subnet_ref': 'http://172.18.1.150:8082/6.1/elements/network/143',
            'substitute_ref': 'http://172.18.1.150:8082/6.1/elements/network/1547',
            'substitute_type': 'substitute_with'},
        {'subnet_ref': 'http://172.18.1.150:8082/6.1/elements/network/979',
            'substitute_type': 'aggregate'}]])
```

#### Variables

- **interface\_settings\_ref** (*OSPFInterfaceSetting*) – reference to the OSPFInterfaceSetting

- **inbound\_filters** (*list* (IPPrefixList, IPAccessList)) – Inbound filters attached to this OSPF Area.
- **outbound\_filters** (*list* (IPPrefixList, IPAccessList)) – Outbound filter attached to this OSPF Area.

```
classmethod create(name, interface_settings_ref=None, area_id=1, area_type='normal',
                    outbound_filters=None, inbound_filters=None, shortcut_capable_area=False,
                    ospfv2_virtual_links_endpoints_container=None,
                    ospf_abr_substitute_container=None, comment=None, **kwargs)
```

Create a new OSPF Area

#### Parameters

- **name** (*str*) – name of OSPFArea configuration
- **interface\_settings\_ref** (*str*, OSPFInterfaceSetting) – an OSPFInterfaceSetting element or href. If None, uses the default system profile
- **name** – area id
- **area\_type** (*str*) – [normal|stub|not\_so\_stubby|totally\_stubby|totally\_not\_so\_stubby]
- **outbound\_filters** (*list*) – reference to an IPAccessList and or IPPrefixList. You can only have one outbound prefix or access list
- **inbound\_filters** (*list*) – reference to an IPAccessList and or IPPrefixList. You can only have one outbound prefix or access list
- **shortcut\_capable\_area** – True|False
- **ospfv2\_virtual\_links\_endpoints\_container** (*list*) – virtual link endpoints
- **ospf\_abr\_substitute\_container** (*list*) – substitute types: [aggregate|not\_advertise|substitute\_with]
- **comment** (*str*) – optional comment

#### Raises

[\*CreateElementFailed\*](#) – failed to create with reason

#### Return type

[\*OSPFArea\*](#)

```
classmethod update_or_create(with_status=False, **kwargs)
```

Update or create the element. If the element exists, update it using the kwargs provided if the provided kwargs after resolving differences from existing values. When comparing values, strings and ints are compared directly. If a list is provided and is a list of strings, it will be compared and updated if different. If the list contains unhashable elements, it is skipped. To handle complex comparisons, override this method on the subclass and process the comparison separately. If an element does not have a *create* classmethod, then it is considered read-only and the request will be redirected to *get()*. Provide a *filter\_key* dict key/value if you want to match the element by a specific attribute and value. If no *filter\_key* is provided, the name field will be used to find the element.

```
>>> host = Host('kali')
>>> print(host.address)
12.12.12.12
>>> host = Host.update_or_create(name='kali', address='10.10.10.10')
>>> print(host, host.address)
Host(name=kali) 10.10.10.10
```

**Parameters**

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements create method.
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

**Raises**

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

**Returns**

element instance by type

**Return type**

*Element*

### 14.8.5.2 OSPFKeyChain

**class** smc.routing.ospf.OSPFKeyChain(*name=None, \*\*meta*)

Bases: *Element*

OSPF Key Chain is used for authenticating OSPFv2 packets. If required, create a key chain and specify authentication in the OSPFInterfaceSetting referencing this element.

If message-digest authentication is required on an OSPFInterfaceSetting, first create the key chain and use the reference to create the ospf interface profile:

```
key_chain = OSPFKeyChain('secure-keychain') #obtain resource
OSPFInterfaceSetting.create(
    name='authenticated-ospf',
    authentication_type='message_digest',
    key_chain_ref=key_chain.href)
```

**classmethod** create(*name, key\_chain\_entry*)

Create a key chain with list of keys

Key\_chain\_entry format is:

```
[{'key': 'xxxx', 'key_id': 1-255, 'send_key': True|False}]
```

**Parameters**

- **name** (*str*) – Name of key chain
- **key\_chain\_entry** (*list*) – list of key chain entries

**Raises**

*CreateElementFailed* – create failed with reason

**Returns**

instance with meta

**Return type**

*OSPFKeyChain*

### 14.8.5.3 OSPFProfile

**class** `smc.routing.ospf.OSPFProfile(name=None, **meta)`

Bases: [Element](#)

An OSPF Profile contains administrative distance and redistribution settings. An OSPF Profile is set on the engine element when enabling OSPF.

These settings are always in effect:

- No autosummary

Example of creating an OSPFProfile with the default domain profile:

```
OSPFProfile.create(name='myospf')
```

---

**Note:** Enable OSPF on engine using `engine.ospf.enable()`

---

#### Variables

- **external\_distance** (*int*) – external distance metric
- **inter\_distance** (*int*) – inter distance metric
- **intra\_distance** (*int*) – intra distance metric
- **default\_metric** (*int*) – set a default metric for all unset areas
- **redistribution\_entry** (*list*) – settings for static, connected, etc
- **domain\_settings\_ref** ([OSPFDomainSetting](#)) – OSPF Domain Settings profile used for this OSPF Profile

**classmethod** `create(name, domain_settings_ref=None, external_distance=110, inter_distance=110, intra_distance=110, redistribution_entry=None, default_metric=None, comment=None)`

Create an OSPF Profile.

If providing a list of redistribution entries, provide in the following dict format:

```
{'enabled': boolean, 'metric_type': 'external_1' or 'external_2',
 'metric': 2, 'type': 'kernel'}
```

Valid types for redistribution entries are: kernel, static, connected, bgp, and default\_originate.

You can also provide a 'filter' key with either an IPAccessList or RouteMap element to use for further access control on the redistributed route type. If metric\_type is not provided, external\_1 (E1) will be used.

An example of a redistribution\_entry would be:

```
{u'enabled': True,
 u'metric': 123,
 u'metric_type': u'external_2',
 u'filter': RouteMap('myroutemap'),
 u'type': u'static'}
```

#### Parameters

- **name** (*str*) – name of profile

- **domain\_settings\_ref** (*str*, *OSPFDomainSetting*) – OSPFDomainSetting element or href
- **external\_distance** (*int*) – route metric (E1-E2)
- **inter\_distance** (*int*) – routes learned from different areas (O IA)
- **intra\_distance** (*int*) – routes learned from same area (O)
- **redistribution\_entry** (*list*) – how to redistribute the OSPF routes.

**Raises**

*CreateElementFailed* – create failed with reason

**Return type**

*OSPFPProfile*

**classmethod** **update\_or\_create**(*filter\_key=None*, *with\_status=False*, *\*\*kwargs*)

Update or create the element. If the element exists, update it using the kwargs provided if the provided kwargs after resolving differences from existing values. When comparing values, strings and ints are compared directly. If a list is provided and is a list of strings, it will be compared and updated if different. If the list contains unhashable elements, it is skipped. To handle complex comparisons, override this method on the subclass and process the comparison separately. If an element does not have a *create* classmethod, then it is considered read-only and the request will be redirected to *get()*. Provide a *filter\_key* dict key/value if you want to match the element by a specific attribute and value. If no *filter\_key* is provided, the name field will be used to find the element.

```
>>> host = Host('kali')
>>> print(host.address)
12.12.12.12
>>> host = Host.update_or_create(name='kali', address='10.10.10.10')
>>> print(host, host.address)
Host(name=kali) 10.10.10.10
```

**Parameters**

- **filter\_key** (*dict*) – filter key represents the data attribute and value to use to find the element. If none is provided, the name field will be used.
- **kwargs** – keyword arguments mapping to the elements *create* method.
- **with\_status** (*bool*) – if set to True, a 3-tuple is returned with (Element, modified, created), where the second and third tuple items are booleans indicating the status

**Raises**

- *CreateElementFailed* – could not create element with reason
- *ElementNotFound* – if read-only element does not exist

**Returns**

element instance by type

**Return type**

*Element*

#### 14.8.5.4 OSPFDomainSetting

**class** `smc.routing.ospf.OSPFDomainSetting(name=None, **meta)`

Bases: *Element*

An OSPF Domain Setting provides settings for area border router (ABR) type, throttle timer settings, and the max metric router link-state advertisement (LSA) settings.

An OSPF Profile requires a reference to an OSPF Domain Setting.

Create a custom OSPF Domain Setting element:

```
OSPFDomainSetting.create(
    name='mydomain',
    abr_type='standard',
    auto_cost_bandwidth=200,
    deprecated_algorithm=True)
```

```
classmethod create(name, abr_type='cisco', auto_cost_bandwidth=100, deprecated_algorithm=False,
                    initial_delay=200, initial_hold_time=1000, max_hold_time=10000,
                    shutdown_max_metric_lsa=0, startup_max_metric_lsa=0)
```

Create custom Domain Settings

Domain settings are referenced by an OSPFProfile

##### Parameters

- **name** (*str*) – name of custom domain settings
- **abr\_type** (*str*) – cisco|shortcut|standard
- **auto\_cost\_bandwidth** (*int*) – Mbits/s
- **deprecated\_algorithm** (*bool*) – RFC 1518 compatibility
- **initial\_delay** (*int*) – in milliseconds
- **initial\_hold\_type** (*int*) – in milliseconds
- **max\_hold\_time** (*int*) – in milliseconds
- **shutdown\_max\_metric\_lsa** (*int*) – in seconds
- **startup\_max\_metric\_lsa** (*int*) – in seconds

##### Raises

*CreateElementFailed* – create failed with reason

##### Returns

instance with meta

##### Return type

*OSPFDomainSetting*

#### 14.8.5.5 OSPFInterfaceSetting

**class** `smc.routing.ospf.OSPFInterfaceSetting(name=None, **meta)`

Bases: *Element*

OSPF Interface Setting indicate specific configurations that are applied to the interface and OSPF Area configuration, including authentication.

If you require non-default settings applied to your interface OSPF instance, you can create a custom interface profile:

```
OSPFInterfaceSetting.create(  
    name='myprofile',  
    dead_interval=30,  
    hello_interval=5)
```

When using authentication on interface settings, there are two types, password authentication (plain text) or message digest.

When specifying an `authentication_type='password'`, the `password` parameter must be provided.

When specifying `authentication_type='message_digest'`, the `key_chain_ref` parameter must be specified.

```
classmethod create(name, dead_interval=40, hello_interval=10, hello_interval_type='normal',  
                    dead_multiplier=1, mtu_mismatch_detection=True, retransmit_interval=5,  
                    router_priority=1, transmit_delay=1, authentication_type=None, password=None,  
                    key_chain_ref=None)
```

Create custom OSPF interface settings profile

##### Parameters

- **name** (*str*) – name of interface settings
- **dead\_interval** (*int*) – in seconds
- **hello\_interval** (*str*) – in seconds
- **hello\_interval\_type** (*str*) – [normal|fast\_hello]
- **dead\_multiplier** (*int*) – fast hello packet multiplier
- **mtu\_mismatch\_detection** (*bool*) – True|False
- **retransmit\_interval** (*int*) – in seconds
- **router\_priority** (*int*) – set priority
- **transmit\_delay** (*int*) – in seconds
- **authentication\_type** (*str*) – [password|message\_digest]
- **password** (*str*) – max 8 chars (required when `authentication_type='password'`)
- **key\_chain\_ref** (*str, Element*) – OSPFKeyChain (required when `authentication_type='message_digest'`)

##### Raises

*CreateElementFailed* – create failed with reason

##### Returns

instance with meta

##### Return type

*OSPFInterfaceSetting*

## 14.9 Policies

Policy module represents the classes required to obtaining and manipulating policies within the SMC.

Policy is the top level base class for all policy subclasses such as `smc.policy.layer3.FirewallPolicy`, `smc.policy.layer2.Layer2Policy`, `smc.policy.ips.IPSPolicy`, `smc.policy.inspection.InspectionPolicy`, `smc.policy.file_filtering.FileFilteringPolicy`

Policy represents actions that are common to all policy types, however for options that are not possible in a policy type, the method is overridden to return None. For example, 'upload' is not called on a template policy, but instead on the policy referencing that template. Therefore 'upload' is overridden.

---

**Note:** It is not required to call `open()` and `save()` on SMC API  $\geq 6.1$ . It is also optional on earlier versions but if longer running operations are needed, calling `open()` will lock the policy from test\_external modifications until `save()` is called.

---

```
class smc.policy.policy.Policy(name=None, **meta)
```

Bases: `Element`

Policy is the base class for all policy types managed by the SMC. This base class is not intended to be instantiated directly.

Subclasses should implement `create(...)` individually as each subclass will likely have different input requirements.

All generic methods that are policy level, such as 'open', 'save', 'force\_unlock', 'export', and 'upload' are encapsulated into this base class.

### Variables

- **template** (`Element`) – The template associated with this policy. Can be None
- **inspection\_policy** (`InspectionPolicy`) – related inspection policy
- **file\_filtering\_policy** (`FileFilteringPolicy`) – related file policy

### force\_unlock()

Forcibly unlock a locked policy

### Returns

None

```
rule_counters(engine=None, duration_type='one_week', duration=0, start_time=0)
```

New in version 0.5.6: Obtain rule counters for this policy. Requires SMC  $\geq 6.2$

Rule counters can be obtained for a given policy and duration for those counters can be provided in duration\_type. A custom start range can also be provided.

### Parameters

- **engine** (`Engine`) – the target engine to obtain rule counters from
- **duration\_type** (`str`) – duration for obtaining rule counters. Valid options are: `one_day`, `one_week`, `one_month`, `six_months`, `one_year`, `custom`, `since_last_upload`
- **duration** (`int`) – if custom set for duration type, specify the duration in seconds (Default: 0)
- **start\_time** (`int`) – start time in milliseconds (Default: 0)

**Raises**

ActionCommandFailed

**Returns**

list of rule counter objects

**Return type**

RuleCounter

**search\_rule**(*search*)

Search a rule for a rule tag or name value Result will be the meta data for rule (name, href, type)

Searching for a rule in specific policy:

```
f = FirewallPolicy(policy)
search = f.search_rule(searchable)
```

**Parameters****search** (*str*) – search string**Returns**

rule elements matching criteria

**Return type***list*(*Element*)

**upload**(*engine*, *timeout=5*, *wait\_for\_finish=False*, *preserve\_connections=True*, *generate\_snapshot=True*, *\*\*kw*)

Upload policy to specific device. Using wait for finish returns a poller thread for monitoring progress:

```
policy = FirewallPolicy('_NSX_Master_Default')
poller = policy.upload('myfirewall', wait_for_finish=True)
while not poller.done():
    poller.wait(3)
    print(poller.task.progress)
print("Task finished: %s" % poller.message())
```

**Parameters**

- **engine** (*str*) – name of device to upload policy to
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises**

TaskRunFailed

**Returns**

TaskOperationPoller

### 14.9.1 InterfacePolicy

Interface Policies are applied at the engine level when layer 3 single engines or cluster layer 3 engines have layer 2 interfaces. The configuration is identical to creating Layer 2 Rules for layer 2 or IPS engines.

**class** `smc.policy.interface.InterfacePolicy`(*name=None, \*\*meta*)

Bases: [\*InterfaceRule\*](#), [\*Policy\*](#)

Layer 2 Interface Policy represents a set of rules applied to layer 2 interfaces installed on a single or cluster layer 3 engine. Set the interface policy on the engine properties. Interface policies do not have inspection policies and instead inherit from the engines primary policy.

Instance Resources:

#### Variables

- **layer2\_ipv4\_access\_rules** – layer2\_ipv4\_access\_rules
- **layer2\_ipv6\_access\_rules** – layer2\_ipv6\_access\_rules
- **layer2\_ethernet\_rules** – layer2\_ethernet\_rules

**classmethod** `create`(*name, template*)

Create a new Layer 2 Interface Policy.

#### Parameters

- **name** (*str*) – name of policy
- **template** (*str*) – name of the NGFW Engine template to base policy on

#### Raises

- [\*LoadPolicyFailed\*](#) – cannot find policy by name
- [\*CreatePolicyFailed\*](#) – cannot create policy with reason

#### Returns

Layer2InterfacePolicy

**inspection\_policy**()

Descriptor to allow get/set operations on an element referenced in an Element.

May be defined with a supported version. ex.

```
monitoring_group = ElementRef(('6.5','monitoring_group_ref'),
                              ('6.6','tunnel_group_ref'))
```

**class** `smc.policy.interface.InterfaceRule`

Bases: [\*object\*](#)

Layer 2 Interface Rules are the same as Layer 2 Engine/IPS rules.

**property** `layer2_ethernet_rules`

Layer 2 Ethernet access rule

#### Return type

rule\_collection([\*EthernetRule\*](#))

**property** `layer2_ipv4_access_rules`

Layer2 IPv4 access rule

#### Return type

rule\_collection([\*IPv4Layer2Rule\*](#))

**property layer2\_ipv6\_access\_rules**

Layer 2 IPv6 access rule

```
class smc.policy.interface.InterfaceSubPolicy(name=None, **meta)
```

Bases: *Policy*

A Interface Sub Policy is a rule section within a Interface policy that provides a container to create rules that are referenced from a ‘jump’ rule. Typically rules in a sub policy are similar in some fashion such as applying to a specific service. Sub Policies can also be delegated from an administrative perspective.

Interface Sub Policies only provide access to creating IPv4 rules.

```
p = InterfaceSubPolicy('MySubPolicy') p.layer2_ethernet_rules.create(
    name='newule', sources='any', destinations='any', services=[TCPService('SSH')], ac-
    tion='discard')
```

```
classmethod create(name)
```

Create a sub policy. Only name is required. Other settings are inherited from the parent firewall policy (template, policy, etc).

**Parameters**

**name** (*str*) – name of sub policy

**Raises**

*CreateElementFailed* – failed to create policy

**Return type**

*InterfaceSubPolicy*

**property layer2\_ipv4\_access\_rules**

IPv4 rule entry point

**Return type**

rule\_collection(*EthernetRule*)

```
class smc.policy.interface.InterfaceTemplatePolicy(name=None, **meta)
```

Bases: *InterfacePolicy*

Interface Template Policy. Required when creating a new Interface Policy. Useful for containing global rules or best practice configurations which will be inherited by the assigned policy.

```
print(list(InterfaceTemplatePolicy.objects.all()))
```

**inspection\_policy()**

Descriptor to allow get/set operations on an element referenced in an Element.

May be defined with a supported version. ex.

```
monitoring_group = ElementRef(('6.5','monitoring_group_ref'),
    ('6.6','tunnel_group_ref'))
```

**upload()**

Upload policy to specific device. Using wait for finish returns a poller thread for monitoring progress:

```
policy = FirewallPolicy('_NSX_Master_Default')
poller = policy.upload('myfirewall', wait_for_finish=True)
while not poller.done():
    poller.wait(3)
    print(poller.task.progress)
print("Task finished: %s" % poller.message())
```

**Parameters**

- **engine** (*str*) – name of device to upload policy to
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises**

TaskRunFailed

**Returns**

TaskOperationPoller

## 14.9.2 FileFilteringPolicy

```
class smc.policy.file_filtering.FileFilteringPolicy(name=None, **meta)
```

Bases: *Policy*

The File Filtering Policy references a specific file based policy for doing additional inspection based on file types. Use the policy parameters to specify how certain files are treated by either threat intelligence feeds, sandbox or by local AV scanning. You can also use this policy to disable threat prevention based on specific files.

**classmethod create**(*name*, *comment=None*)

Create the custom File Filtering Policy :param str name: name of File Filtering Policy :param str comment: optional comment :raises CreateElementFailed: failed creating element with reason :return: instance with meta of File Filtering Policy. :rtype: FileFilteringPolicy

**property file\_filtering\_rules**

File filtering rules for this policy.

**Return type**

rule\_collection(*FileFilteringRule*)

```
class smc.policy.file_filtering.FileFilteringRule(**meta)
```

Bases: RuleCommon, *Rule*, *SubElement*

Represents a file filtering rule.

**create**(*name*, *sources=None*, *destinations=None*, *action='allow'*, *log\_options=None*, *connection\_tracking=None*, *is\_disabled=False*, *situations=None*, *add\_pos=None*, *after=None*, *before=None*, *comment=None*, *validate=True*, *\*\*kw*)

Create a file filtering rule.

Changed in version 0.7.0: Action field now requires a list of actions as strings when using API version >= 6.6

**Example::**

Api version <=6.5 action is a string rule\_file = p.file\_filtering\_rules.create( name="newrule",

```
sources=[Network("London Internal Network")], destinations=[Network("net-172.31.14.0/24")], action="apply_vpn", vpn_policy=vpn)
```

Api version >=6.6 action is a list vpn\_actions = Action() vpn\_actions.action = ['allow', 'apply\_vpn']  
p.file\_filtering\_rules.create(name='new\_rule',

```
sources=[Network("London Internal Network")], destinations=[Network("net-172.31.14.0/24")], action=vpn_actions, vpn_policy=vpn)
```

**Parameters**

- **name** (*str*) – name of rule
- **sources** (*Source*, *list*[*str*, *Element*]) – source/s for rule
- **destinations** (*Destination*, *list*[*str*, *Element*]) – destination/s for rule
- **action** (*Action*, *str*, *list*[*str*]) – allow, discard, continue, refuse, jump, apply\_vpn, enforce\_vpn, forward\_vpn, block\_list, terminate, forward, next\_hop, forced\_next\_hop (default: allow)
- **log\_options** (*LogOptions*) – LogOptions object
- **connection\_tracking** (*ConnectionTracking*) – custom connection tracking settings
- **is\_disabled** (*bool*) – whether to disable rule or not
- **situations** (*Situations*, *str*, *dict*) – A set of matching criteria that defines the file types the rule matches. If nothing is specified, that means None and the Rule will be ignored.
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with after and before params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with add\_pos and before params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with add\_pos and after params.
- **comment** (*str*) – optional comment for this rule
- **validate** (*bool*) – validate the inspection policy during rule creation. Default: True

**Raises**

- *MissingRequiredInput* – when options are specified the need additional setting, i.e. use\_vpn action requires a vpn policy be specified.
- *CreateRuleFailed* – rule creation failure

**Returns**

the created FileFilteringRule

**Return type**

*FileFilteringRule*

**get\_action()**

Return action instance. rtype: FileFilteringRuleAction

**property situations**

Situations for this rule

**Return type**

Situations

### 14.9.3 FirewallPolicy

Layer 3 Firewall Policy

Module that represents resources related to creating and managing layer 3 firewall engine policies.

To get an existing policy:

```
>>> from smc.policy.layer3 import FirewallPolicy
>>> policy = FirewallPolicy('Standard Firewall Policy with Inspection')
>>> print(policy.template)
FirewallTemplatePolicy(name=Firewall Inspection Template)
```

Or through collections:

```
>>> list(FirewallPolicy.objects.all())
[FirewallPolicy(name=Standard Firewall Policy with Inspection),
 FirewallPolicy(name=Layer 3 Virtual FW Policy)]
```

To create a new policy, use:

```
policy = FirewallPolicy.create(name='newpolicy', template='layer3_fw_template')
```

Example rule creation:

```
policy = FirewallPolicy('Amazon Cloud')
policy.open() #Only required for SMC API <= 6.0
policy.fw_ipv4_access_rules.create(name='mynewrule', sources='any',
                                   destinations='any', services='any',
                                   action='permit')
policy.save() #Only required for SMC API <= 6.0
```

Example rule deletion:

```
policy = FirewallPolicy('Amazon Cloud')
for rule in policy.fw_ipv4_access_rules.all():
    if rule.name == 'mynewrule':
        rule.delete()
```

```
class smc.policy.layer3.FirewallIPv6SubPolicy(name=None, **meta)
```

Bases: *FirewallSubPolicy*

**property fw\_ipv6\_access\_rules**

IPv6 rule entry point

**Return type**

rule\_collection(*IPv4Rule*)

```
class smc.policy.layer3.FirewallPolicy(name=None, **meta)
```

Bases: *FirewallRule, Policy*

FirewallPolicy represents a set of rules installed on layer 3 devices. Layer 3 engine's support either ipv4 or ipv6 rules.

They also have NAT rules and reference to an Inspection and File Filtering Policy.

**Variables**

**template** – which policy template is used

Instance Resources:

#### Variables

- **fw\_ipv4\_access\_rules** – *fw\_ipv4\_access\_rules*
- **fw\_ipv4\_nat\_rules** – *ipv4\_nat\_rules*
- **fw\_ipv6\_access\_rules** – *ipv6\_access\_rules*
- **fw\_ipv6\_nat\_rules** – *ipv6\_nat\_rules*

**classmethod** **create**(*name*, *template*='Firewall Inspection Template')

Create Firewall Policy. Template policy is required for the policy. The template parameter should be the name of the firewall template.

This policy will then inherit the Inspection and File Filtering policy from the specified template.

#### Parameters

- **name** (*str*) – name of policy
- **template** (*str*) – name of the NGFW engine template to base policy on

#### Raises

- *LoadPolicyFailed* – Cannot load the policy after creation
- *CreatePolicyFailed* – policy creation failed with message

#### Returns

FirewallPolicy

To use after successful creation, reference the policy to obtain context:

```
FirewallPolicy('newpolicy')
```

**update**(*cautious\_update*=True, *\*\*kwargs*)

Update Firewall Policy. By default this will load the etag from the API. This is to handle cases where a subelement has changed the etag of the policy. If the policy is updated prior to these additions then *cautious\_update* can be turned off.

#### Cautious\_update

True to load etag from API before updating.

**class** `smc.policy.layer3.FirewallRule`

Bases: `object`

Encapsulates all references to firewall rule related entry points. This is referenced by multiple classes such as FirewallPolicy and FirewallPolicyTemplate.

**property** **fw\_ipv4\_access\_rules**

IPv4 rule entry point

#### Return type

*rule\_collection(IPv4Rule)*

**property** **fw\_ipv4\_nat\_rules**

IPv4NAT Rule entry point

#### Return type

*rule\_collection(IPv4NATRule)*

**property fw\_ipv6\_access\_rules**

IPv6 Rule entry point

**Return type**rule\_collection(*IPv6Rule*)**property fw\_ipv6\_nat\_rules**

IPv6NAT Rule entry point

**Return type**rule\_collection(*IPv6NATRule*)**class** smc.policy.layer3.FirewallSubPolicy(*name=None, \*\*meta*)Bases: *Policy*

A Firewall Sub Policy is a rule section within a firewall policy that provides a container to create rules that are referenced from a ‘jump’ rule. Typically rules in a sub policy are similar in some fashion such as applying to a specific service. Sub Policies can also be delegated from an administrative perspective.

Firewall Sub Policies only provide access to creating IPv4 rules. NAT is done on the parent firewall policy:

```
p = FirewallSubPolicy('MySubPolicy')
p.fw_ipv4_access_rules.create(
    name='newule',
    sources='any',
    destinations='any',
    services=[TCPService('SSH')],
    action='discard')
```

**classmethod create(*name*)**

Create a sub policy. Only name is required. Other settings are inherited from the parent firewall policy (template, inspection policy, etc).

**Parameters****name** (*str*) – name of sub policy**Raises***CreateElementFailed* – failed to create policy**Return type***FirewallSubPolicy***property fw\_ipv4\_access\_rules**

IPv4 rule entry point

**Return type**rule\_collection(*IPv4Rule*)**class** smc.policy.layer3.FirewallTemplatePolicy(*name=None, \*\*meta*)Bases: *FirewallPolicy*

All Firewall Policies will reference a firewall policy template.

Most templates will be pre-configured best practice configurations and rarely need to be modified. However, you may want to view the details of rules configured in a template or possibly insert additional rules.

For example, view rules in firewall policy template after loading the firewall policy:

```
policy = FirewallPolicy('Amazon Cloud')
for rule in policy.template.fw_ipv4_access_rules.all():
    print rule
```

### upload()

Upload policy to specific device. Using wait for finish returns a poller thread for monitoring progress:

```
policy = FirewallPolicy('_NSX_Master_Default')
poller = policy.upload('myfirewall', wait_for_finish=True)
while not poller.done():
    poller.wait(3)
    print(poller.task.progress)
print("Task finished: %s" % poller.message())
```

#### Parameters

- **engine** (*str*) – name of device to upload policy to
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

#### Raises

`TaskRunFailed`

#### Returns

`TaskOperationPoller`

## 14.9.4 InspectionPolicy

```
class smc.policy.policy.InspectionPolicy(name=None, **meta)
```

Bases: *Policy*

The Inspection Policy references a specific inspection policy that is a property (reference) to either a FirewallPolicy, IPSPolicy or Layer2Policy. This policy defines specific characteristics for threat based prevention. In addition, exceptions can be made at this policy level to bypass scanning based on the rule properties.

### export()

Export this element.

Usage:

```
engine = Engine('myfirewall')
extask = engine.export(filename='fooexport.zip')
while not extask.done():
    extask.wait(3)
print("Finished download task: %s" % extask.message())
print("File downloaded to: %s" % extask.filename)
```

#### Parameters

**filename** (*str*) – filename to store exported element

#### Raises

***TaskRunFailed*** – invalid permissions, invalid directory, or this element is a system element and cannot be exported.

**Returns**

DownloadTask

---

**Note:** It is not possible to export system elements

---

**upload()**

Upload policy to specific device. Using wait for finish returns a poller thread for monitoring progress:

```
policy = FirewallPolicy('_NSX_Master_Default')
poller = policy.upload('myfirewall', wait_for_finish=True)
while not poller.done():
    poller.wait(3)
    print(poller.task.progress)
print("Task finished: %s" % poller.message())
```

**Parameters**

- **engine** (*str*) – name of device to upload policy to
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises**

TaskRunFailed

**Returns**

TaskOperationPoller

## 14.9.5 IPSPolicy

IPS Engine policy

Module that represents resources related to creating and managing IPS engine policies.

To get an existing policy:

```
>>> policy = IPSPolicy('Default IPS Policy')
>>> print(policy.template)
IPSTemplatePolicy(name=High-Security IPS Template)
```

Or through collections:

```
>>> from smc.policy.ips import IPSPolicy
>>> list(IPSPolicy.objects.all())
[IPSPolicy(name=Default IPS Policy), IPSPolicy(name=High-Security Inspection IPS Policy)]
```

To create a new policy, use:

```
policy = IPSPolicy.create(name='my_ips_policy',
                          template='High Security Inspection Template')
policy.ips_ipv4_access_rules.create(name='ipsrule1',
                                    sources='any',
                                    action='continue')
```

(continues on next page)

(continued from previous page)

```
for rule in policy.ips_ipv4_access_rules.all():
    print(rule)
```

Example rule deletion:

```
policy = IPSPolicy('Amazon Cloud')
for rule in policy.ips_ipv4_access_rules.all():
    if rule.name == 'ipsrule1':
        rule.delete()
```

**class** `smc.policy.ips.IPSPolicy`(*name=None, \*\*meta*)

Bases: [\*IPSPolicyRule\*](#), [\*Policy\*](#)

IPS Policy represents a set of rules installed on an IPS / IDS engine. IPS mode supports both inline and SPAN interface types and ethernet based rules. Layer 2 and IPS engines do not current features that require routed interfaces.

**Variables**

**template** – which policy template is used

Instance Resources:

**Variables**

- **ips\_ipv4\_access\_rules** – `ips_ipv4_access_rules`
- **ips\_ipv6\_access\_rules** – `ips_ipv6_access_rules`
- **ips\_ethernet\_rules** – `ips_ethernet_rules`

**classmethod** `create`(*name, template='High-Security IPS Template'*)

Create an IPS Policy

**Parameters**

- **name** (*str*) – Name of policy
- **template** (*str*) – name of template

**Raises**

[\*CreatePolicyFailed\*](#) – policy failed to create

**Returns**

`IPSPolicy`

**class** `smc.policy.ips.IPSPolicyRule`

Bases: [\*object\*](#)

Encapsulates all references to IPS rule related entry points. This is referenced by multiple classes such as IPSPolicy and IPSPolicyTemplate.

**property** `ips_ethernet_rules`

IPS Ethernet access rule

**Return type**

`rule_collection`([\*EthernetRule\*](#))

**property** `ips_ipv4_access_rules`

IPS ipv4 access rules

**Return type**rule\_collection(*IPv4Layer2Rule*)**class** smc.policy.ips.IPSSubPolicy(*name=None, \*\*meta*)Bases: *Policy*

A IPS Sub Policy is a rule section within an IPS policy that provides a container to create rules that are referenced from a ‘jump’ rule. Typically rules in a sub policy are similar in some fashion such as applying to a specific service. Sub Policies can also be delegated from an administrative perspective.

```
p = IPSSubPolicy('MyIPSSubPolicy') p.fw_ipv4_access_rules.create(
    name='newrule', sources='any', destinations='any', services=[TCPService('SSH')], ac-
    tion='discard')
```

**classmethod** create(*name*)

Create a sub policy. Only name is required. Other settings are inherited from the parent IPS policy (template, inspection policy, etc).

**Parameters****name** (*str*) – name of sub policy**Raises***CreateElementFailed* – failed to create policy**Return type***IPSSubPolicy***property** ips\_ipv4\_access\_rules

IPv4 rule entry point

**Return type**rule\_collection(*IPSRule*)**class** smc.policy.ips.IPSTemplatePolicy(*name=None, \*\*meta*)Bases: *IPSPolicy*

All IPS Policies will reference an IPS policy template.

Most templates will be pre-configured best practice configurations and rarely need to be modified. However, you may want to view the details of rules configured in a template or possibly insert additional rules.

For example, view rules in an ips policy template after loading the ips policy:

```
policy = IPSPolicy('InlineIPS')
for rule in policy.template.ips_ipv4_access_rules.all():
    print(rule)
```

**upload()**

Upload policy to specific device. Using wait for finish returns a poller thread for monitoring progress:

```
policy = FirewallPolicy('_NSX_Master_Default')
poller = policy.upload('myfirewall', wait_for_finish=True)
while not poller.done():
    poller.wait(3)
    print(poller.task.progress)
print("Task finished: %s" % poller.message())
```

**Parameters**

- **engine** (*str*) – name of device to upload policy to
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises**

TaskRunFailed

**Returns**

TaskOperationPoller

## 14.9.6 Layer2Policy

Layer 2 Firewall Policy

Module that represents resources related to creating and managing layer 2 firewall engine policies.

To get an existing policy:

```
>>> from smc.policy.layer2 import Layer2Policy
>>> policy = Layer2Policy('MyLayer2Policy')
>>> print(policy.template)
Layer2TemplatePolicy(name=Layer 2 Firewall Inspection Template)
```

Or through collections:

```
>>> from smc.policy.layer2 import Layer2Policy
>>> list(Layer2Policy.objects.all())
[Layer2Policy(name=MyLayer2Policy)]
```

To create a new policy, use:

```
policy = Layer2Policy.create(name='newpolicy', template='layer2_fw_template')
```

Example rule creation:

```
policy = Layer2Policy('smcpython-l2')

policy.layer2_ipv4_access_rules.create(
    name='nonerule',
    sources='any',
    destinations='any',
    services='any',
    logical_interfaces=[location_href_to_logical_interface])
```

Create Ethernet rule for layer 2 firewall:

```
policy.layer2_ethernet_rules.create(name='nonerule',
    sources='any',
    destinations='any',
    services='any')
```

---

**Note:** Leaving parameter `logical_interfaces` out of `create` will default to 'ANY'.

---

Example rule deletion:

```
policy = Layer2Policy('Amazon Cloud')
for rule in policy.layer2_ipv4_access_rules.all():
    if rule.name == 'myrule':
        print rule.delete()
```

**class** smc.policy.layer2.Layer2Policy(*name=None, \*\*meta*)

Bases: [Layer2Rule](#), [Policy](#)

Layer 2 Policy represents a set of rules installed on a layer 2 firewall engine. Layer 2 mode supports both inline and SPAN interface types and ethernet based rules. Layer 2 and IPS engines do not current features that require routed interfaces.

#### Variables

**template** – which policy template is used

Instance Resources:

#### Variables

- **layer2\_ipv4\_access\_rules** – [layer2\\_ipv4\\_access\\_rules](#)
- **layer2\_ipv6\_access\_rules** – [layer2\\_ipv6\\_access\\_rules](#)
- **layer2\_ethernet\_rules** – [layer2\\_ethernet\\_rules](#)

**classmethod** **create**(*name, template='Layer 2 Firewall Inspection Template'*)

Create Layer 2 Firewall Policy. Template policy is required for the policy. The template parameter should be the name of the template.

The template should exist as a layer 2 template policy and should be referenced by name.

This policy will then inherit the Inspection and File Filtering policy from the specified template.

To use after successful creation, reference the policy to obtain context:

```
Layer2Policy('newpolicy')
```

#### Parameters

- **name** (*str*) – name of policy
- **template** (*str*) – name of the NGFW engine template to base policy on

#### Raises

- [LoadPolicyFailed](#) – cannot find policy by name
- [CreatePolicyFailed](#) – cannot create policy with reason

#### Returns

Layer2Policy

**class** smc.policy.layer2.Layer2Rule

Bases: [object](#)

Encapsulates all references to layer 2 firewall rule related entry points. This is referenced by multiple classes such as Layer2Policy and Layer2TemplatePolicy.

**property** **layer2\_ethernet\_rules**

Layer 2 Ethernet access rule

**Return type**rule\_collection(*EthernetRule*)**property layer2\_ipv4\_access\_rules**

Layer2 Firewall access rule

**Return type**rule\_collection(*IPv4Layer2Rule*)**property layer2\_ipv6\_access\_rules**

Layer 2 IPv6 access rule

**class** smc.policy.layer2.Layer2TemplatePolicy(*name=None, \*\*meta*)Bases: *Layer2Policy*

All Layer 2 Firewall Policies will reference a firewall policy template.

Most templates will be pre-configured best practice configurations and rarely need to be modified. However, you may want to view the details of rules configured in a template or possibly insert additional rules.

For example, view rules in the layer 2 policy template after loading the firewall policy:

```
policy = Layer2Policy('Amazon Cloud')
for rule in policy.template.layer2_ipv4_access_rules.all():
    print rule
```

**upload()**

Upload policy to specific device. Using wait for finish returns a poller thread for monitoring progress:

```
policy = FirewallPolicy('_NSX_Master_Default')
poller = policy.upload('myfirewall', wait_for_finish=True)
while not poller.done():
    poller.wait(3)
    print(poller.task.progress)
print("Task finished: %s" % poller.message())
```

**Parameters**

- **engine** (*str*) – name of device to upload policy to
- **preserve\_connections** (*bool*) – flag to preserve connections (True by default)
- **generate\_snapshot** (*bool*) – flag to generate snapshot (True by default)

**Raises**

TaskRunFailed

**Returns**

TaskOperationPoller

## 14.9.7 QoSPolicy

QoS Policy that would be applied to a rule set or physical / tunnel interface. QoS can also be applied at the VLAN level of an interface.

```
class smc.policy.qos.QoSClass(name=None, **meta)
```

Bases: *Element*

This represents a QoS Class. It is an element that works as a link between a rule in a QoS Policy and one or more Firewall Actions. The traffic allowed in the access rule is assigned the QoS Class defined for the rule, and the QoS class is used as the matching criteria for applying QoS Policy rules.

```
classmethod create(name, comment=None, lsv_override=None, link_selection=None)
```

Create the QoS Class.

### Parameters

- **name** (*str*) – name of QoS Class
- **comment** (*str*) – optional comment
- **lsv\_override** (*bool*) – optional can be True or False

:param object link\_selection : optional value to override link selection value :raises CreateElementFailed:  
failed creating element with reason :return: instance with meta :rtype: QoSClass

**property** link\_selection

```
class smc.policy.qos.QoSPolicy(name=None, **meta)
```

Bases: *Element*

This represents a QoS Policy. A set of rules for Bandwidth Management and Traffic Prioritization for traffic that has a particular QoS Class, or rules for assigning QoS Classes based on a DSCP Match found in the traffic.

## 14.10 Sub Policies

Sub Policies are referenced from within a normal policy as a parameter to a ‘jump’ action. They provide rule encapsulation for similar rules and can be delegated to an Admin User for more granular policy control.

### 14.10.1 FirewallSubPolicy

```
class smc.policy.layer3.FirewallSubPolicy(name=None, **meta)
```

Bases: *Policy*

A Firewall Sub Policy is a rule section within a firewall policy that provides a container to create rules that are referenced from a ‘jump’ rule. Typically rules in a sub policy are similar in some fashion such as applying to a specific service. Sub Policies can also be delegated from an administrative perspective.

Firewall Sub Policies only provide access to creating IPv4 rules. NAT is done on the parent firewall policy:

```
p = FirewallSubPolicy('MySubPolicy')
p.fw_ipv4_access_rules.create(
    name='newule',
    sources='any',
    destinations='any',
    services=[TCPService('SSH')],
    action='discard')
```

**classmethod** `create(name)`

Create a sub policy. Only name is required. Other settings are inherited from the parent firewall policy (template, inspection policy, etc).

**Parameters**

**name** (*str*) – name of sub policy

**Raises**

*CreateElementFailed* – failed to create policy

**Return type**

*FirewallSubPolicy*

**property** `fw_ipv4_access_rules`

IPv4 rule entry point

**Return type**

*rule\_collection(IPv4Rule)*

## 14.11 Rules

Represents classes responsible for configuring rule types.

### 14.11.1 Rule

**class** `smc.policy.rule.Rule`

Bases: `object`

Top level rule construct with methods required to modify common behavior of any rule types. To retrieve a rule, access by reference:

```
policy = FirewallPolicy('mypolicy')
for rule in policy.fw_ipv4_nat_rules.all():
    print(rule.name, rule.comment, rule.is_disabled)
```

**property** `action`

Action for this rule.

**Return type**

*Action*

**property** `authentication_options`

Read only authentication options field

**Return type**

*AuthenticationOptions*

**property** `comment`

Optional comment for this rule.

**Parameters**

**value** (*str*) – string comment

**Return type**

*str*

**property destinations**

Destinations for this rule

**Return type**

*Destination*

**disable()**

Disable this rule

**enable()**

Enable this rule

**property history**

New in version 0.6.3: Requires SMC version  $\geq 6.5$

Obtain the history of this element. This will not chronicle every modification made over time, but instead a current snapshot with historical information such as when the element was created, by whom, when it was last modified and it's current state.

**Raises**

*ResourceNotFound* – If not running SMC version  $\geq 6.5$

**Return type**

*History*

**property is\_disabled**

Whether the rule is enabled or disabled

**Parameters**

**value** (*bool*) – True, False

**Return type**

*bool*

**property is\_rule\_section**

Is this rule considered a rule section

**Return type**

*bool*

**property match\_vpn\_options**

Read only match vpn options field

**Return type**

SourceVpn

**move\_rule\_after(*other\_rule*)**

Add this rule after another. This process will make a copy of the existing rule and add after the specified rule. If this raises an exception, processing is stopped. Otherwise the original rule is then deleted. You must re-retrieve the new element after running this operation as new references will be created.

**Parameters**

**Rule** (*other\_rule*) – rule where this rule will be positioned after

**Raises**

*CreateRuleFailed* – failed to duplicate this rule, no move is made

**move\_rule\_before(*other\_rule*)**

Move this rule after another. This process will make a copy of the existing rule and add after the specified rule. If this raises an exception, processing is stopped. Otherwise the original rule is then deleted. You must re-retrieve the new element after running this operation as new references will be created.

**Parameters**

**Rule** (*other\_rule*) – rule where this rule will be positioned before

**Raises**

*CreateRuleFailed* – failed to duplicate this rule, no move is made

**property name**

Name attribute of rule element

**property options**

Options for this rule.

**Return type**

*LogOptions*

**property parent\_policy**

Read-only name of the parent policy

**Returns**

*smc.base.model.Element* of type policy

**save()**

After making changes to a rule element, you must call save to apply the changes. Rule changes are made to cache before sending to SMC.

**Raises**

*PolicyCommandFailed* – failed to save with reason

**Returns**

href of this rule

**Return type**

*str*

**property services**

Services assigned to this rule

**Return type**

*Service*

**property sources**

Sources assigned to this rule

**Return type**

*Source*

**property tag**

Value of rule tag. Read only.

**Returns**

rule tag

**Return type**

*str*

**update**(*validate=True, sources=None, destinations=None, services=None, action=None, \*\*kwargs*)

update a rule

**Parameters**

- **sources** (*str, list[Element]* *str can be "any" or json*) – source/s for rule

- **destinations**(*str*, *list*[*Element*] *str* can be "any" or *json*) – destination/s for rule
- **services**(*str*, *list*[*Element*] *str* can be "any" or *json*) – service/s for rule
- **validate**(*bool*) – validate the policy before update; default True
- **action**(*str*, *list*[*str*] since API 6.6, *json*) – action/s for rule

**Returns**

href of this rule

**Return type**

*str*

**14.11.1.1 IPv4Rule**

**class** `smc.policy.rule.IPv4Rule`(\*\**meta*)

Bases: `RuleCommon`, `Rule`, `SubElement`

Represents an IPv4 Rule for a layer 3 engine.

Create a rule:

```
policy = FirewallPolicy('mypolicy')
policy.fw_ipv4_access_rules.create(name='smcpython',
                                   sources='any',
                                   destinations='any',
                                   services='any')
```

Sources and Destinations can be one of any valid network element types defined in `smc.elements.network`.

Source entries by href:

```
sources=['http://1.1.1.1:8082/elements/network/myelement',
         'http://1.1.1.1:8082/elements/host/myhost'], etc
```

Source entries using network elements:

```
sources=[Host('myhost'), Network('thenetwork'), AddressRange('range')]
```

Services have a similar syntax and can take any type of `smc.elements.service` or the element href or both:

```
services=[TCPService('myservice'),
         'http://1.1.1.1/8082/elements/tcp_service/mytcp_service',
         'http://1.1.1.1/8082/elements/udp_server/myudp_service'], etc
```

You can obtain services and href for the elements by using the `smc.base.collection` collections:

```
>>> services = list(TCPService.objects.filter('80'))
>>> for service in services:
...     print(service, service.href)
...
(TCPService(name=tcp80443), u'http://172.18.1.150:8082/6.1/elements/tcp_service/3535
↪')
(TCPService(name=HTTP to Web SaaS), u'http://172.18.1.150:8082/6.1/elements/tcp_
↪service/589')
(TCPService(name=HTTP), u'http://172.18.1.150:8082/6.1/elements/tcp_service/440')
```

Services by application (get all facebook applications):

```
>>> applications = Search.objects.entry_point('application_situation').filter(
↳ 'facebook')
>>> print(list(applications))
[ApplicationSituation(name=Facebook-Plugins-Share-Button),
 ApplicationSituation(name=Facebook-Plugins)
...]
```

Sources / Destinations and Services can also take the string value 'any' to allow all. For example:

```
sources='any'
```

```
create(name, sources=None, destinations=None, services=None, action='allow', log_options=None,
authentication_options=None, match_vpn_options=None, connection_tracking=None,
is_disabled=False, vpn_policy=None, mobile_vpn=False, add_pos=None, after=None, before=None,
sub_policy=None, comment=None, validate=True, **kw)
```

Create a layer 3 firewall rule

Changed in version 0.7.0: Action field now requires a list of actions as strings when using API version >= 6.6

#### Example::

Api version <=6.5 action is a string rule\_vpn = p.fw\_ipv4\_access\_rules.create( name="newrule\_vpn",  
sources=[Network("London Internal Network")], destinations=[Network("net-172.31.14.0/24")], services="any", action="apply\_vpn", vpn\_policy=vpn)

Api version >=6.6 action is a list vpn\_actions = Action() vpn\_actions.action = ['allow', 'apply\_vpn']  
p.fw\_ipv4\_access\_rules.create(name="",  
sources=[Network("London Internal Network")], destinations=[Network("net-172.31.14.0/24")], services='any', action=vpn\_actions, vpn\_policy=vpn)

#### Parameters

- **name** (*str*) – name of rule
- **sources** (*Source*, *list[str, Element]*) – source/s for rule
- **destinations** (*Destination*, *list[str, Element]*) – destination/s for rule
- **services** (*Service*, *list[str, Element]*) – service/s for rule
- **action** (*Action*, *str*, *list[str]*) – allow, continue, discard, refuse, enforce\_vpn, apply\_vpn, forward\_vpn, blacklist, forced\_next\_hop (default: allow)
- **log\_options** (*LogOptions*) – LogOptions object
- **connection\_tracking** (*ConnectionTracking*) – custom connection tracking settings
- **authentication\_options** (*AuthenticationOptions*) – options for auth if any
- **match\_vpn\_options** (*SourceVpn*) – rule matches traffic from specific VPNs
- **vpn\_policy** (*PolicyVPN*, *str*) – policy element or str href; required for enforce\_vpn, use\_vpn and apply\_vpn actions
- **mobile\_vpn** (*bool*) – if using a vpn action, you can set mobile\_vpn to True and omit the vpn\_policy setting if you want this VPN to apply to any mobile VPN based on the policy VPN associated with the engine

- **sub\_policy** (*str*, *Element*) – sub policy required when rule has an action of ‘jump’. Can be the FirewallSubPolicy element or href.
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with after and before params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with add\_pos and before params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with add\_pos and after params.
- **comment** (*str*) – optional comment for this rule
- **validate** (*bool*) – validate the inspection policy during rule creation. Default: True

#### Raises

- **MissingRequiredInput** – when options are specified the need additional setting, i.e. use\_vpn action requires a vpn policy be specified.
- **CreateRuleFailed** – rule creation failure

#### Returns

the created ipv4 rule

#### Return type

*IPv4Rule*

**create\_rule\_section**(*name*, *add\_pos=None*, *after=None*, *before=None*)

Create a rule section in a Firewall Policy. To specify a specific numbering position for the rule section, use the *add\_pos* field. If no position or before/after is specified, the rule section will be placed at the top which will encapsulate all rules below. Create a rule section for the relevant policy:

```
policy = FirewallPolicy('mypolicy')
policy.fw_ipv4_access_rules.create_rule_section(name='atop')
# For NAT rules
policy.fw_ipv4_nat_rules.create_rule_section(name='mysection', add_pos=5)
```

#### Parameters

- **name** (*str*) – create a rule section by name
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with after and before params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with add\_pos and before params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with add\_pos and after params.

#### Raises

- **MissingRequiredInput** – when options are specified the need additional setting, i.e. use\_vpn action requires a vpn policy be specified.
- **CreateRuleFailed** – rule creation failure

**Returns**

the created ipv4 rule

**Return type**

*IPv4Rule*

### 14.11.1.2 IPv4Layer2Rule

**class** smc.policy.rule.IPv4Layer2Rule(\*\*meta)

Bases: RuleCommon, *Rule*, *SubElement*

Create IPv4 rules for Layer 2 Firewalls

Example of creating an allow all rule:

```
policy = Layer2Policy('mylayer2')
policy.layer2_ipv4_access_rules.create(name='myrule',
                                       sources='any',
                                       destinations='any',
                                       services='any')
```

**create**(name, sources=None, destinations=None, services=None, action='allow', is\_disabled=False, logical\_interfaces=None, add\_pos=None, after=None, before=None, comment=None, validate=True, sub\_policy=None, \*\*kw)

Create an IPv4 Layer 2 Engine rule

Changed in version 0.7.0: Action field now requires a list of actions as strings when using SMC version >= 6.6.0

**Parameters**

- **name** (*str*) – name of rule
- **sources** (*list*[*str*, *Element*]) – source/s for rule
- **destinations** (*list*[*str*, *Element*]) – destination/s for rule
- **services** (*list*[*str*, *Element*]) – service/s for rule
- **action** (*str*, *Action*) – |allow|continue|discard|refuse|blacklist
- **is\_disabled** (*bool*) – whether to disable rule or not
- **logical\_interfaces** (*list*) – logical interfaces by name
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with after and before params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with add\_pos and before params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with add\_pos and after params.
- **comment** (*str*) – optional comment for this rule
- **validate** (*bool*) – validate the inspection policy during rule creation. Default: True
- **sub\_policy** (*str*, *Element*) – sub policy required when rule has an action of 'jump'. Can be the IPSSubPolicy element or href.

**Raises**

- ***MissingRequiredInput*** – when options are specified the need additional setting, i.e. use\_vpn action requires a vpn policy be specified.
- ***CreateRuleFailed*** – rule creation failure

**Returns**

newly created rule

**Return type**

*IPv4Layer2Rule*

**create\_rule\_section**(name, add\_pos=None, after=None, before=None)

Create a rule section in a Firewall Policy. To specify a specific numbering position for the rule section, use the *add\_pos* field. If no position or before/after is specified, the rule section will be placed at the top which will encapsulate all rules below. Create a rule section for the relevant policy:

```
policy = FirewallPolicy('mypolicy')
policy.fw_ipv4_access_rules.create_rule_section(name='atop')
# For NAT rules
policy.fw_ipv4_nat_rules.create_rule_section(name='mysection', add_pos=5)
```

**Parameters**

- **name** (*str*) – create a rule section by name
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If *add\_pos* is not provided, rule is inserted in position 1. Mutually exclusive with *after* and *before* params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with *add\_pos* and *before* params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with *add\_pos* and *after* params.

**Raises**

- ***MissingRequiredInput*** – when options are specified the need additional setting, i.e. use\_vpn action requires a vpn policy be specified.
- ***CreateRuleFailed*** – rule creation failure

**Returns**

the created ipv4 rule

**Return type**

*IPv4Rule*

**14.11.1.3 EthernetRule**

**class** smc.policy.rule.**EthernetRule**(\*\*meta)

Bases: RuleCommon, *Rule*, *SubElement*

Ethernet Rule represents a policy on a layer 2 or IPS engine.

If logical\_interfaces parameter is left blank, 'any' logical interface is used.

Create an ethernet rule for a layer 2 policy:

```
policy = Layer2Policy('layer2policy')
policy.layer2_ethernet_rules.create(name='l2rule',
                                    logical_interfaces=['dmz'],
                                    sources='any',
                                    action='discard')
```

**create**(*name*, *sources*=None, *destinations*=None, *services*=None, *action*='allow', *is\_disabled*=False, *logical\_interfaces*=None, *add\_pos*=None, *after*=None, *before*=None, *comment*=None, *validate*=True, *\*\*kw*)

Create an Ethernet rule

Changed in version 0.7.0: Action field now requires a list of actions as strings when using SMC version >= 6.6.0

#### Parameters

- **name** (*str*) – name of rule
- **sources** (*list*[*str*, *Element*]) – source/s for rule
- **destinations** (*list*[*str*, *Element*]) – destination/s for rule
- **services** (*list*[*str*, *Element*]) – service/s for rule
- **action** (*str*) – |allow|continue|discard|refuse|blacklist
- **is\_disabled** (*bool*) – whether to disable rule or not
- **logical\_interfaces** (*list*) – logical interfaces by name
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If *add\_pos* is not provided, rule is inserted in position 1. Mutually exclusive with *after* and *before* params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with *add\_pos* and *before* params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with *add\_pos* and *after* params.
- **validate** (*bool*) – validate the inspection policy during rule creation. Default: True

#### Raises

- **MissingRequiredInput** – when options are specified the need additional setting, i.e. *use\_vpn* action requires a *vpn* policy be specified.
- **CreateRuleFailed** – rule creation failure

#### Returns

newly created rule

#### Return type

*EthernetRule*

**create\_rule\_section**(*name*, *add\_pos*=None, *after*=None, *before*=None)

Create a rule section in a Firewall Policy. To specify a specific numbering position for the rule section, use the *add\_pos* field. If no position or before/after is specified, the rule section will be placed at the top which will encapsulate all rules below. Create a rule section for the relevant policy:

```
policy = FirewallPolicy('mypolicy')
policy.fw_ipv4_access_rules.create_rule_section(name='attop')
# For NAT rules
policy.fw_ipv4_nat_rules.create_rule_section(name='mysection', add_pos=5)
```

#### Parameters

- **name** (*str*) – create a rule section by name
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with after and before params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with add\_pos and before params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with add\_pos and after params.

#### Raises

- **MissingRequiredInput** – when options are specified the need additional setting, i.e. use\_vpn action requires a vpn policy be specified.
- **CreateRuleFailed** – rule creation failure

#### Returns

the created ipv4 rule

#### Return type

*IPv4Rule*

#### 14.11.1.4 IPv6Rule

```
class smc.policy.rule.IPv6Rule(**meta)
```

Bases: *IPv4Rule*

IPv6 access rule defines sources and destinations that must be in IPv6 format.

---

**Note:** It is possible to submit a source or destination in IPv4 format, however this will fail validation when attempting to push policy.

---

#### 14.11.2 NATRule

```
class smc.policy.rule_nat.NATRule
```

Bases: *Rule*

#### property action

Action for this rule.

#### Return type

*Action*

**property authentication\_options**

Read only authentication options field

**Return type**

*AuthenticationOptions*

**property dynamic\_src\_nat**

Dynamic Source NAT configuration for this NAT rule.

**Return type**

*DynamicSourceNAT*

**property static\_dst\_nat**

Static Destination NAT configuration for this NAT rule

**Return type**

StaticDestNAT

**property static\_src\_nat**

Static Source NAT configuraiton for this NAT rule.

**Return type**

*StaticSourceNAT*

**update**(*validate=True, sources=None, destinations=None, services=None, dynamic\_src\_nat=None, dynamic\_src\_nat\_ports=(1024, 65535), dynamic\_src\_nat\_automatic\_proxy=None, static\_src\_nat=None, static\_dst\_nat=None, static\_dst\_nat\_ports=None, static\_dst\_nat\_automatic\_proxy=None, \*\*kwargs*)

update a rule

**Parameters**

- **sources** (*str, list[Element]* *str can be "any" or json*) – source/s for rule
- **destinations** (*str, list[Element]* *str can be "any" or json*) – destination/s for rule
- **services** (*str, list[Element]* *str can be "any" or json*) – service/s for rule
- **validate** (*bool*) – validate the policy before update; default True
- **dynamic\_src\_nat** (*str, Element*) – str ip or Element for dest NAT
- **dynamic\_src\_nat\_ports** (*tuple*) – starting and ending ports for PAT. Default: (1024, 65535)
- **dynamic\_src\_nat\_automatic\_proxy** (*bool*) – Is Automatic Proxy ARP enabled?
- **static\_src\_nat** (*str*) – ip or element href of used for source NAT
- **static\_dst\_nat** (*str*) – destination NAT IP address or element href
- **static\_dst\_nat\_ports** (*tuple*) – ports or port range used for original and destination ports (only needed if a different destination port is used and does not match the rules service port)
- **static\_dst\_nat\_automatic\_proxy** (*bool*) – Is Automatic Proxy ARP enabled?

**Returns**

href of this rule

**Return type**

*str*

**property used\_on**

Used on specific whether this NAT rule has a specific engine that this rule applies to. Default is ANY (unspecified).

**Parameters**

**value** (*str*, *Element*) – Can be the strings ‘ANY’ or ‘NONE’ or an Engine element type.

**Returns**

‘ANY’, ‘NONE’ or the Engine element

**14.11.2.1 IPv4NATRule**

**class** smc.policy.rule\_nat.IPv4NATRule(\*\*meta)

Bases: RuleCommon, NATRule, SubElement

Create NAT Rules for relevant policy types. Rule requirements are similar to a normal rule with exception of the NAT field and no action field.

Like policy rules, specifying source/destination and services can be done either using the element href or element defined in element classes defined under package smc.elements. For example, using networks from *smc.elements.network* or services from *smc.elements.service*.

Example of creating a dynamic source NAT for host ‘kali’:

```
policy = FirewallPolicy('smcpython')
policy.fw_ipv4_nat_rules.create(name='mynat',
                                sources=[Host('kali')],
                                destinations='any',
                                services='any',
                                dynamic_src_nat='1.1.1.1',
                                dynamic_src_nat_ports=(1024,65535))
```

Example of creating a static source NAT for host ‘kali’:

```
policy.fw_ipv4_nat_rules.create(name='mynat',
                                sources=[Host('kali')],
                                destinations='any',
                                services='any',
                                static_src_nat='1.1.1.1')
```

Example of creating a destination NAT rule for destination host ‘3.3.3.3’ with destination translation address of ‘1.1.1.1’:

```
policy.fw_ipv4_nat_rules.create(name='mynat',
                                sources='any',
                                destinations=[Host('3.3.3.3')],
                                services='any',
                                static_dst_nat='1.1.1.1')
```

Destination NAT with destination port translation:

```
policy.fw_ipv4_nat_rules.create(name='aws_client',
                                sources='any',
                                destinations=[Alias('$ Interface ID 0.ip')],
                                services='any',
```

(continues on next page)

(continued from previous page)

```
static_dst_nat='1.1.1.1',
static_dst_nat_ports=(2222,22),
used_on=engine.href)
```

Create an any/any no NAT rule from host 'kali':

```
policy.fw_ipv4_nat_rules.create(name='nonat',
                                sources=[Host('kali')],
                                destinations='any',
                                services='any')
```

```
create(name, sources=None, destinations=None, services=None, dynamic_src_nat=None,
       dynamic_src_nat_ports=(1024, 65535), dynamic_src_nat_automatic_proxy=None,
       static_src_nat=None, static_dst_nat=None, static_dst_nat_ports=None,
       static_dst_nat_automatic_proxy=None, is_disabled=False, used_on='ANY', add_pos=None,
       after=None, before=None, comment=None, validate=True)
```

Create a NAT rule.

When providing sources/destinations or services, you can provide the element href, network element or services from `smc.elements`. You can also mix href strings with Element types in these fields.

#### Parameters

- **name** (*str*) – name of NAT rule
- **sources** (*list(str, Element)*) – list of sources by href or Element
- **destinations** (*list(str, Element)*) – list of destinations by href or Element
- **services** (*list(str, Element)*) – list of services by href or Element
- **dynamic\_src\_nat** (*str, Element*) – str ip or Element for dest NAT
- **dynamic\_src\_nat\_ports** (*tuple*) – starting and ending ports for PAT. Default: (1024, 65535)
- **dynamic\_src\_nat\_automatic\_proxy** (*bool*) – Is Automatic Proxy ARP enabled?
- **static\_src\_nat** (*str*) – ip or element href of used for source NAT
- **static\_dst\_nat** (*str*) – destination NAT IP address or element href
- **static\_dst\_nat\_ports** (*tuple*) – ports or port range used for original and destination ports (only needed if a different destination port is used and does not match the rules service port)
- **static\_dst\_nat\_automatic\_proxy** (*bool*) – Is Automatic Proxy ARP enabled?
- **is\_disabled** (*bool*) – whether to disable rule or not
- **used\_on** (*str, Element*) – Can be None, 'ANY' or and Engine element. Default is 'ANY'.
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If add\_pos is not provided, rule is inserted in position 1. Mutually exclusive with after and before params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with add\_pos and before params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with add\_pos and after params.

- **comment** (*str*) – optional comment for the NAT rule
- **validate** (*bool*) – validate the inspection policy during rule creation. Default: True

**Raises**

- *InvalidRuleValue* – if rule requirements are not met
- *CreateRuleFailed* – rule creation failure

**Returns**

newly created NAT rule

**Return type**

*IPv4NATRule*

**create\_rule\_section**(*name*, *add\_pos=None*, *after=None*, *before=None*)

Create a rule section in a Firewall Policy. To specify a specific numbering position for the rule section, use the *add\_pos* field. If no position or before/after is specified, the rule section will be placed at the top which will encapsulate all rules below. Create a rule section for the relevant policy:

```
policy = FirewallPolicy('mypolicy')
policy.fw_ipv4_access_rules.create_rule_section(name='atop')
# For NAT rules
policy.fw_ipv4_nat_rules.create_rule_section(name='mysection', add_pos=5)
```

**Parameters**

- **name** (*str*) – create a rule section by name
- **add\_pos** (*int*) – position to insert the rule, starting with position 1. If the position value is greater than the number of rules, the rule is inserted at the bottom. If *add\_pos* is not provided, rule is inserted in position 1. Mutually exclusive with *after* and *before* params.
- **after** (*str*) – Rule tag to add this rule after. Mutually exclusive with *add\_pos* and *before* params.
- **before** (*str*) – Rule tag to add this rule before. Mutually exclusive with *add\_pos* and *after* params.

**Raises**

- *MissingRequiredInput* – when options are specified the need additional setting, i.e. *use\_vpn* action requires a *vpn* policy be specified.
- *CreateRuleFailed* – rule creation failure

**Returns**

the created ipv4 rule

**Return type**

*IPv4Rule*

### 14.11.2.2 IPv6NATRule

**class** smc.policy.rule\_nat.IPv6NATRule(\*\*meta)

Bases: *IPv4NATRule*

Represents an IPv6 NAT rule. Source and/or destination (depending on NAT type) should be an IPv6 address. It will be possible to submit an IPv4 address however the policy validation engine will fail when being deployed to an engine and the rule will be ignored.

### 14.11.3 RuleElements

**class** smc.policy.rule\_elements.RuleElement

Rule Element encapsulates actions for source, destination and service fields.

**add**(data)

Add a single entry to field.

Entries can be added to a rule using the href of the element or by loading the element directly. Element should be of type *smc.elements.network*. After modifying rule, call *save()*.

Example of adding entry by element:

```
policy = FirewallPolicy('policy')
for rule in policy.fw_ipv4_nat_rules.all():
    if rule.name == 'therule':
        rule.sources.add(Host('myhost'))
        rule.save()
```

---

**Note:** If submitting type Element and the element cannot be found, it will be skipped.

---

#### Parameters

**data** (Element or str) – entry to add

**add\_many**(data)

Add multiple entries to field. Entries should be list format. Entries can be of types relevant to the field type. For example, for source and destination fields, elements may be of type *smc.elements.network* or be the elements direct href, or a combination of both.

Add several entries to existing rule:

```
policy = FirewallPolicy('policy')
for rule in policy.fw_ipv4_nat_rules.all():
    if rule.name == 'therule':
        rule.sources.add_many([Host('myhost'),
                              'http://1.1.1.1/hosts/12345'])
        rule.save()
```

#### Parameters

**data** (list) – list of sources

---

**Note:** If submitting type Element and the element cannot be found, it will be skipped.

---

**all()**

Return all destinations for this rule. Elements returned are of the object type for the given element for further introspection.

Search the fields in rule:

```
for sources in rule.sources.all():
    print('My source: %s' % sources)
```

**Returns**

elements by resolved object type

**Return type**

`list(Element)`

**all\_as\_href()**

Return all elements without resolving to `smc.elements.network` or `smc.elements.service`. Just raw representation as href.

**Returns**

elements in href form

**Return type**

`list`

**property is\_any**

Is the field set to any

**Return type**

`bool`

**property is\_none**

Is the field set to none

**Return type**

`bool`

**set\_any()**

Set field to any

**set\_none()**

Set field to none

**unset\_any()**

UnSet field to any

**update\_field(*elements*)**

Update the field with a list of provided values but only if the values are different. Return a boolean indicating whether a change was made indicating whether *save* should be called. If the field is currently set to any or none, then no comparison is made and field is updated.

**Parameters**

**elements** (`list`) – list of elements in href or Element format to compare to existing field

**Return type**

`bool`

#### 14.11.3.1 Source

**class** smc.policy.rule\_elements.**Source**(*rule=None*)

Bases: [RuleElement](#), [NestedDict](#)

Source fields for a rule

**property** src

All elements corresponding to the matching criteria (if not any or none).

**Returns**

list value: source elements

#### 14.11.3.2 Destination

**class** smc.policy.rule\_elements.**Destination**(*rule=None*)

Bases: [RuleElement](#), [NestedDict](#)

Destination fields for a rule.

**property** dst

All elements corresponding to the matching criteria (if not any or none).

**Returns**

list value: source elements

#### 14.11.3.3 Service

**class** smc.policy.rule\_elements.**Service**(*rule=None*)

Bases: [RuleElement](#), [NestedDict](#)

Service fields for a rule

**property** service

All elements corresponding to the matching criteria (if not any or none).

**Returns**

list value: source elements

#### 14.11.3.4 Action

**class** smc.policy.rule\_elements.**Action**(*rule=None*)

Bases: [ActionMixin](#)

This represents the action associated with the rule.

**property** antis spam

Enable or disable anti-spam

**Parameters**

**value** (*bool*) – True, False, None (inherit from continue rule)

**Return type**

*bool*

**property decrypting**

New in version 0.6.0: Requires SMC version >= 6.3.3

Whether the decryption is enabled on this rule.

**Parameters**

**value** (*bool*) – True, False, None (inherit from continue rule)

**Return type**

*bool*

**property deep\_inspection**

Selects traffic that matches this rule for checking against the Inspection Policy referenced by this policy. Traffic is inspected as the Protocol that is attached to the Service element in this rule.

**Parameters**

**value** (*bool*) – True, False, None (inherit from continue rule)

**Return type**

*bool*

**property dos\_protection**

Enable or disable DOS protection mode

**Parameters**

**value** (*bool*) – True, False, None (inherit from continue rule)

**Return type**

*bool*

**property file\_filtering**

(IPv4 Only) Inspects matching traffic against the File Filtering policy. Selecting this option should also activates the Deep Inspection option. You can further adjust virus scanning in the Inspection Policy.

**Parameters**

**value** (*bool*) – True, False, None (inherit from continue rule)

**Return type**

*bool*

**property forced\_next\_hop\_element**

If action includes forced\_next\_hop, specify the forced next hop element.

**Return type**

NetworkElement

**property forced\_next\_hop\_ip**

If action includes forced\_next\_hop, specify the forced next hop IP address.

**Return type**

str ip address

**property mobile\_vpn**

Mobile VPN only applies to engines that support VPN and that have the action of ‘enforce\_vpn’, ‘apply\_vpn’ or ‘forward\_vpn’ set. This will enable mobile VPN traffic on this VPN rule.

**Parameters**

**value** (*boolean*) – set mobile vpn on or off

**Return type**

boolean

**property network\_application\_latency\_monitoring**

Enable or Disable the Application Health Monitoring for the matching Traffic

**Parameters**

- **value** (*str*) – True, False, None (inherit from continue rule) in 7.0
- **value** – true, false, probing, None (inherit from continue rule) in 7.1 and above

**Return type**

*bool*

**property scan\_detection**

Enable or disable Scan Detection for traffic that matches the rule. This overrides the option set in the Engine properties.

Enable scan detection on this rule:

```
for rule in policy.fw_ipv4_access_rules.all():
    rule.action.scan_detection = 'on'
```

**Parameters**

**value** (*str*) – on|off|undefined

**Returns**

scan detection setting (on,off,undefined)

**Return type**

*str*

**property sub\_policy**

Sub policy is used when action=jump.

**Return type**

*FirewallSubPolicy*

**property user\_response**

Read-only user response setting

**property valid\_blacklist**

Used when action=blacklist. If specified with blacklist action, you want to restrict allowed block lists for this rule. Black list entries are only accepted from the components you specify (and from the engine command line). NGFW Engines are always allowed to add entries to their own block lists. If not specified with blacklist action, any black list entries are accepted from all components.

**Return type**

*list(Engine)*

---

**Note:** This method requires SMC version < 7.0

---

**property valid\_block\_list**

Used when action=block\_list. If specified with block\_list action, you want to restrict allowed block lists for this rule. Block list entries are only accepted from the components you specify (and from the engine command line). NGFW Engines are always allowed to add entries to their own block lists. If not specified with block\_list action, any block list entries are accepted from all components.

**Return type**

*list(Engine)*

---

**Note:** This method requires SMC version >= 7.0

---

### property vpn

Return vpn reference. Only used if ‘enforce\_vpn’, ‘apply\_vpn’, or ‘forward\_vpn’ is the action type.

#### Parameters

**value** (*PolicyVPN*) – set the policy VPN for VPN action

#### Return type

*PolicyVPN*

## 14.11.3.5 ConnectionTracking

**class** `smc.policy.rule_elements.ConnectionTracking(rule=None)`

Bases: `NestedDict`

Connection tracking settings can be configured on a per rule basis to control settings such as enforced MSS and how to handle connection states.

Configuring a rule to enable MSS and set connection state tracking to normal:

```
for rule in policy.fw_ipv4_access_rules.all():
    rule.action.connection_tracking_options.mss_enforced = True
    rule.action.connection_tracking_options.state = 'normal'
    rule.action.connection_tracking_options.mss_enforced_min_max = (1400, 1450)
    rule.action.connection_tracking_options.sync_connections = True
    rule.save()
```

### property mss\_enforced

Is MSS enforced

#### Parameters

**value** (*bool*) – True, False

#### Returns

*bool*

### property mss\_enforced\_min\_max

Allows entering the Minimum and Maximum value for the MSS in bytes. Headers are not included in the MSS value; MSS concerns only the payload portion of the packet.

#### Parameters

**value** (*tuple int*) – tuple containing (min, max) in bytes

#### Returns

(min, max) values

#### Return type

*tuple*

### property state

Connection tracking mode. See documentation for more info.

#### Parameters

**value** (*str*) – no, loose, normal, strict

**Returns**

str

**property sync\_connections**

Are sync connections enabled for this engine. If None, then this is set to inherit from a continue rule.

:return True, False, None (inherit from continue rule)

**property timeout**

The timeout (in seconds) after which inactive connections are closed. This timeout only concerns idle connections. Connections are not cut because of timeouts while the hosts are still communicating.

**Parameters**

**value** (*int*) – time in seconds

**Returns**

int

### 14.11.3.6 LogOptions

```
class smc.policy.rule_elements.LogOptions(rule=None)
```

Bases: `NestedDict`

Log Options represent the settings related to per rule logging.

Example of obtaining a rule reference and turning logging on for a particular rule:

```
policy = FirewallPolicy('smcpython')
for rule in policy.fw_ipv4_access_rules.all():
    if rule.name == 'foo':
        rule.options.log_accounting_info_mode = True
        rule.options.log_level = 'stored'
        rule.options.application_logging = 'enforced'
        rule.options.user_logging = 'enforced'
        rule.save()
```

**property application\_logging**

Stores information about Application use. You can log application use even if you do not use Applications for access control.

**Parameters**

**value** (*str*) – off|default|enforced

**Returns**

str

**property endpoint\_executable\_logging**

Stores information about EndPoint Executable use. You can log EndPoint Executable use even if you do not use EndPoint Executables for accesscontrol.

**Parameters**

**value** (*str*) – off|default|enforced

**Returns**

str

**property log\_accounting\_info\_mode**

Both connection opening and closing are logged and information on the volume of traffic is collected. This option is not available for rules that issue alerts. If you want to create reports that are based on traffic volume, you must select this option for all rules that allow traffic that you want to include in the reports.

**Parameters**

**value** (*bool*) – log accounting information (bits/bytes transferred)

**Returns**

bool

**property log\_alert**

If Log Level is set to Alert, specifies the Alert that is sent. Specifying different Alerts for different types of rules allows more fine-grained alert escalation policies.

**Returns**

AlertElement

**property log\_closing\_mode**

Specifying False means no log entries are created when connections are closed. True will mean both connection opening and closing are logged, but no information is collected on the volume of traffic.

**Parameters**

**value** (*bool*) – enable/disable accounting data

**Returns**

bool

**property log\_compression**

Stores information about Compress Logs.

**Parameters**

**value** (*str*) – off|only\_access|also\_inspection

**Returns**

str

**property log\_compression\_max\_burst\_size**

If Compress Logs is selected, the max burst size.

**Returns**

int. None if not defined.

**property log\_compression\_max\_log\_rate**

If Compress Logs is selected, the max log rate.

**Returns**

int. None if not defined.

**property log\_level**

Configure per rule logging. It is recommended to configure an Any/Any/Any/Continue rule in position 1 if global logging is required. This can be used to override any global logging setting.

**Parameters**

**value** (*str*) – none|stored|transient|essential|alert|undefined

**Returns**

str

**property log\_payload\_excerpt**

Stores an excerpt of the packet that matched. The maximum recorded excerpt size is 4 KB. This allows quick viewing of the payload in the logs view.

**Parameters**

**value** (*bool*) – collect excerpt or not

**Returns**

bool

**property log\_payload\_record**

Records the traffic up to the limit that is set in the Record Length field.

**Parameters**

**value** (*bool*) – True, False

**Returns**

bool

**property log\_severity**

Read only log severity level :return: int

**property qos\_class**

Matching QoS Classes. The QoS Rules are linked to different types of traffic using the QoS Classes. QoS Classes are matched to traffic in the Access rules with the following actions: - Access rules with the Allow action set a QoS Class for traffic

that matches the rules.

- Access rules with the Continue action set a QoS Class for all subsequent matching rules that have no specific QoS Class defined.
- Access rules with the Use VPN action (Firewall only) set a QoS Class for VPN traffic. Incoming VPN traffic may also match a normal Allow rule after decryption. Otherwise, for outgoing traffic, encryption is done after the QoS Policy is checked. For incoming traffic, decryption is done before the QoS Policy is checked.

However, if you only want to read and use DSCP markers set by other devices, the QoS Class is assigned according to the rules on the DSCP Match/Mark tab of the QoS Policy.

**Returns**

QoSClass

**property url\_category\_logging**

Stores information about URL categories use.

**Parameters**

**value** (*str*) – off|default|enforced

**Returns**

str

**property user\_logging**

Stores information about Users when they are used as the Source or Destination of an Access rule. You must select this option if you want Users to be referenced by name in log entries, statistics, reports, and user monitoring. Otherwise, only the IP address associated with the User at the time the log was created is stored.

**Parameters**

**value** (*str*) – off|default|enforced

**Returns**

str

**14.11.3.7 AuthenticationOptions****class** smc.policy.rule\_elements.**AuthenticationOptions**(rule=None)Bases: `NestedDict`

Authentication options are set on a per rule basis and dictate whether a user requires identification to match.

**property methods**

Read only authentication methods enabled

**Returns**

list value: auth methods enabled

**property require\_auth**

Ready only authentication required

**Returns**

boolean

**property timeout**

Timeout between authentications

**Returns**

int

**property users**

List of users required to authenticate

**Returns**

list

**14.11.3.8 MatchExpression****class** smc.policy.rule\_elements.**MatchExpression**(name=None, \*\*meta)Bases: `Element`

A match expression is used in the source / destination / service fields to group together elements into an 'AND'ed configuration. For example, a normal rule might have a source field that could include network=172.18.1.0/24 and zone=Internal objects. A match expression enables you to AND these elements together to enforce the match requires both. Logically it would be represented as (network 172.18.1.0/24 AND zone Internal).

```
>>> from smc.elements.network import Host, Zone
>>> from smc.policy.rule_elements import MatchExpression
>>> from smc.policy.layer3 import FirewallPolicy
>>> match = MatchExpression.create(name='mymatch', network_element=Host('kali'),
                                zone=Zone('Mail'))
>>> policy = FirewallPolicy('smcpython')
>>> policy.fw_ipv4_access_rules.create(name='myrule', sources=[match],
                                destinations='any', services='any')
'http://172.18.1.150:8082/6.2/elements/fw_policy/261/fw_ipv4_access_rule/2099740'
>>> rule = policy.search_rule('myrule')
...
>>> for source in rule[0].sources.all():
```

(continues on next page)

(continued from previous page)

```
... print(source, source.values())
...
MatchExpression(name=MatchExpression _1491760686976_2) [Zone(name=Mail),
↪Host(name=kali)]
```

Match expressions can also be used on service fields by providing values for service and service ports as follows:

```
match = MatchExpression.create(name='mymatch', service=ApplicationSituation('FTP'),
                               service_ports='TCPService('Any TCP Service')')
```

If the service match expression requires ANY ports, you can use the string of 'any' to provide this logic:

```
match = MatchExpression.create(name='mymatch', service=ApplicationSituation('FTP'),
                               service_ports='any')
```

Once the service match expression is created, you can use that in the policy rule:

```
policy.fw_ipv4_access_rules.create(name='myrule', sources='any', destinations='any',
                                   services=[match], action=['discard'])
```

```
classmethod create(name, user=None, network_element=None, domain_name=None, zone=None,
                   executable=None, service=None, service_ports='any')
```

Create a match expression

#### Parameters

- **name** (*str*) – name of match expression
- **user** (*str*) – name of user or user group
- **network\_element** (*Element*) – valid network element type, i.e. host, network, etc
- **domain\_name** (*DomainName*) – domain name network element
- **zone** (*Zone*) – zone to use
- **executable** (*str*) – name of executable or group
- **service** – any service type, i.e. TCPService, UDPService, ApplicationSituation
- **service\_ports** (*str*, *Element*) – specify the service ports for the given service. Provide 'ANY' as the value if the match expression requires a service/application and ANY ports

#### Raises

*ElementNotFound* – specified object does not exist

#### Returns

instance with meta

#### Return type

*MatchExpression*

#### 14.11.4 NATElements

**class** `smc.policy.rule_nat.NATElement(rule=None)`

Common structure for source and destination NAT configurations.

**property** `automatic_proxy`

Is proxy arp enabled. Leaving this in the on state is recommended.

**Parameters**

**value** (*bool*) – enable/disable proxy arp

**Return type**

*bool*

**property** `has_nat`

Is NAT already enabled (assuming modification) or newly created.

**Returns**

boolean

**set\_none()**

Clear the NAT field for this NAT rule. You must call *update* or *save* on the rule to commit this change.

**Returns**

None

**property** `translated_value`

The translated value for this NAT type. If this rule does not have a NAT value defined, this will return None.

**Returns**

NATValue or None

**Return type**

NATValue

**update\_field**(*element\_or\_ip\_address=None, start\_port=None, end\_port=None, \*\*kw*)

Update the source NAT translation on this rule. You must call *save* or *update* on the rule to make this modification. To update the source target for this NAT rule, update the source field directly using `rule.sources.update_field(...)`. This will automatically update the NAT value. This method should be used when you want to change the translated value or the port mappings for dynamic source NAT.

Starting and ending ports are only used for dynamic source NAT and define the available ports for doing PAT on the outbound connection.

**Parameters**

- **element\_or\_ip\_address** (*str, Element*) – Element or IP address that is the NAT target
- **start\_port** (*int*) – starting port value, only used for dynamic source NAT
- **end\_port** (*int*) – ending port value, only used for dynamic source NAT
- **automatic\_proxy** (*bool*) – whether to enable proxy ARP (default: True)

**Returns**

boolean indicating whether the rule was modified

**Return type**

*bool*

#### 14.11.4.1 DynamicSourceNAT

**class** smc.policy.rule\_nat.DynamicSourceNAT(*rule=None*)

Bases: [NATElement](#)

Dynamic source NAT is typically used for outbound traffic and typically uses a range of ports to perform PAT operations.

**property end\_port**

Ending port specified for outbound dynamic source NAT (PAT)

**Return type**

[int](#)

**property start\_port**

Start port for dynamic source NAT (PAT)

**Return type**

[int](#)

**property translated\_value**

The translated value for this NAT type. If this rule does not have a NAT value defined, this will return None.

**Returns**

NATValue or None

**Return type**

NATValue

#### 14.11.4.2 StaticSourceNAT

**class** smc.policy.rule\_nat.StaticSourceNAT(*rule=None*)

Bases: [NATElement](#)

Source NAT defines the available options for configuration. This is typically used for outbound traffic where you need to hide the original source address.

Example of changing existing source NAT rule to use a different source NAT address:

```
for rule in policy.fw_ipv4_nat_rules.all():
    if rule.name == 'sourcenat':
        rule.static_src_nat.translated_value = '10.10.50.50'
        rule.save()
```

#### 14.11.4.3 DynamicSourceNAT

**class** smc.policy.rule\_nat.DynamicSourceNAT(*rule=None*)

Bases: [NATElement](#)

Dynamic source NAT is typically used for outbound traffic and typically uses a range of ports to perform PAT operations.

**property end\_port**

Ending port specified for outbound dynamic source NAT (PAT)

**Return type**`int`**property start\_port**

Start port for dynamic source NAT (PAT)

**Return type**`int`**property translated\_value**

The translated value for this NAT type. If this rule does not have a NAT value defined, this will return None.

**Returns**

NATValue or None

**Return type**

NATValue

## 14.12 VPN

Represents classes responsible for configuring VPN settings such as PolicyVPN, RouteVPN and all associated configurations.

---

**Note:** See API reference documentation on the Engine for instructions on how to enable the engine for VPN.

---

### 14.12.1 PolicyVPN

**class** `smc.vpn.policy.PolicyVPN`(*name=None, \*\*meta*)

Bases: *Element*

Create a new VPN Policy.

```
>>> PolicyVPN.create(name='myvpn')
PolicyVPN(name=myvpn)
>>> v = PolicyVPN('myvpn')
>>> print(v.vpn_profile)
VPNProfile(name=VPN-A Suite)
```

When making VPN Policy modifications, you must first call `open()`, make your modifications and then call `save()` followed by `close()`.

**Variables**

**vpn\_profile** (*VPNProfile*) – VPN Profile used by this Policy VPN

**add\_central\_gateway**(*gateway*)

Add SMC managed internal gateway to the Central Gateways of this VPN

**Parameters**

**gateway** (*Engine*, *ExternalGateway*) – An external gateway, engine or href for the central gateway

**Raises**

*PolicyCommandFailed* – could not add gateway

**Returns**

None

**static add\_internal\_gateway\_to\_vpn**(*internal\_gateway\_href*, *vpn\_policy*, *vpn\_role*='central')

Add an internal gateway (managed engine node) to a VPN policy based on the internal gateway href.

**Parameters**

- **internal\_gateway\_href** (*str*) – href for engine internal gw
- **vpn\_policy** (*str*) – name of vpn policy
- **vpn\_role** (*str*) – central|satellite

**Returns**

True for success

**Return type**

bool

**add\_mobile\_gateway**(*all\_central\_gateways*=False, *all\_gateways*=False, *gateways*=None)

Add a mobile VPN gateway to this policy VPN. You can select all central gateways, all gateways in overall topology or specify a list of gateways to allow for mobile VPN.

Example of adding or removing a mobile VPN gateway:

```
policy_vpn = PolicyVPN('myvpn')
policy_vpn.update(mobile_vpn_topology_mode='Selected Gateways below')
policy_vpn.open()

policy_vpn.add_mobile_vpn_gateway(gateways=Engine('azure'))

policy_vpn.save()
policy_vpn.close()
```

**Parameters****gateway** (*Engine*, *ExternalGateway*) – An external gateway, engine or href for the mobile gateway**Raises***PolicyCommandFailed* – could not add gateway**Return type**

None

**add\_satellite\_gateway**(*gateway*)

Add gateway node as a satellite gateway for this VPN. You must first have the gateway object created. This is typically used when you either want a hub-and-spoke topology or the test\_external gateway is a non-SMC managed device.

**Parameters****gateway** (*Engine*, *ExternalGateway*) – An external gateway, engine or href for the central gateway**Raises***PolicyCommandFailed* – could not add gateway**Returns**

None

**property central\_gateway\_node**

Central Gateway Node acts as the hub of a hub-spoke VPN.

**Return type**

*SubElementCollection(GatewayNode)*

**close()**

Close the policy. This is only a valid method for SMC version <= 6.1

**Raises**

***PolicyCommandFailed*** – close failed with reason

**Returns**

None

**classmethod create**(*name, nat=False, mobile\_vpn\_topology\_mode=None, vpn\_profile=None, link\_usage\_profile=None*)

Create a new policy based VPN

**Parameters**

- **name** – name of vpn policy
- **nat** (*bool*) – whether to apply NAT to the VPN (default False)
- **mobile\_vpn\_topology\_mode** – whether to allow remote vpn
- **vpn\_profile** (*VPNProfile*) – reference to VPN profile, or uses default
- **link\_usage\_profile** (*LinkUsageProfile*) – reference to link usage profile of set

**Return type**

*PolicyVPN*

**enable\_disable\_nat()**

Enable or disable NAT on this policy. If NAT is disabled, it will be enabled and vice versa.

**Returns**

None

**property mobile\_gateway\_node**

Mobile Gateway's are represented by client endpoints connecting to the policy based VPN.

**Return type**

*SubElementCollection(GatewayNode)*

**property mobile\_vpn\_topology**

Is the policy VPN configured for mobile VPN gateways. Valid modes: 'Selected Gateways below', 'Only central Gateways from overall topology', 'All Gateways from overall topology', 'None'

**property nat**

Is NAT enabled on this vpn policy

**Returns**

NAT enabled

**Return type**

*bool*

**open()**

Open the policy for editing. This is only a valid method for SMC version <= 6.1

**Raises**

*PolicyCommandFailed* – couldn't open policy with reason

**Returns**

None

**property satellite\_gateway\_node**

Node level settings for configured satellite gateways

**Return type**

*SubElementCollection(GatewayNode)*

**save()**

Save the policy after editing. This is only a valid method for SMC version <= 6.1

**Raises**

*PolicyCommandFailed* – save failed with reason

**Returns**

None

**property tunnels**

Return all tunnels for this VPN. A tunnel is defined as two end points within the VPN topology. Endpoints are automatically configured based on whether they are a central gateway or satellite gateway. This provides access to enabling/disabling and setting the preshared key for the linked endpoints. List all tunnel mappings for this policy vpn:

```
for tunnel in policy.tunnels:
    tunnela = tunnel.tunnel_side_a
    tunnelb = tunnel.tunnel_side_b
    print(tunnela.gateway)
    print(tunnelb.gateway)
```

**Return type**

*SubElementCollection(GatewayTunnel)*

**validate()**

Return a validation string from the SMC after running validate on this VPN policy.

**Returns**

status as dict

**Return type**

dict

## 14.12.2 RouteVPN

New in version 0.5.6: Route based VPNs with multi-domain support, requires SMC >=6.3

Module for configuring Route Based VPN. Creating a route based VPN consists of creating a local and remote tunnel endpoint. Once you have the required endpoints, use TunnelEndpoint classmethods to create the VPN by type (i.e. GRE, IPSEC).

List all existing route based VPNs:

```
print(list(RouteVPN.objects.all()))
```

Example of fully provisioning an IPSEC wrapped RBVPN using a third party remote GW:

```

engine = Layer3Firewall.create(name='myfw', mgmt_ip='1.1.1.1', mgmt_network='1.1.1.0/24')

# Add a second layer 3 interface for VPN
engine.physical_interface.add_layer3_interface(
    interface_id=1, address='10.10.10.10', network_value='10.10.10.0/24', zone_ref='vpn')

engine.tunnel_interface.add_layer3_interface(
    interface_id=1000,
    address='2.2.2.2',
    network_value='2.2.2.0/24')

# Enable VPN on the 'Internal Endpoint' interface
vpn_endpoint = engine.vpn_endpoint.get_contains('10.10.10.10')
vpn_endpoint.update(enabled=True)

# A Tunnel Endpoint pairs the interface of the NGFW with it's local VPN gateway.
# You must create a tunnel endpoint for both sides of the Route VPN.

# Create the local Tunnel Endpoint using the engine internal gateway
# and previously created tunnel interface
tunnel_if = engine.tunnel_interface.get(1000)
local_gateway = TunnelEndpoint.create_ipsec_endpoint(engine.vpn.internal_gateway, tunnel_
→ if)

# Define the remote side details

# Create the remote side network elements
Network.create(name='remotenet', ipv4_network='172.18.10.0/24')

# An ExternalGateway defines the remote side as a 3rd party gateway
# Add the address of the remote gateway and the network element created
# that defines the remote network/s.
gw = ExternalGateway.create(name='remotegw')
gw.external_endpoint.create(name='endpoint1', address='10.10.10.10')
gw.vpn_site.create(name='remotesite', site_element=[Network('remotenet')])

# Create the remote Tunnel Endpoint using the external gateway
remote_gateway = TunnelEndpoint.create_ipsec_endpoint(gw)

RouteVPN.create_ipsec_tunnel(
    name='myvpn',
    preshared_key='abcdefgh123456789',
    local_endpoint=local_gateway,
    remote_endpoint=remote_gateway)

```

Create a GRE Tunnel Mode RBVPN with a remote gateway (non-SMC managed):

```

engine = Engine('fw')

# Enable VPN endpoint on interface 0
# Note: An interface can have multiple IP addresses in which case you
# may want to get the VPN endpoint match by address
vpn_endpoint = None

```

(continues on next page)

(continued from previous page)

```

for endpoint in engine.vpn_endpoint:
    if endpoint.physical_interface.interface_id == '0':
        endpoint.update(enabled=True)
        vpn_endpoint = endpoint
        break

# Create a new Tunnel Interface for the engine
engine.tunnel_interface.add_layer3_interface(
    interface_id=3000, address='30.30.30.30', network_value='30.30.30.0/24')

tunnel_interface = engine.tunnel_interface.get(3000)
local_endpoint = TunnelEndpoint.create_gre_tunnel_endpoint(
    endpoint=vpn_endpoint, tunnel_interface=tunnel_interface)

# Create GRE tunnel endpoint for remote gateway
remote_endpoint = TunnelEndpoint.create_gre_tunnel_endpoint(
    remote_address='10.1.1.2')

# Create the top level IPSEC tunnel to encapsulate RBVPN
policy_vpn = PolicyVPN.create(name='myIPSEC')

RouteVPN.create_gre_tunnel_mode(
    name='mytunnelvpn',
    local_endpoint=local_endpoint,
    remote_endpoint=remote_endpoint,
    policy_vpn=policy_vpn)

```

Create a no-encryption GRE route based VPN between two managed NGFWs:

```

engine1 = Layer3Firewall.create(name='engine1', mgmt_ip='1.1.1.1', mgmt_network='1.1.1.0/
↪24')
engine1.tunnel_interface.add_layer3_interface(
    interface_id=1000,
    address='2.2.2.2',
    network_value='2.2.2.0/24')

# Obtain the 'internal endpoint' from the NGFW and enable VPN
for vpn in engine1.vpn_endpoint:
    internal_endpoint = vpn
    vpn.update(enabled=True)

tunnel_if = engine1.tunnel_interface.get(1000)
local_gateway = TunnelEndpoint.create_gre_tunnel_endpoint(
    internal_endpoint, tunnel_if)

engine2 = Layer3Firewall.create(name='engine2', mgmt_ip='1.1.1.1', mgmt_network='1.1.1.0/
↪24')
engine2.tunnel_interface.add_layer3_interface(
    interface_id=1000,
    address='2.2.2.2',
    network_value='2.2.2.0/24')

```

(continues on next page)

(continued from previous page)

```
# Obtain the 'internal_endpoint' from the NGFW and enable VPN
for vpn in engine2.vpn_endpoint:
    internal_endpoint = vpn
    vpn.update(enabled=True)

tunnel_if = engine2.tunnel_interface.get(1000)
remote_gateway = TunnelEndpoint.create_gre_tunnel_endpoint(
    internal_endpoint, tunnel_if)

RouteVPN.create_gre_tunnel_no_encryption(
    name='openvpn',
    local_endpoint=local_gateway,
    remote_endpoint=remote_gateway)
```

**class** `smc.vpn.route.EndpointTunnel`(\*\*meta)

Bases: `SubElement`

An Endpoint tunnel represents the point to point connection between two IPSEC endpoints in a RouteVPN configuration. This provides access to see the point to point connections, whether the link is enabled.

**enable\_disable()**

Enable or disable the tunnel link between endpoints.

**Raises**

`UpdateElementFailed` – failed with reason

**Returns**

None

**property enabled**

Whether the VPN link between endpoints is enabled

**Return type**

bool

**class** `smc.vpn.route.RouteVPN`(name=None, \*\*meta)

Bases: `Element`

Route based VPN in NGFW.

**Variables**

- **vpn\_profile** (`VPNProfile`) – VPNProfile reference for this RouteVPN
- **monitoring\_group** (`TunnelMonitoringGroup`) – tunnel monitoring group reference

**classmethod** `create_geneve_mode`(name, local\_endpoint, remote\_endpoint, geneve\_vni=0, geneve\_destination\_port=6081, mtu=0, pmtu\_discovery=True, ttl=0, enabled=True, comment=None)

Create a Geneve (Generic Network Virtualization Encapsulation) No Encryption tunnel. The Geneve packet format consists of a compact tunnel header encapsulated in UDP over either IPv4 or IPv6. :param bool enabled: to enable/disable the tunnel by default :param int geneve\_destination\_port: default 6081 (1-65535) :param int geneve\_vni: Virtual network interface (0-16777215) :param str name: name of VPN :param TunnelEndpoint local\_endpoint: the local side endpoint for

this VPN.

**Parameters**

- **remote\_endpoint** ([TunnelEndpoint](#)) – the remote side endpoint for this VPN.
- **mtu** ([int](#)) – Set MTU for this VPN tunnel (default: 0)
- **pmtu\_discovery** ([boolean](#)) – enable pmtu discovery (default: True)
- **ttl** ([int](#)) – ttl for connections on the VPN (default: 0)
- **comment** ([str](#)) – optional comment

**Raises**

[CreateVPNFailed](#) – failed to create the VPN with reason

**Return type**

[RouteVPN](#)

**classmethod** **create\_gre\_transport\_mode**(\*args, \*\*kwargs)

Create a transport based route VPN. This VPN type uses IPSEC for protecting the payload, therefore a VPN Profile is specified.

**Parameters**

- **name** ([str](#)) – name of VPN
- **local\_endpoint** ([TunnelEndpoint](#)) – the local side endpoint for this VPN.
- **remote\_endpoint** ([TunnelEndpoint](#)) – the remote side endpoint for this VPN.
- **preshared\_key** ([str](#)) – preshared key for RBVPN
- **monitoring\_group** ([TunnelMonitoringGroup](#)) – the group to place this VPN in for monitoring. (default: 'Uncategorized')
- **vpn\_profile** ([VPNProfile](#)) – VPN profile for this VPN. (default: VPN-A Suite)
- **mtu** ([int](#)) – Set MTU for this VPN tunnel (default: 0)
- **pmtu\_discovery** ([boolean](#)) – enable pmtu discovery (default: True)
- **ttl** ([int](#)) – ttl for connections on the VPN (default: 0)
- **comment** ([str](#)) – optional comment

**Raises**

[CreateVPNFailed](#) – failed to create the VPN with reason

**Return type**

[RouteVPN](#)

**classmethod** **create\_gre\_tunnel\_mode**(name, local\_endpoint, remote\_endpoint, policy\_vpn, mtu=0, pmtu\_discovery=True, ttl=0, enabled=True, comment=None)

Create a GRE based tunnel mode route VPN. Tunnel mode GRE wraps the GRE tunnel in an IPSEC tunnel to provide encrypted end-to-end security. Therefore a policy based VPN is required to 'wrap' the GRE into IPSEC.

**Parameters**

- **name** ([str](#)) – name of VPN
- **local\_endpoint** ([TunnelEndpoint](#)) – the local side endpoint for this VPN.
- **remote\_endpoint** ([TunnelEndpoint](#)) – the remote side endpoint for this VPN.
- **policy\_vpn** ([PolicyVPN](#)) – reference to a policy VPN

- **monitoring\_group** ([TunnelMonitoringGroup](#)) – the group to place this VPN in for monitoring. (default: 'Uncategorized')
- **mtu** ([int](#)) – Set MTU for this VPN tunnel (default: 0)
- **pmtu\_discovery** ([boolean](#)) – enable pmtu discovery (default: True)
- **ttl** ([int](#)) – ttl for connections on the VPN (default: 0)
- **comment** ([str](#)) – optional comment

**Raises**

[CreateVPNFailed](#) – failed to create the VPN with reason

**Return type**

[RouteVPN](#)

```
classmethod create_gre_tunnel_no_encryption(name, local_endpoint, remote_endpoint, mtu=0,
                                             pmtu_discovery=True, ttl=0, enabled=True,
                                             comment=None, gre_keepalive_period=None,
                                             gre_keepalive_retry=None)
```

Create a GRE No Encryption route Based VPN.

**Parameters**

- **name** ([str](#)) – name of VPN
- **local\_endpoint** ([TunnelEndpoint](#)) – the local side endpoint for this VPN.
- **remote\_endpoint** ([TunnelEndpoint](#)) – the remote side endpoint for this VPN.
- **monitoring\_group** ([TunnelMonitoringGroup](#)) – the group to place this VPN in for monitoring. (default: 'Uncategorized')
- **mtu** ([int](#)) – Set MTU for this VPN tunnel (default: 0)
- **pmtu\_discovery** ([boolean](#)) – enable pmtu discovery (default: True)
- **ttl** ([int](#)) – ttl for connections on the VPN (default: 0)
- **gre\_keepalive\_period** ([int](#)) – in s (1-32767s)
- **gre\_keepalive\_retry** ([int](#)) – (0-255)
- **comment** ([str](#)) – optional comment

**Raises**

[CreateVPNFailed](#) – failed to create the VPN with reason

**Return type**

[RouteVPN](#)

```
classmethod create_ipsec_tunnel(*args, **kwargs)
```

The VPN tunnel type negotiates IPsec tunnels in the same way as policy-based VPNs, but traffic is selected to be sent into the tunnel based on routing.

**Parameters**

- **name** ([str](#)) – name of VPN
- **local\_endpoint** ([TunnelEndpoint](#)) – the local side endpoint for this VPN.
- **remote\_endpoint** ([TunnelEndpoint](#)) – the remote side endpoint for this VPN.
- **preshared\_key** ([str](#)) – required if remote endpoint is an ExternalGateway

- **monitoring\_group** ([TunnelMonitoringGroup](#)) – the group to place this VPN in for monitoring. Default: ‘Uncategorized’.
- **vpn\_profile** ([VPNProfile](#)) – VPN profile for this VPN. (default: VPN-A Suite)
- **mtu** ([int](#)) – Set MTU for this VPN tunnel (default: 0)
- **pmtu\_discovery** ([boolean](#)) – enable pmtu discovery (default: True)
- **ttl** ([int](#)) – ttl for connections on the VPN (default: 0)
- **enabled** ([bool](#)) – enable the RBVPN or leave it disabled
- **comment** ([str](#)) – optional comment

**Raises**

[CreateVPNFailed](#) – failed to create the VPN with reason

**Return type**

[RouteVPN](#)

**disable()**

Disable this route based VPN

**Returns**

None

**enable()**

Enable this route based VPN

**Returns**

None

**property local\_endpoint**

The local endpoint for this RBVPN

**Return type**

[TunnelEndpoint](#)

**property remote\_endpoint**

The remote endpoint for this RBVPN

**Return type**

[TunnelEndpoint](#)

**set\_preshared\_key(new\_key)**

Set the preshared key for this VPN. A pre-shared key is only present when the tunnel type is ‘VPN’ or the encryption mode is ‘transport’.

**Returns**

None

**set\_tunnel\_group(tunnel\_group)**

Set the tunnel group for this RBVPN.

**Returns**

None

**property tunnel\_mode**

The tunnel mode for this RBVPN

**Return type**

[str](#)

**property tunnels**

Return all tunnels for this RBVPN in case of Tunnel Type ‘VPN’. This provides access to enabling/disabling for the linked endpoints. List all tunnel mappings for this route vpn:

```
for tunnel in rb_vpn.tunnels:
    print(tunnel.enabled)
```

**Return type**

*SubElementCollection(EndpointTunnel)*

**class** smc.vpn.route.RouteVPNTunnelMonitoringGroup(*name=None, \*\*meta*)

Bases: *TunnelMonitoringGroup*

Compatability class for 6.5 and earlier monitoring group entry point

**class** smc.vpn.route.TunnelEndpoint(*gateway\_ref=None, tunnel\_interface\_ref=None, endpoint\_ref=None, ip\_address=None*)

Bases: *object*

A Tunnel Endpoint represents one side of a route based VPN. Based on the RBVPN type required, you must create the local and remote endpoints and pass them into the RouteVPN create classmethods.

**Variables**

- **gateway** (*InternalGateway, ExternalGateway*) – reference to the element that is used by this tunnel endpoint
- **tunnel\_interface** (*TunnelInterface*) – Tunnel interface used by this tunnel endpoint

**classmethod** **create\_gre\_transport\_endpoint**(*endpoint, tunnel\_interface=None*)

Create the GRE transport mode endpoint. If the GRE transport mode endpoint is an SMC managed device, both an endpoint and a tunnel interface is required. If the GRE endpoint is an externally managed device, only an endpoint is required.

**Parameters**

- **endpoint** (*InternalEndpoint, ExternalEndpoint*) – the endpoint element for this tunnel endpoint.
- **tunnel\_interface** (*TunnelInterface*) – the tunnel interface for this tunnel endpoint. Required for SMC managed devices.

**Return type**

*TunnelEndpoint*

**classmethod** **create\_gre\_tunnel\_endpoint**(*endpoint=None, tunnel\_interface=None, remote\_address=None*)

Create the GRE tunnel mode or no encryption mode endpoint. If the GRE tunnel mode endpoint is an SMC managed device, both an endpoint and a tunnel interface is required. If the endpoint is externally managed, only an IP address is required.

**Parameters**

- **endpoint** (*InternalEndpoint, ExternalEndpoint*) – the endpoint element for this tunnel endpoint.
- **tunnel\_interface** (*TunnelInterface*) – the tunnel interface for this tunnel endpoint. Required for SMC managed devices.

- **remote\_address** (*str*) – IP address, only required if the tunnel endpoint is a remote gateway.

**Return type***TunnelEndpoint***classmethod** **create\_ipsec\_endpoint**(*gateway, tunnel\_interface=None*)

Create the VPN tunnel endpoint. If the VPN tunnel endpoint is an SMC managed device, both a gateway and a tunnel interface is required. If the VPN endpoint is an externally managed device, only a gateway is required.

**Parameters**

- **gateway** (*InternalGateway, ExternalGateway*) – the gateway for this tunnel endpoint
- **tunnel\_interface** (*TunnelInterface*) – Tunnel interface for this RBVPN. This can be None if the gateway is a non-SMC managed gateway.

**Return type***TunnelEndpoint***classmethod** **create\_tunnel\_endpoint**(*endpoint=None, tunnel\_interface=None, remote\_address=None*)

Create the and endpoint in tunnel mode or no encryption. If the tunnel mode endpoint is an SMC managed device, both an endpoint and a tunnel interface is required. If the endpoint is externally managed, only an IP address is required.

**Parameters**

- **endpoint** (*InternalEndpoint, ExternalEndpoint*) – the endpoint element for this tunnel endpoint.
- **tunnel\_interface** (*TunnelInterface*) – the tunnel interface for this tunnel endpoint. Required for SMC managed devices.
- **remote\_address** (*str*) – IP address, only required if the tunnel endpoint is a remote gateway.

**Return type***TunnelEndpoint***property endpoint**

Endpoint is used to specify which interface is enabled for VPN. This is the InternalEndpoint property of the InternalGateway.

---

**Note:** This will only return a value if the tunnel type is GRE

---

**Returns**

internal endpoint where VPN is enabled

**Return type***InternalEndpoint, ExternalGateway***property remote\_address**

Show the remote IP address configured for a GRE RBVPN using Tunnel or No Encryption Mode configurations.

**property tunnel\_interface**

Show the tunnel interface for this TunnelEndpoint.

**Returns**

interface for this endpoint

**Return type**

*TunnelInterface*

**class** smc.vpn.route.**TunnelMonitoringGroup**(name=None, \*\*meta)

Bases: *Element*

A tunnel monitoring group is used to group route based VPNs for monitoring on the Home->VPN dashboard.  
:param str name: name tunnel group :param str comment: comment :param object/href link\_usage\_profile : link usage profile to use for rbvpn tunnel group

## 14.12.3 Gateways

### 14.12.3.1 ExternalGateway

VPN Elements are used in conjunction with Policy or Route Based VPN configurations. VPN elements consist of external gateway and VPN site settings that identify 3rd party gateways to be used as a VPN termination endpoint.

There are several ways to create an external gateway configuration. A step by step process which first creates a network element to be used in a VPN site, then creates the ExternalGateway, an ExternalEndpoint for the gateway, and inserts the VPN site into the configuration:

```
Network.create(name='mynetwork', ipv4_network='172.18.1.0/24')
gw = ExternalGateway.create(name='mygw')
gw.external_endpoint.create(name='myendpoint', address='10.10.10.10')
gw.vpn_site.create(name='mysite', site_element=[Network('mynetwork')])
```

You can also use the convenience method *update\_or\_create* on the ExternalGateway to fully provision in a single step. Note that the ExternalEndpoint and VPNSite also have an *update\_or\_create* method to limit the update to those respective configurations (SMC version 6.4.x):

```
>>> from smc.elements.network import Network
>>> from smc.vpn.elements import ExternalGateway
>>> network = Network.get_or_create(name='network-172.18.1.0/24', ipv4_network='172.18.1.
↳ 0/24')
>>>
>>> g = ExternalGateway.update_or_create(name='newgw',
external_endpoint=[
    {'name': 'endpoint1', 'address': '1.1.1.1', 'enabled': True},
    {'name': 'endpoint2', 'address': '2.2.2.2', 'enabled': True}],
vpn_site=[{'name': 'sitea', 'site_element': [network]}])
>>> g
ExternalGateway(name=newgw)
>>> for endpoint in g.external_endpoint:
...     endpoint
...
ExternalEndpoint(name=endpoint1 (1.1.1.1))
ExternalEndpoint(name=endpoint2 (2.2.2.2))
>>> for site in g.vpn_site:
...     site, site.site_element
...
(VPNSite(name=sitea), [Network(name=network-172.18.1.0/24)])
```

In SMC version >= 6.5, you can also provide the *connection\_type\_ref* parameter when defining the external gateway:

.. note:: When calling ``update_or_create`` from the `ExternalGateway`, providing the parameters for `external_endpoints` and `vpn_site` is optional.

**class** `smc.vpn.elements.ExternalGateway`(*name=None, \*\*meta*)

Bases: [\*Element\*](#)

External Gateway defines an VPN Gateway for a non-SMC managed device. This will specify details such as the endpoint IP, and VPN site protected networks. Example of manually provisioning each step:

```
Network.create(name='mynetwork', ipv4_network='172.18.1.0/24')
gw = ExternalGateway.create(name='mygw')
gw.external_endpoint.create(name='myendpoint', address='10.10.10.10')
gw.vpn_site.create(name='mysite', site_element=[Network('mynetwork')])
```

#### Variables

**gateway\_profile** ([\*GatewayProfile\*](#)) – A gateway profile will define the capabilities (i.e. crypto) allowed for this VPN.

**classmethod** `create`(*name, trust\_all\_cas=True, gateway\_profile=None, \*\*kwargs*)

Create new External Gateway

#### Parameters

- **name** (*str*) – name of test\_external gateway
- **trust\_all\_cas** (*bool*) – whether to trust all internal CA's (default: True)
- **gateway\_profile** ([\*GatewayProfile\*](#), *href*) – optional gateway profile, otherwise default

#### Returns

instance with meta

#### Return type

[\*ExternalGateway\*](#)

#### property `external_endpoint`

An External Endpoint is the IP based definition for the destination VPN peers. There may be multiple per External Gateway. Add a new endpoint to an existing test\_external gateway:

```
>>> list(ExternalGateway.objects.all())
[ExternalGateway(name=cisco-remote-side), ExternalGateway(name=remoteside)]
>>> gateway.external_endpoint.create('someendpoint', '12.12.12.12')
'http://1.1.1.1:8082/6.1/elements/external_gateway/22961/external_endpoint/27467'
↪'
```

#### Return type

[\*CreateCollection\(ExternalEndpoint\)\*](#)

#### property `trust_all_cas`

Gateway setting identifying whether all CA's specified in the profile are supported or only specific ones.

#### Return type

*bool*

**classmethod** `update_or_create(name, external_endpoint=None, vpn_site=None, trust_all_cas=True, with_status=False)`

Update or create an ExternalGateway. The `external_endpoint` and `vpn_site` parameters are expected to be a list of dicts with key/value pairs to satisfy the respective elements create constructor. VPN Sites will represent the final state of the VPN site list. ExternalEndpoint that are pre-existing will not be deleted if not provided in the `external_endpoint` parameter, however existing elements will be updated as specified.

#### Parameters

- **name** (*str*) – name of external gateway
- **external\_endpoint** (*list(dict)*) – list of dict items with key/value to satisfy ExternalEndpoint.create constructor
- **vpn\_site** (*list(dict)*) – list of dict items with key/value to satisfy VPNSite.create constructor
- **with\_status** (*bool*) – If set to True, returns a 3-tuple of (ExternalGateway, modified, created), where modified and created is the boolean status for operations performed.

#### Raises

**ValueError** – missing required argument/s for constructor argument

#### Return type

*ExternalGateway*

#### property `vpn_site`

A VPN site defines a collection of IP's or networks that identify address space that is defined on the other end of the VPN tunnel.

#### Return type

*CreateCollection(VPNSite)*

### 14.12.3.2 ExternalEndpoint

**class** `smc.vpn.elements.ExternalEndpoint(**meta)`

Bases: *SubElement*

External Endpoint is used by the External Gateway and defines the IP and other VPN related settings to identify the VPN peer. This is created to define the details of the non-SMC managed gateway. This class is a property of `smc.vpn.elements.ExternalGateway` and should not be called directly. Add an endpoint to existing External Gateway:

```
gw = ExternalGateway('aws')
gw.external_endpoint.create(name='aws01', address='2.2.2.2')
```

Changed in version 0.7.0: When using SMC >= 6.5, you can also provide a value for `ConnectionType` parameter when creating the element, for example:

```
gw = ExternalGateway('aws')
gw.external_endpoint.create(name='aws01', address='2.2.2.2',
                           connection_type=ConnectionType('Active'))
```

See also:

*ConnectionType*

**property connection\_type\_ref**

The reference to the connection type for this external endpoint. A connection type specifies how the endpoint will behave in an active, standby or aggregate role.

---

**Note:** This will return None on SMC versions < 6.5

---

**Return type**

*ConnectionType*

**property contact\_addresses**

Contact Addresses are a mutable collection of contact addresses assigned to a supported element.

**Return type**

ElementContactAddress

**create**(*name*, *address=None*, *enabled=True*, *ipsec\_vpn=True*, *nat\_t=False*, *force\_nat\_t=False*, *dynamic=False*, *ike\_phase1\_id\_type=None*, *ike\_phase1\_id\_value=None*, *connection\_type\_ref=None*, *\*\*kw*)

Create an test\_external endpoint. Define common settings for that specify the address, enabled, nat\_t, name, etc. You can also omit the IP address if the endpoint is dynamic. In that case, you must also specify the ike\_phase1 settings.

**Parameters**

- **name** (*str*) – name of test\_external endpoint
- **address** (*str*) – address of remote host
- **enabled** (*bool*) – True|False (default: True)
- **ipsec\_vpn** (*bool*) – True|False (default: True)
- **nat\_t** (*bool*) – True|False (default: False)
- **force\_nat\_t** (*bool*) – True|False (default: False)
- **dynamic** (*bool*) – is a dynamic VPN (default: False)
- **ike\_phase1\_id\_type** (*int*) – If using a dynamic endpoint, you must set this value. Valid options: 0=DNS name, 1=Email, 2=DN, 3=IP Address
- **ike\_phase1\_id\_value** (*str*) – value of ike\_phase1\_id. Required if ike\_phase1\_id\_type and dynamic set.
- **connection\_type\_ref** (*ConnectionType*) – SMC>=6.5 setting. Specifies the mode for this endpoint; i.e. Active, Aggregate, Standby. (Default: Active)

**Raises**

*CreateElementFailed* – create element with reason

**Returns**

newly created element

**Return type**

*ExternalEndpoint*

**enable\_disable()**

Enable or disable this endpoint. If enabled, it will be disabled and vice versa.

**Returns**

None

**enable\_disable\_force\_nat\_t()**

Enable or disable NAT-T on this endpoint. If enabled, it will be disabled and vice versa.

**Returns**

None

**property enabled**

Whether this endpoint is enabled.

**Return type**

bool

**property force\_nat\_t**

Whether force\_nat\_t is enabled on this endpoint.

**Return type**

bool

**classmethod update\_or\_create**(*external\_gateway*, *name*, *with\_status=False*, *\*\*kw*)

Update or create external endpoints for the specified external gateway. An ExternalEndpoint is considered unique based on the IP address for the endpoint (you cannot add two external endpoints with the same IP). If the external endpoint is dynamic, then the name is the unique identifier.

**Parameters**

- **external\_gateway** ([ExternalGateway](#)) – external gateway reference
- **name** (*str*) – name of the ExternalEndpoint. This is only used as a direct match if the endpoint is dynamic. Otherwise the address field in the keyword arguments will be used as you cannot add multiple external endpoints with the same IP address.
- **with\_status** (*bool*) – If set to True, returns a 3-tuple of (ExternalEndpoint, modified, created), where modified and created is the boolean status for operations performed.
- **kw** (*dict*) – keyword arguments to satisfy ExternalEndpoint.create constructor

**Raises**

- [CreateElementFailed](#) – Failed to create external endpoint with reason
- [ElementNotFound](#) – If specified ExternalGateway is not valid

**Returns**

if with\_status=True, return tuple(ExternalEndpoint, created). Otherwise return only ExternalEndpoint.

## 14.12.4 VPNSite

**class** `smc.vpn.elements.VPNSite`(*\*\*meta*)

Bases: [SubElement](#)

VPN Site information for an internal or test\_external gateway Sites are used to encapsulate hosts or networks as ‘protected’ for VPN policy configuration.

Create a new vpn site for an engine:

```
engine = Engine('myengine')
network = Network('network-192.168.5.0/25') #get resource
engine.vpn.sites.create('newsite', [network.href])
```

Sites can also be added to ExternalGateway's as well:

```
extgw = ExternalGateway('mygw')
extgw.vpn_site.create('newsite', [Network('foo')])
```

This class is a property of *smc.core.engine.InternalGateway* or *smc.vpn.elements.ExternalGateway* and should not be accessed directly.

#### Variables

**gateway** (*InternalGateway*, *ExternalGateway*) – gateway referenced

#### **add\_site\_element**(*element*)

Add a site element or list of elements to this VPN.

##### Parameters

**element** (*list* (*str*, *Network*)) – list of Elements or href's of vpn site elements

##### Raises

*UpdateElementFailed* – fails due to reason

##### Returns

None

#### **create**(*name*, *site\_element*)

Create a VPN site for an internal or external gateway

##### Parameters

- **name** (*str*) – name of site
- **site\_element** (*list* [*str*, *Element*]) – list of protected networks/hosts

##### Raises

*CreateElementFailed* – create element failed with reason

##### Returns

href of new element

##### Return type

*str*

#### **property site\_element**

Site elements for this VPN Site.

##### Returns

Elements used in this VPN site

##### Return type

*list*(*Element*)

#### **classmethod update\_or\_create**(*external\_gateway*, *name*, *site\_element*=None, *with\_status*=False)

Update or create a VPN Site elements or modify an existing VPN site based on value of provided *site\_element* list. The resultant VPN site end result will be what is provided in the *site\_element* argument (can also be an empty list to clear existing).

##### Parameters

- **external\_gateway** (*ExternalGateway*) – The external gateway for this VPN site

- **name** (*str*) – name of the VPN site
- **site\_element** (*list(str, Element)*) – list of resolved Elements to add to the VPN site
- **with\_status** (*bool*) – If set to True, returns a 3-tuple of (VPNSite, modified, created), where modified and created is the boolean status for operations performed.

**Raises**

*ElementNotFound* – ExternalGateway or unresolved site\_element

## 14.12.5 Other Elements

Other elements associated with VPN configurations

### 14.12.5.1 GatewaySettings

**class** smc.vpn.elements.GatewaySettings(*name=None, \*\*meta*)

Bases: *Element*

Gateway settings define various VPN related settings that are applied at the firewall level such as negotiation timers and mobike settings. A gateway setting is a property of an engine:

```
engine = Engine('myfw')
engine.vpn.gateway_settings
```

```
classmethod create(name, negotiation_expiration=200000, negotiation_retry_timer=500,
                  negotiation_retry_max_number=32, negotiation_retry_timer_max=7000,
                  certificate_cache_crl_validity=90000, mobike_after_sa_update=False,
                  mobike_before_sa_update=False, mobike_no_rrc=True)
```

Create a new gateway setting profile.

**Parameters**

- **name** (*str*) – name of profile
- **negotiation\_expiration** (*int*) – expire after (ms)
- **negotiation\_retry\_timer** (*int*) – retry time length (ms)
- **negotiation\_retry\_max\_num** (*int*) – max number of retries allowed
- **negotiation\_retry\_timer\_max** (*int*) – maximum length for retry (ms)
- **certificate\_cache\_crl\_validity** (*int*) – cert cache validity (seconds)
- **mobike\_after\_sa\_update** (*boolean*) – Whether the After SA flag is set for Mobike Policy
- **mobike\_before\_sa\_update** (*boolean*) – Whether the Before SA flag is set for Mobike Policy
- **mobike\_no\_rrc** (*boolean*) – Whether the No RRC flag is set for Mobike Policy

**Raises**

*CreateElementFailed* – failed creating profile

**Returns**

instance with meta

**Return type**

*GatewaySettings*

### 14.12.5.2 GatewayNode

**class** smc.vpn.policy.GatewayNode(\*\*meta)

Bases: *SubElement*

Top level VPN gateway node operations. A gateway node is characterized by a Central Gateway, Satellite Gateway or Mobile Gateway node. This template class will return these as a collection. Gateway Node references need to be obtained from a VPN Policy reference:

```
>>> vpn = PolicyVPN('sg_vm_vpn')
>>> vpn.open()
>>> for gw in vpn.central_gateway_node.all():
...     list(gw.enabled_sites)
...
[GatewayTreeNode(name=Automatic Site for sg_vm_vpn)]
>>> vpn.close()
```

**property** disabled\_sites

Return a collection of VPN Site elements that are disabled for this VPN gateway.

**Return type**

*SubElementCollection(VPNSite)*

**property** enabled\_sites

Return a collection of VPN Site elements that are enabled for this VPN gateway.

**Return type**

*SubElementCollection(VPNSite)*

**property** name

Get the name from the gateway\_profile reference

### 14.12.5.3 GatewayProfile

**class** smc.vpn.elements.GatewayProfile(name=None, \*\*meta)

Bases: *Element*

Gateway Profiles describe the capabilities of a Gateway, i.e. supported cipher, hash, etc. Gateway Profiles of Internal Gateways are read-only and computed from the firewall version and FIPS mode. Gateway Profiles of External Gateways are user-defined.

**property** capabilities

Capabilities are all boolean values that specify features or cryptography features to enable or disable on this gateway profile. To update or change these values, you can use the built in *update* with a key of ‘capabilities’ and dict value of attributes, i.e:

```
gateway_profile = GatewayProfile('myGatewayProfile')
pprint(gateway_profile.capabilities) # <-- show all options
gateway_profile.update(capabilities={'sha2_for_ipsec': True, 'sha2_for_ike':
↪ True})
```

**Return type**

dict

**classmethod create**(*name*, *capabilities=None*, *comment=None*)

Create new GatewayProfile :param str name: name of the gateway profile. :param dict capabilities: Capabilities are all boolean values that specify features or cryptography features to enable or disable on this gateway profile. Following additional boolean attributes can be set:

```
sha2_ike_hash_length sha2_ipsec_hash_length aes128_for_ike aes128_for_ipsec aes256_for_ike
aes256_for_ipsec aes_gcm_256_for_ipsec aes_gcm_for_ipsec aes_xcbc_for_ipsec
aggressive_mode ah_for_ipsec blowfish_for_ike blowfish_for_ipsec des_for_ike
des_for_ipsec dh_group_14_for_ike dh_group_15_for_ike dh_group_16_for_ike
dh_group_17_for_ike dh_group_18_for_ike dh_group_19_for_ike dh_group_1_for_ike
dh_group_20_for_ike dh_group_21_for_ike dh_group_2_for_ike dh_group_5_for_ike
dss_signature_for_ike ecdsa_signature_for_ike esp_for_ipsec external_for_ipsec forward_client_vpn
forward_client_vpn forward_gw_to_gw_vpn ike_v1 ike_v2 ipcomp_deflate_for_ipsec
main_mode md5_for_ike md5_for_ipsec null_for_ipsec pfs_dh_group_14_for_ipsec
pfs_dh_group_15_for_ipsec pfs_dh_group_16_for_ipsec pfs_dh_group_17_for_ipsec
pfs_dh_group_18_for_ipsec pfs_dh_group_19_for_ipsec pfs_dh_group_1_for_ipsec
pfs_dh_group_20_for_ipsec pfs_dh_group_21_for_ipsec pfs_dh_group_2_for_ipsec
pfs_dh_group_5_for_ipsec pre_shared_key_for_ike rsa_signature_for_ike sa_per_host
sa_per_net sha1_for_ike sha1_for_ipsec sha2_for_ike sha2_for_ipsec triple_des_for_ike
triple_des_for_ipsec vpn_client_dss_signature_for_ike vpn_client_ecdsa_signature_for_ike
vpn_client_rsa_signature_for_ike vpn_client_sa_per_host vpn_client_sa_per_net
```

#### Parameters

**comment** (*str*) – comment message

#### Return GatewayProfile

instance with metadata

#### Return type

*GatewayProfile*

### 14.12.5.4 VPNProfile

**class** `smc.vpn.elements.VPNProfile`(*name=None*, *\*\*meta*)

Bases: *Element*

A VPN Profile is used to specify cryptography and other Policy VPN specific features. Every PolicyVPN requires a VPNProfile. The system will provide common profiles labeled as System Element that can be used without modification.

#### property capabilities

Capabilities are all boolean values that specify features or cryptography features to enable or disable on this VPN profile. To update or change these values, you can use the built in *update* with a key of ‘capabilities’ and dict value of attributes, i.e:

```
vpn = VPNProfile('mySecureVPN')
pprint(vpn.capabilities) # <-- show all options
vpn.update(capabilities={'sha2_for_ipsec': True, 'sha2_for_ike': True})
```

#### Return type

dict

```
classmethod create(name, capabilities=None, cn_authentication_for_mobile_vpn=False,
                    disable_anti_replay=False, disable_path_discovery=False,
                    hybrid_authentication_for_mobile_vpn=False, keep_alive=False,
                    preshared_key_for_mobile_vpn=False, sa_life_time=86400,
                    sa_to_any_network_allowed=False, trust_all_cas=True,
                    trusted_certificate_authority=[], tunnel_life_time_kbytes=0,
                    tunnel_life_time_seconds=28800, comment=None, **kw)
```

Create a VPN Profile. There are a variety of kwargs that can be and also retrieved about a VPN Profile. Keyword parameters are specified below. To access a valid keyword, use the standard dot notation.

To validate supported keyword attributes for a VPN Profile, consult the native SMC API docs for your version of SMC. You can also optionally print the element contents after retrieving the element and use *update* to modify the element.

For example:

```
vpn = VPNProfile.create(name='mySecureVPN', comment='test comment')
pprint(vars(vpn.data))
```

Then once identifying the attribute, update the relevant top level attribute values:

```
vpn.update(sa_life_time=128000, tunnel_life_time_seconds=57600)
```

#### Parameters

- **name** (*str*) – Name of profile
- **capabilities** (*dict*) – Capabilities are all boolean values that specify features or cryptography features to enable or disable on this VPN profile. To update or change these values, you can use the built in *update* with a key of ‘capabilities’ and dict value of attributes
- **cn\_authentication\_for\_mobile\_vpn** (*bool*) – Indicates whether CN authentication is allowed or not, for a client using a certificate authentication.
- **disable\_anti\_replay** (*bool*) – IPSEC\_DISABLE\_ANTI\_REPLAY flag value
- **disable\_path\_discovery** (*bool*) – NO\_PATH\_MTU\_DISCOVERY flag value.
- **hybrid\_authentication\_for\_mobile\_vpn** (*bool*) – Indicates whether Hybrid authentication is allowed or not, for a client using a certificate authentication.
- **keep\_alive** (*bool*) – Keep-alive option
- **preshared\_key\_for\_mobile\_vpn** (*bool*) – Indicates whether preshared key authentication is allowed or not, together with a client certificate authentication.,
- **sa\_life\_time** (*int*) – IKE Lifetime in seconds
- **sa\_to\_any\_network\_allowed** (*bool*) – IPSEC\_SA\_To\_ANY flag value.
- **trust\_all\_cas** (*bool*) – Whether to trust all certificate authorities or not
- **trusted\_certificate\_authority** (*list*) – List of trusted certificate authorities for this profile (if the trust all CAs attribute has been set to false)
- **tunnel\_life\_time\_kbytes** (*int*) – IPsec Lifetime in KBytes.
- **tunnel\_life\_time\_seconds** (*int*) – IPsec Lifetime in seconds.
- **comment** (*str*) – optional comment

#### Raises

*CreateElementFailed* – failed creating element with reason

**Return type**  
*VPNProfile*

#### 14.12.5.5 GatewayTreeNode

**class** smc.vpn.policy.GatewayTreeNode(\*\*meta)

Bases: *SubElement*

Gateway Tree node is a list of VPN Site elements returned when retrieving a VPN policies enabled or disabled site list. These provide an enable\_disable link to the VPN site.

```
for gw in policy.central_gateway_node.all():
    for site in list(gw.enabled_sites):
        site.enable_disable()
```

**enable\_disable()**

Enable or disable this VPN Site from within the VPN policy context.

**Raises**

*PolicyCommandFailed* – enabling or disabling failed

**Returns**

None

**property vpn\_site**

The VPN Site element associated with this gateway

:return VPNSite element :rtype: VPNSite

#### 14.12.5.6 GatewayTunnel

**class** smc.vpn.policy.GatewayTunnel(\*\*meta)

Bases: *SubElement*

A gateway tunnel represents the point to point connection between two IPSEC endpoints in a PolicyVPN configuration. The tunnel arrangement is based on whether the nodes are placed as a central gateway or a satellite gateway. This provides access to see the point to point connections, whether the link is enabled, and setting the preshared key.

---

**Note:** Setting the preshared key is only required if using an ExternalGateway element as one side of the VPN. Preshared keys are generated automatically but read only, therefore if two gateways are internally managed by SMC, the key is generated and shared between the gateways automatically. However for external gateways, you must set a new key to provide the same value to the remote gateway.

---

**enable\_disable()**

Enable or disable the tunnel link between endpoints.

**Raises**

*UpdateElementFailed* – failed with reason

**Returns**

None

**property enabled**

Whether the VPN link between endpoints is enabled

**Return type**

`bool`

**property endpoint\_tunnels**

Return all Endpoint tunnels for this gateway tunnel. A tunnel is defined as two end points within the VPN topology. Endpoints are automatically configured based on whether they are a central gateway or satellite gateway. This provides access to enabling/disabling and setting the preshared key for the linked endpoints. List all Endpoint tunnel mappings for this policy vpn:

```
for tunnel in policy.tunnels:
    tunnela = tunnel.tunnel_side_a
    tunnelb = tunnel.tunnel_side_b
    print(tunnela.gateway)
    print(tunnelb.gateway)
    for endpointtunnel in tunnel.endpoint_tunnels:
        print(endpointtunnel)
```

**Return type**

*SubElementCollection(GatewayTunnel)*

**preshared\_key(key)**

Set a new preshared key for the IPSEC endpoints.

**Parameters**

**key** (*str*) – shared secret key to use

**Raises**

*UpdateElementFailed* – fail with reason

**Returns**

`None`

**property tunnel\_side\_a**

Return the gateway node for tunnel side A. This will be an instance of GatewayNode.

**Return type**

*GatewayNode*

**property tunnel\_side\_b**

Return the gateway node for tunnel side B. This will be an instance of GatewayNode.

**Return type**

*GatewayNode*

### 14.12.5.7 ConnectionType

**class** `smc.vpn.elements.ConnectionType`(*name=None, \*\*meta*)

Bases: *Element*

New in version 0.7.0: Introduced in SMC >= 6.5 to provide a way to group VPN element types

ConnectionTypes are used in various VPN configurations such as an ExternalGateway endpoint element to define how the endpoint should be treated, i.e. active, aggregate or standby.

#### Variables

- **connectivity\_group** (*int*) – connectivity group for this connection type
- **mode** (*str*) – mode, valid options: ‘active’, ‘aggregate’, ‘standby’

**classmethod** `create`(*name, mode='active', connectivity\_group=1, link\_type\_ref=None, comment=None*)

Create a connection type for an VPN endpoint.

```
ConnectionType.create(name='mygroup', mode='standby')
```

#### Parameters

- **name** (*str*) – name of connection type
- **mode** (*str*) – mode of connection type, valid options active, standby, aggregate
- **connectivity\_group** (*int*) – integer used to group multiple connection types into a single monitoring group (default: 1)
- **link\_type\_ref** (*str*) – Indicates link type for the connections. Not Required.
- **comment** (*str*) – optional comment

#### Raises

*CreateElementFailed* – reason for failure

#### Return type

*ConnectionType*

### 14.12.5.8 VPNBrokerDomain

**class** `smc.vpn.elements.VPNBrokerDomain`(*name=None, \*\*meta*)

Bases: *Element*

VPN Broker Domain element defines the virtual network that contains the VPN Broker gateway and the VPN Broker members.

**classmethod** `create`(*name, mac\_address\_prefix, file\_ref, comment=None, \*\*kw*)

Create a VPN broker domain.

```
VPNBrokerDomain.create(
    name="vpn1", mac_address_prefix="06:02:02", file_ref=myfile.href
)
```

#### Parameters

- **name** (*str*) – name of vpn broker domain

- **mac\_address\_prefix** (*str*) – unique identifier for the VPN Broker Domain in MAC address format. The length must be three octets. The first octet must be even. The address must be a unique unicast MAC address.

:param file\_ref href or VPNBrokerConfigFile element

**Raises**

*CreateElementFailed* – reason for failure

**Return type**

*VPNBrokerDomain*

#### 14.12.5.9 VPNBrokerConfigFile

**class** smc.vpn.elements.VPNBrokerConfigFile(*name=None, \*\*meta*)

Bases: *Element*

exported VPN Broker Domain element from the NGFW Manager

**classmethod** **create**(*name, comment=None, \*\*kw*)

Create a VPN broker config file

VPNBrokerConfigFile.create(name="myfile")

**Parameters**

**name** (*str*) – name of config file

**Raises**

*CreateElementFailed* – reason for failure

**Return type**

*VPNBrokerConfigFile*

## 14.13 Collections Reference

Collections module provides interfaces to obtain resources from this API and provides searching mechanisms to auto-load resources into the correct class type.

An ElementCollection is bound to *smc.base.model.Element* as the *objects* class property and provides the ability to use an element as the base for iterating elements of that type:

```
for hosts in Host.objects.all():
    ...
```

SubElementCollections are used when references to element data require a fetch from the SMC, but these element references do not have a direct SMC entry point.

See *Collections* for examples on search capabilities.

### 14.13.1 ElementCollection

**class** smc.base.collection.ElementCollection(\*\*params)

ElementCollection is generated dynamically from the CollectionManager and provides methods to obtain data from the SMC. Filters can be chained together to generate more complex queries. Each time a filter is added, a clone is returned to preserve the parent query parameters.

Chaining filters do not affect the parent iterator:

```
>>> iterator = Host.objects.iterator()    <-- Obtain iterator from CollectionManager
>>> query1 = iterator.filter('10.10.10.1')
>>> query1._params, query1._iexact
({'filter': '10.10.10.1', 'exact_match': False, 'filter_context': 'router'}, None)
>>> query2 = query1.limit(2)
>>> query2._params, query2._iexact
({'filter': '10.10.10.1', 'exact_match': False, 'filter_context': 'router', 'limit
→': 2},
 None)
>>> query3 = query2.filter(address='10.10.10.1')
>>> query3._params, query3._iexact
({'filter': '10.10.10.1', 'exact_match': False, 'filter_context': 'router', 'limit
→': 2},
 {'address': '10.10.10.1'})
```

Search operations can access a collection directly through chained syntax:

```
>>> for router in Router.objects.filter('192.168'):
...     print(router)
...
Router(name=router-192.168.19.241)
Router(name=router-192.168.21.241)
Router(name=router-192.168.5.241)
Router(name=router-192.168.15.241)
```

Adding additional filtering via kwargs:

```
>>> print(list(Router.objects.filter(address='10.10.10.1')))
[Router(name=Router-10.10.10.1)]
```

Checking if items from the query exist before accessing:

```
>>> query1 = iterator.filter('10.10.10.1')
>>> if query1.exists():
...     list(query1.all())
...
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10),
 Router(name=Router-10.10.10.1)]
```

Helper methods first, last and exists are provided to simplify retrieving a result from the collection:

```
>>> query1 = iterator.filter('10.10.10.1')
>>> list(query1)
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10),
 Router(name=Router-10.10.10.1)]
```

(continues on next page)

(continued from previous page)

```
>>> query1.first()
Router(name=Router-110.10.10.10)
>>> query1.last()
Router(name=Router-10.10.10.1)
>>> query1.count()
3
>>> query2 = query1.filter(address='10.10.10.1') # change filter to kwarg
>>> list(query2)
[Router(name=Router-10.10.10.1)]
```

---

**Note:** `exists` does not perform filtering when using `filter_key`. Results on `filter(kwarg)` are only done by retrieving the list of results or iterating.

---

### `all()`

Retrieve all elements based on element type. When using the `all` option, any filters are automatically removed.

#### Returns

*ElementCollection*

### `batch(num)`

Iterator returning results in batches. When making more general queries that might have larger results, specify a batch result that should be returned with each iteration.

#### Parameters

**num** (*int*) – number of results per iteration

#### Returns

iterator holding list of results

### `between(start, end)`

Specify a batch of records to return. Start and end correlate to which records to return from a batch. Convenience method to capture only a specific number of records, i.e:

```
>>> objects = situation.objects.between(1, 2)
>>> print(list(objects))
>>>
[InspectionSituation(name=MySQL_Oracle-MySQL-Dumpfile-DLL-Upload)]
```

---

**Note:** Limit is ignored if also chained to the iterator query.

---

#### Parameters

- **start** (*str, int*) – starting record
- **end** (*str, int*) – ending record

#### Returns

*ElementCollection*

### `count()`

Return number of results

**Return type**`int`**exists()**

Returns True if the query contains any results, and False if not. This is handy for checking existence without having to iterate.

```
>>> host = Host.objects.filter('1.1.1.1')
>>> if host.exists():
...     print(host.first())
...
Host(name=hax0r)
```

**Return type**`bool`**filter(\*filter, \*\*kw)**

Filter results for specific element type.

keyword arguments can be used to specify a match against the elements attribute directly. It's important to note that if the search filter contains a / or -, the SMC will only search the name and comment fields. Otherwise other key fields of an element are searched. In addition, SMC searches are a 'contains' search meaning you may return more results than wanted. Use a key word argument to specify the elements attribute and value expected.

```
>>> list(Router.objects.filter('10.10.10.1'))
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10),
 Router(name=Router-10.10.10.1)]
>>> list(Router.objects.filter(address='10.10.10.1'))
[Router(name=Router-10.10.10.1)]
```

**Parameters**

- **filter** (*str*) – any parameter to attempt to match on. For example, if this is a service, you could match on service name 'http' or ports of interest, '80'.
- **exact\_match** (*bool*) – Can be passed as a keyword arg. Specifies whether the match needs to be exact or not (default: False)
- **case\_sensitive** (*bool*) – Can be passed as a keyword arg. Specifies whether the match is case sensitive or not. (default: True)
- **kw** – keyword args can specify an attribute=value to use as an exact match against the elements attribute.

**Returns***ElementCollection***first()**

Returns the first object matched or None if there is no matching object.

```
>>> iterator = Host.objects.iterator()
>>> c = iterator.filter('kali')
>>> if c.exists():
>>>     print(c.count())
>>>     print(c.first())
```

(continues on next page)

(continued from previous page)

```
7
Host(name=kali67)
```

If results are not needed and you only 1 result, this can be called from the `CollectionManager`:

```
>>> Host.objects.first()
Host(name=SMC)
```

**Returns**

element or None

**last()**

Returns the last object matched or None if there is no matching object.

```
>>> iterator = Host.objects.iterator()
>>> c = iterator.filter('kali')
>>> if c.exists():
>>>     print(c.last())
Host(name=kali-foo)
```

**Returns**

element or None

**limit(count)**

Limit provides the ability to limit the number of results returned from the collection.

**Parameters**

**count** (*int*) – number of records to page

**Returns**

*ElementCollection*

**class** smc.base.collection.**CollectionManager**(*resource*)

`CollectionManager` takes a class type as input and dynamically creates an `ElementCollection` for that class. All classes of type `Element` have an `objects` property which returns a manager. You can consume the manager as a re-usable iterator or just called it and it's methods directly.

To get an iterator object that can be re-used, obtain an `iterator()` from the manager:

```
it = Host.objects.iterator()
it.filter(...)
...
```

Or more simply call the managers proxied methods to return the `ElementCollection` for the class type it was called for:

```
>>> from smc.elements.network import Host
>>> for host in Host.objects.all():
...     host
...
Host(name=IGMP v3)
Host(name=ALL-PIM-ROUTERS)
Host(name=Microsoft Lync Online Servers)
...
```

**Returns***CollectionManager***all()**

Retrieve all elements based on element type. When using the `all` option, any filters are automatically removed.

**Returns***ElementCollection***batch(*num*)**

Iterator returning results in batches. When making more general queries that might have larger results, specify a batch result that should be returned with each iteration.

**Parameters**

**num** (*int*) – number of results per iteration

**Returns**

iterator holding list of results

**between(*start*, *end*)**

Specify a batch of records to return. Start and end correlate to which records to return from a batch. Convenience method to capture only a specific number of records, i.e:

```
>>> objects = situation.objects.between(1, 2)
>>> print(list(objects))
>>>
[InspectionSituation(name=MySQL_Oracle-MySQL-Dumpfile-DLL-Upload)]
```

---

**Note:** Limit is ignored if also chained to the iterator query.

---

**Parameters**

- **start** (*str*, *int*) – starting record
- **end** (*str*, *int*) – ending record

**Returns***ElementCollection***filter(*\*filter*, *\*\*kw*)**

Filter results for specific element type.

keyword arguments can be used to specify a match against the elements attribute directly. It's important to note that if the search filter contains a / or -, the SMC will only search the name and comment fields. Otherwise other key fields of an element are searched. In addition, SMC searches are a 'contains' search meaning you may return more results than wanted. Use a key word argument to specify the elements attribute and value expected.

```
>>> list(Router.objects.filter('10.10.10.1'))
[Router(name=Router-110.10.10.10), Router(name=Router-10.10.10.10),
 Router(name=Router-10.10.10.1)]
>>> list(Router.objects.filter(address='10.10.10.1'))
[Router(name=Router-10.10.10.1)]
```

**Parameters**

- **filter** (*str*) – any parameter to attempt to match on. For example, if this is a service, you could match on service name 'http' or ports of interest, '80'.
- **exact\_match** (*bool*) – Can be passed as a keyword arg. Specifies whether the match needs to be exact or not (default: False)
- **case\_sensitive** (*bool*) – Can be passed as a keyword arg. Specifies whether the match is case sensitive or not. (default: True)
- **kw** – keyword args can specify an attribute=value to use as an exact match against the elements attribute.

**Returns***ElementCollection***first()**

Returns the first object matched or None if there is no matching object.

```
>>> iterator = Host.objects.iterator()
>>> c = iterator.filter('kali')
>>> if c.exists():
>>>     print(c.count())
>>>     print(c.first())
7
Host(name=kali67)
```

If results are not needed and you only 1 result, this can be called from the CollectionManager:

```
>>> Host.objects.first()
Host(name=SMC)
```

**Returns**

element or None

**iterator(\*\*kwargs)**

Return an iterator from the collection manager. The iterator can be re-used to chain together filters, each chaining event will be it's own unique element collection.

**Returns***ElementCollection***limit(count)**

Limit provides the ability to limit the number of results returned from the collection.

**Parameters**

**count** (*int*) – number of records to page

**Returns***ElementCollection*

### 14.13.2 SubElementCollection

**class** `smc.base.collection.SubElementCollection(href, cls)`

Collection class providing an iterable interface to sub elements referenced from a top level Element resource. Return types for this collection will be based on the class where the collection was obtained. Elements returned will be serialized into their Element types and only contain the top level meta for each element. The element cache will only be inflated (resulting in an additional query) if an operation is performed that requires the *data* (cache) attribute.

Helper methods are provided to simplify fetching from the collection without having to iterate and code the matching yourself. Fetching from the collection has the limitation that only the returned *name* field is used to find a match (to prevent inflating every element before it is needed). If you want to match an available attribute in the resulting class that requires the elements full json, use a loop to attempt your match.

Example of using SubElementCollection results to obtain matches from the collection:

```
>>> from smc.administration.system import System
>>> system = System()
>>> upgrades = system.engine_upgrade()
>>> upgrades
EngineUpgradeCollection(items: 29)
>>> list(upgrades)
[EngineUpgrade(name=Security Engine upgrade 6.1.2 build 17037 for x86-64),
 EngineUpgrade(name=Security Engine upgrade 6.2.3 build 18067 for x86-64), ...]
>>> upgrades.get(5)
EngineUpgrade(name=Security Engine upgrade 5.8.8 build 12093 for i386)
>>> upgrades.get_contains('6.2')
EngineUpgrade(name=Security Engine upgrade 6.2.3 build 18067 for x86-64)
>>> upgrades.get_contains('6.1')
EngineUpgrade(name=Security Engine upgrade 6.1.2 build 17037 for x86-64)
>>> upgrades.get_all_contains('6.2')
[EngineUpgrade(name=Security Engine upgrade 6.2.3 build 18067 for x86-64),
 EngineUpgrade(name=Security Engine upgrade 6.2.2 build 18062 for x86-64), ...]
>>>
```

#### Raises

***FetchElementFailed*** – If the resource could not be retrieved

#### `all()`

Generator returning collection for sub element types. Return full contents as list or iterate through each.

#### Returns

element type based on collection

#### Return type

*list(SubElement)*

#### `count()`

Return the number of results in this collection

#### Returns

int

#### `get(index)`

Get the element by index. If index is out of bounds for the internal list, None is returned. Indexes cannot be negative.

**Parameters**

**index** (*int*) – retrieve element by positive index in list

**Return type**

*SubElement* or None

**get\_all\_contains**(*value*, *case\_sensitive=True*)

A partial match on the name field. Does an *in* comparison to elements by the meta *name* field. Returns all elements that match the specified value.

**See also:**

[\*get\\_contains\(\)\*](#) for returning only a single item.

**Parameters**

- **value** (*str*) – searchable string for contains match
- **case\_sensitive** (*bool*) – whether the match should consider case (default: True)

**Returns**

element or empty list

**Return type**

*list(SubElement)*

**get\_contains**(*value*, *case\_sensitive=True*)

A partial match on the name field. Does an *in* comparison to elements by the meta *name* field. Sub elements created by SMC will generally have a descriptive name that helps to identify their purpose. Returns only the first entry matched even if there are multiple.

**See also:**

[\*get\\_all\\_contains\(\)\*](#) to return all matches

**Parameters**

- **value** (*str*) – searchable string for contains match
- **case\_sensitive** (*bool*) – whether the match should consider case (default: True)

**Return type**

*SubElement* or None

**get\_exact**(*value*)

Get an element using an exact match based on the elements meta *name* field. The SMC is case sensitive so the name will need to honor the case for a valid value match.

**See also:**

[\*get\\_contains\(\)\*](#) and [\*get\\_all\\_contains\(\)\*](#) for partial matching

**Parameters**

**value** (*str*) – name to match

**Return type**

*SubElement* or None

### 14.13.2.1 CreateCollection

**class** smc.base.collection.CreateCollection(*href*, *cls*)

Bases: [SubElementCollection](#)

A CreateCollection extends SubElementCollection by dynamically proxying the elements *create* method into the collection. This provides a simplified way to create sub elements and also iterate through existing.

For example, obtaining VPN Sites from an engine returns a CreateCollection so existing sites can be iterated while still being able to create new sites:

```
>>> engine = Engine('dingo')
>>> print(engine.vpn.sites)
<smc.base.collection.VPNSite object at 0x1098a9ed0>
>>> print(help(engine.vpn.sites))
Help on VPNSite in module smc.base.collection object:

class VPNSite(CreateCollection)
|   Method resolution order:
|       VPNSite
|       CreateCollection
|       SubElementCollection
|       __builtin__.object
|
|   Methods defined here:
|
|   create(self, name, site_element) from smc.vpn.elements.VPNSite
|       Create a VPN site for an internal or external gateway
|
|       :param str name: name of site
|       :param list site_element: list of protected networks/hosts
|       :type site_element: list[str,Element]
|       :raises CreateElementFailed: create element failed with reason
|       :return: href of new element
|       :rtype: str
|
|   ....
```

List existing sites:

```
list(engine.vpn.sites.all())
```

Creating new VPN sites:

```
engine.vpn.sites.create('mynewsite')
```

**create**(\*args, \*\*kwargs)

The create function from the sub element is proxied by this collections class to provide the iterable functionality from the parent container, but also protected access to the create method of the instance.

### 14.13.2.2 RuleCollection

`smc.base.collection.rule_collection(href, cls)`

Rule collections insert a `create_insert_point` and `create_rule_section` method into the collection. This collection type is returned when accessing rules through a reference, as:

```
policy = FirewallPolicy('mypolicy')
policy.fw_ipv4_access_rules.create(...)
policy.fw_ipv4_access_rules.create_rule_section(...)
policy.fw_ipv4_access_rules.create_insert_point(...)
```

See the class types documentation, or use `help()`:

```
print(help(policy.fw_ipv4_access_rules))
```

#### Return type

*SubElementCollection*

## 14.13.3 Search

**class** `smc.base.collection.Search(**params)`

Bases: *ElementCollection*

Changed in version 0.5.6: Added `entry_point` and `context_filter` chaining to make search syntax the same as direct element object searches.

Search extends *ElementCollection* and provides a way to search for any object by type, as long as there is a valid entry point. Syntax for general searches are the same as initializing a search by a specific element:

```
Search.object_types()      # Get all available search entry points
...
Search.objects.entry_point('ips_alert') # Search for IPS Alerts
...
Search.objects.entry_point('network').filter('1.1.1') # Network with filter
...
Search.objects.context_filter('engine_clusters') # by context filter
...
Search.objects.filter('2.2.2.2') # All element types with filter
...
Search.objects.entry_point('router,host')) # Search using multiple element types
...
Search.objects.entry_point('router,host').filter('2.2.2.2') # with filter
```

Search also provides convenience shortcuts to find duplicate and unused elements:

```
Search.objects.unused()
...
Search.objects.duplicates()
```

If searching a broad range of elements, it is advisable to return results in batches:

```
for batch in Search.objects.batch(100): # All elements search
    ...
```

---

**Note:** If no entry point is specified, the search is done at the ‘elements’ entry point which contains all SMC elements. It is recommended to use `filter` and possibly `batch` to control the result set.

---

**context\_filter**(*context*)

Provide a context filter to search.

**Parameters**

**context** (*str*) – Context filter by name

**duplicates**()

Return duplicate user-created elements.

**Return type**

*list(Element)*

**entry\_point**(*entry\_point*)

Provide an entry point for element types to search.

**Parameters**

**entry\_point** (*str*) – valid entry point. Use *~object\_types()* to find all available entry points.

**static object\_types**()

Show all available ‘entry points’ available for searching. An entry point defines a uri that provides unfiltered access to all elements of the entry point type.

**Returns**

list of entry points by name

**Return type**

*list(str)*

**unused**()

Return unused user-created elements.

**Return type**

*list(Element)*

## 14.13.4 BaselIterable

Common structures

**class** `smc.base.structs.BaseIterable`(*items*)

A collections container that provides a pre-filled container. This container type is used when an element retrieval returns all of an elements data in a single query and will contain multiple values for the same serialized type. Elements can be retrieved from the container through iteration, slicing, or by using *get* and providing either the index or an attribute / value pair.

If subclassing, it may be useful to override *get* to provide a restricted interface to common attributes to fetch.

Examples:

```
>>> for status in engine.nodes[0].interface_status:
...     status
...
InterfaceStatus(aggregate_is_active=False, ...)
```

By index:

```
>>> engine.nodes[0].interface_status[1]
```

Slicing:

```
>>> engine.nodes[0].interface_status[1:5:2]
>>> engine.nodes[0].interface_status[::-1]
```

Using get by index or attribute:

```
>>> engine.nodes[0].interface_status.get(1)
>>> engine.nodes[0].interface_status.get(interface_id=2)
```

#### Parameters

**item** (*iterable*) – items for which to perform iteration. Can be another class with an `__iter__` method also to chain iterators.

#### `all()`

Return the iterable as a list

#### `count()`

Return the number of entries

#### Return type

`int`

#### `get(*args, **kwargs)`

Get an element from the iterable by an arg or kwarg. Args can be a single positional argument that is an index value to retrieve. If the specified index is out of range, `None` is returned. Otherwise use kwargs to provide a key/value. The key is expected to be a valid attribute of the iterated class. For example, to get an element that has a attribute name of 'foo', pass `name='foo'`.

#### Raises

**ValueError** – An argument was missing

#### Returns

the specified item, type is based on what is returned by this iterable, may be `None`

## 14.13.5 SerializedIterable

```
class smc.base.structs.SerializedIterable(items, model)
```

Bases: `BaseIterable`

A pre-serialized list of elements. This is used when it's easier to provide a pre-serialized class as long as all elements are of the same type.

#### Parameters

- **item** (*iterable*) – items for which to perform iteration. Can be another class with an `__iter__` method also to chain iterators.
- **model** – optional class to serialize each iteration.

## 14.14 Monitoring

Module that controls aspects of the getting Monitoring Status To get Monitoring Status for VPN and tunnels, do:

```
>>> vpn = PolicyVPN("Corporate VPN")
>>> status = MonitoringStatus.get_status(href=vpn.href)
>>> print("vpn status monitoring={}".format(status))
vpn status monitoring=VPNStatus=>_health_in_percent = 69, _jitter_in_ms = 3, _latency_in_
↳ms = 71

>>> # get tunnel status in result
>>> for sub_status in status.result:
>>>     sub_status = MonitoringStatus.get_status(href=sub_status.get("href"))
>>>     print("tunnel status monitoring={}".format(sub_status))
...
tunnel status monitoring=VPNGatewayTunnelStatus=>_gatewayA = Helsinki VPN Gateway, _
↳gatewayB =..
...
```

```
class smc.administration.monitoring_status.EngineNodeStatus(**data)
```

Bases: *MonitoringStatus*

This represents the Node Status. For a node, this structure describes the state, version, current policy, ... of the specified Node.

**property active\_policy**

**Returns**

The Currently Active Policy.

**Return type**

*str*

**property active\_policy\_ref**

**Returns**

The reference to Currently Active Policy.

**Return type**

*Policy*

**property configuration\_status**

**Returns**

The Configuration status:

-*Initial* no initial configuration file is yet generated. -*Declared* initial configuration file is generated. -*Configured* initial configuration is done with the engine. -*Installed* policy is installed on the engine. :rtype: *str*

**property dyn\_up**

**Returns**

The dynamic update package received.

**Return type**

*str*

**property engine\_node\_status****Returns**

The engine status (“Not Monitored”/“Unknown”/“Online”/“Going Online”/“Locked Online”/“Going Locked Online”/“Offline”/“Going Offline”/“Locked Offline”/“Going Locked Offline”/“Standby”/“Going Standby”/“No Policy Installed” :rtype: str

**property hw\_info****Returns**

The information about the hardware.

**Return type**

str

**property installed\_policy\_ref****Returns**

The reference to Last Uploaded Policy.

**Return type**

*Policy*

**property last\_upload\_time****Returns**

The Upload Time of the Last Uploaded Policy.

**Return type**

str

**property local\_alternative\_policies****Returns**

The Local Alternative Policies.

**Return type**

list str

**property local\_alternative\_policy\_activation\_time****Returns**

The timestamp when Active Policy was activated, if it is a Local Alternative Policy. :rtype: long

**property master\_node**

Used by Virtual Firewall Node :return: master node. :rtype: MasterNode

**property platform****Returns**

The engine platform.

**Return type**

str

**property version****Returns**

The engine version.

**Return type**

str

```
class smc.administration.monitoring_status.MonitoringStatus(**data)
```

Bases: `object`

MonitoringStatus represents the base class for all statuses returned by `get_monitoring_status(href)` for the element href. Valid attributes (read-only) are:

**Variables**

- **name** – name of the element described by href
- **monitoring\_state** –
  - INITIAL For an Engine Node: Status not yet received.**  
For a Composite Status: Composite Status value is not yet computed.
  - READY For an Engine Node: Status is received.**  
For a Composite Status: Composite Status value is computed.
  - NO\_STATUS For an Engine Node: No status received for the node during initial time-out.**
  - TIMEOUT For an Engine Node: Status was not received in time.**
  - ERROR Not used.**
  - SERVER\_ERROR For an Engine Node: Status poll from log server failed.**
  - DELETED Status is deleted.**
  - DUMMY For an Engine Node: Status is a dummy Status for Demo mode.**  
For a Composite Status: Status is a dummy Status for Demo mode.
  - **monitoring\_status** – ANY Refers to any Status. **NOT\_MONITORED** Indicates that Status element has currently Not Monitored Status. **UNKNOWN For an Engine Node: Node has not sent status (or timeout).**  
For a Composite Status: All children are unknown.
  - OK For an Engine Node: Status is OK.**  
For a Composite Status: All children are OK.
  - PARTIAL\_OK For a Composite Status: Some children are OK and others are unknown.**
  - WARNING For a Composite Status: Some children have warning.**
  - ERROR For an Engine Node: Status is KO.**  
For a Composite Status: Some children are KO.
  - IDLE For a VPN Node: Status is STANDBY.**  
Composite: All children are STANDBY.
- **result** – The API's links of the children or SDWAN Branch Status if exists

```
static get_status(href)
```

Return Monitoring Status for the given href element :param str href: the uri of the element to retrieve the status :raises NotMonitored: the element is not monitored :rtype: MonitoringStatus

**property history**

This function returns a history object. :return: history object. :rtype: History

**property monitoring\_state**

The monitoring state:

- **\*INITIAL\* For an Engine Node: Status not yet received.**  
For a Composite Status: Composite Status value is not yet computed.

**\*READY\*** For an Engine Node: Status is received.

For a Composite Status: Composite Status value is computed.

**\*NO\_STATUS\*** For an Engine Node: No status received for the node during initial timeout. **\*TIMEOUT\***

For an Engine Node: Status was not received in time. **\*ERROR\*** Not used. **\*SERVER\_ERROR\*** For

an Engine Node: Status poll from log server failed. **\*DELETED\*** Status is deleted. **\*DUMMY\*** For an

Engine Node: Status is a dummy Status for Demo mode.

For a Composite Status: Status is a dummy Status for Demo mode.

#### Returns

The monitoring state.

#### Return type

`str`

### property monitoring\_status

The monitoring status:

**\*ANY\*** Refers to any Status. **\*NOT\_MONITORED\*** Indicates that Status element has currently Not Monitored Status. **\*UNKNOWN\*** For an Engine Node: Node has not sent status (or timeout).

For a Composite Status: All children are unknown.

**\*OK\*** For an Engine Node: Status is OK. For a Composite Status: All children are OK. **\*PARTIAL\_OK\***

For a Composite Status: Some children are OK and others are unknown. **\*WARNING\*** For a Composite

Status: Some children have warning. **\*ERROR\*** For an Engine Node: Status is KO. For a Composite

Status: Some children are KO. **\*IDLE\*** For a VPN Node: Status is STANDBY. Composite: All children are STANDBY.

#### Returns

The monitoring status.

#### Return type

`str`

### property name

name of the element described by href

### property result

#### Returns

The API's links of the children or SDWAN Branch Status if exists

#### Return type

`list str`

```
class smc.administration.monitoring_status.SDWANBranchStatus(**data)
```

Bases: *MonitoringStatus*

This represents the SDWAN branch Status.

### property health\_in\_percent

#### Returns

SDWAN Branch Health in percent

#### Return type

`int`

```
class smc.administration.monitoring_status.SDWANBranchToBranchStatus(**data)
```

Bases: *MonitoringStatus*

This represents the SDWAN branch to branch Status.

**property health\_in\_percent**

**Returns**

SDWAN Branch Health in percent

**Return type**

*int*

**property jitter\_in\_ms**

**Returns**

SDWAN Branch to Branch Jitter in milliseconds

**Return type**

*int*

**property latency\_in\_ms**

**Returns**

SDWAN Branch to Branch Latency in milliseconds

**Return type**

*int*

**property packet\_loss\_in\_permyriad**

**Returns**

SDWAN Branch to Branch Packet loss in permyriad (per ten thousands)

**Return type**

*int*

**property traffic\_in\_bits\_per\_sec**

**Returns**

SDWAN Branch to Branch Traffic in bits per second

**Return type**

*long*

```
class smc.administration.monitoring_status.SDWANNetlinkElementStatus(**data)
```

Bases: *MonitoringStatus*

This represents the SDWAN Netlink element Status.

**property connection\_count**

**Returns**

Connection Count

**Return type**

*long*

**property inbound\_traffic\_in\_bits\_per\_sec**

**Returns**

Inbound traffic in bits per second

**Return type**

long

**property** `outbound_traffic_in_bits_per_sec`**Returns**

Outbound traffic in bits per second

**Return type**

long

**property** `packet_loss_in_permyriad`**Returns**

Packet loss in permyriad (per ten thousands)

**Return type**

long

**class** `smc.administration.monitoring_status.ServerStatus(**data)`Bases: [\*MonitoringStatus\*](#)

This represents an SMC Server Status (Class `ServerStatus`).<br/> Field from the StatusDTO map the normal server status.

For an SMC Server, additionnal information is provided about: - replication states ( detailed content depend on server role ) - resource monitoring common to all servers ( memory only currently )

**class** `smc.administration.monitoring_status.VPNEndpointTunnelStatus(**data)`Bases: [\*VPNGatewayTunnelStatus\*](#)

This represents the Monitoring Status for an endpoint tunnel.

**property** `endpointA`**Returns**

Name of Endpoint A

**Return type**[\*str\*](#)**property** `endpointB`**Returns**

Name of Endpoint B

**Return type**[\*str\*](#)**class** `smc.administration.monitoring_status.VPNGatewayTunnelStatus(**data)`Bases: [\*VPNStatus\*](#)

This represents the Monitoring Status for a Gateway Tunnel.

**property** `gatewayA`**Returns**

Name of Gateway A

**Return type**[\*str\*](#)

**property gatewayB**

**Returns**

Name of Gateway B

**Return type**

`str`

**class** `smc.administration.monitoring_status.VPNStatus(**data)`

Bases: `MonitoringStatus`

This represents the VPN Status

**property health\_in\_percent**

**Returns**

VPN Health in percent.

**Return type**

`str`

**property jitter\_in\_ms**

**Returns**

VPN Jitter in milliseconds

**Return type**

`int`

**property latency\_in\_ms**

**Returns**

VPN Latency in milliseconds

**Return type**

`int`

**property packet\_loss\_in\_permyriad**

**Returns**

VPN Packet loss in permyriad (per ten thousands)

**Return type**

`int`

**property traffic\_in\_bits\_per\_sec**

**Returns**

VPN Traffic in bits per second

**Return type**

`long`

**property vpn\_status\_code**

**Returns**

The Monitoring VPN Status Code.

**Return type**

`str`

## 14.15 Advanced Usage

### 14.15.1 SMCRequest

Middle tier helper module to wrap CRUD operations and catch exceptions

SMCRequest is the general data structure that is sent to the `send_request` method in `smc.api.web.SMCConnection` to submit the data to the SMC.

```
class smc.api.common.SMCRequest(href=None, json=None, params=None, filename=None, etag=None,  
                                user_session=None, send_none_json=False, **kwargs)
```

SMCRequest represents the data structure that will be submitted to the web layer for submission to the SMC API.

#### Parameters

- **href** (*str*) – href for request, required by all methods
- **json** (*dict*) – json to submit, required by create, update
- **params** (*dict*) – query string parameters
- **filename** (*str*) – name of file for download, optional for create
- **etag** (*str*) – etag of element, required for update
- **send\_none\_json** (*bool*) – To force json to be none instead of {} set this boolean to True. In this case json param will be ignored.

#### etag

ETag for PUT or DELETE request modifications

#### filename

Filename if a file download is requested

#### headers

Default headers

#### href

href for this request

#### params

dictionary of query parameters

### 14.15.2 SMCResult

Operations being performed that involve REST calls to SMC will return an SMCResult object. This object will hold attributes that are useful to determine if the operation was successful and if not, the reason. An SMCResult is handled automatically and uses exceptions to provide statuses between modules and user interaction. The simplest way to get access to an SMCResult directly is to make an SMCRequest using `smc.base.model.prepared_request()` and observe the attributes in the return message. All response data is serialized into the `SMCResult.json` attribute when it is received by the SMC. Web actions to SMC

SSL certificates are not verified to the CA authority, need to implement for urllib3: <https://urllib3.readthedocs.io/en/latest/user-guide.html#ssl>

**class** `smc.api.web.SMCResult`(*respobj=None, msg=None, user\_session=None*)

SMCResult will store the return data for operations performed against the SMC API. If the function returns an SMCResult, the following attributes are set. Note: SMC API will return a list if searches are done and a dict if the attempt is made to get the element directly from href.

Instance attributes:

#### Variables

- **etag** (*str*) – etag from HTTP GET, representing unique value from server
- **href** (*str*) – href of location header if it exists
- **content** (*str*) – content if return was application/octet
- **msg** (*str*) – error message, if set
- **code** (*int*) – http code
- **json** (*dict*) – element full json

Example of using SMCRequest to fetch an element by href, returning an SMCResult:

```
>>> vars(SMCRequest(href='http://1.1.1.1:8082/6.2/elements/host/978').read())
{'code': 200, 'content': None, 'json': {'comment': u'this is a searchable comment', u
→ 'read_only': False, u'ipv6_address': u'2001:db8:85a3::8a2e:370:7334', u'name': u'kali',
→ u'third_party_monitoring': {'netflow': False, u'snmp_trap': False}, u'system': False,
→ u'link': [{u'href': u'http://1.1.1.1:8082/6.2/elements/host/978', u'type': u'host', u
→ 'rel': u'self'}, {u'href': u'http://1.1.1.1:8082/6.2/elements/host/978/export', u'rel
→ ': u'export'}, {u'href': u'http://1.1.1.1:8082/6.2/elements/host/978/search_category_
→ tags_from_element', u'rel': u'search_category_tags_from_element'}], u'key': 978, u
→ 'address': u'1.1.11.1', u'secondary': [u'7.7.7.7']}, 'href': None, 'etag': '
→ OTc4MzExMzkxNDk2MzI1MTMyMDI4"', 'msg': None}
```

## 14.16 Waiters

Waiters are convenience classes that use blocking or non-blocking threads to monitor for a particular state of an engine node.

A waiter can have a callback added that will be executed after either the state has matched, a number of iterations exceeded or an exception is caught while monitoring. The callback should be a callable that takes a single argument.

They provide the ability to perform logical actions such as “wait for the engine to have status ‘Configured’, then fire a policy upload task”.

Example of waiting for an engine to be ready, then send policy:

```
class ContainerPolicyCallback(object):
    def __init__(self, container):
        self.engine = engine

    def __call__(self, status):
        if status == 'Configured':
            self.engine.upload(policy='MyPolicy')
```

```
engine = Engine('myengine')
callback = ContainerPolicyCallback(engine)
```

(continues on next page)

(continued from previous page)

```
waiter = ConfigurationStatusWaiter(engine.nodes[0], 'Configured')
waiter.add_done_callback(callback)
```

Waiters can also be blocking while waiting for status. Example of using a waiter to block input while waiting for the engine to reach a specific status:

```
waiter = ConfigurationStatusWaiter(node, 'Initial', max_wait=5)
while not waiter.done():
    print("Status after 5 sec wait: %s" % waiter.result(5))
```

```
class smc.core.waiters.ConfigurationStatusWaiter(resource, status, **kw)
```

Bases: *NodeWaiter*

Configuration status waiter provides a current engine status with respects to having a configuration.

#### Parameters

- **resource** (*Node*) – Engine node to check for status
- **status** (*str*) – used defined status to wait for.

#### Raises

*NodeCommandFailed* – Failure to obtain a status back from the engine. This can be thrown when getting initial status. If thrown after the thread has started, it is caught and returned in the result after ending the thread.

```
class smc.core.waiters.NodeStateWaiter(resource, status, **kw)
```

Bases: *NodeWaiter*

Node State specifies where the engine is within it's lifecycle, such as initial state, ready state, error, timeout, etc.

#### Parameters

- **resource** (*Node*) – Engine node to check for status
- **status** (*str*) – used defined status to wait for.

#### Raises

*NodeCommandFailed* – Failure to obtain a status back from the engine. This can be thrown when getting initial status. If thrown after the thread has started, it is caught and returned in the result after ending the thread.

```
class smc.core.waiters.NodeStatusWaiter(resource, status, **kw)
```

Bases: *NodeWaiter*

Node Status specifies the current state of the engine such as offline, online, locked offline, no policy installed, etc.

#### Parameters

- **resource** (*Node*) – Engine node to check for status
- **status** (*str*) – used defined status to wait for.

#### Raises

*NodeCommandFailed* – Failure to obtain a status back from the engine. This can be thrown when getting initial status. If thrown after the thread has started, it is caught and returned in the result after ending the thread.

**class** `smc.core.waiters.NodeWaiter(resource, status, timeout=5, max_wait=36, **kw)`

Bases: `Thread`

Node Waiter provides a common threaded interface to monitoring a nodes status and wait for a specific response.

**add\_done\_callback**(*callback*)

Add a callback to run after the task completes. The callable must take 1 argument which will be the completed Task.

:param callable callback

**done**()

Is the task still running or considered complete

**Return type**

`bool`

**result**(*timeout=None*)

Get current status result after waiting timeout Result does a join on the thread to get a status update. It is possible the first couple of statuses are None if an update has not yet been joined.

**run**()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

**stop**()

Stop thread if it's still running

**wait**(*timeout=None*)

Blocking method to wait for thread

`smc.core.waiters.STATE = frozenset({'DELETED', 'DUMMY', 'ERROR', 'INITIAL', 'NO_STATUS', 'READY', 'SERVER_ERROR', 'TIMEOUT'})`

Node state constant values

`smc.core.waiters.STATUS = frozenset({'Going Locked Offline', 'Going Locked Online', 'Going Offline', 'Going Online', 'Going Standby', 'Locked Offline', 'Locked Online', 'No Policy Installed', 'Not Monitored', 'Offline', 'Online', 'Policy Out Of Date', 'Standby', 'Unknown'})`

Node status constant values

## 14.17 Exceptions

Exceptions thrown throughout smc-python. Be sure to check functions or class methods that have raises documentation. All exception classes subclass `SMCException` Exceptions Module

**exception** `smc.api.exceptions.ActionCommandFailed`

Bases: `SMCException`

Action type commands use this exception

**exception** `smc.api.exceptions.AlertChainError`

Bases: [\*SMCException\*](#)

This exception is related to AlertChain based operations like AlertChain and Alert Chain Rule creation update, and deletion.

**exception** `smc.api.exceptions.AlertPolicyError`

Bases: [\*SMCException\*](#)

This exception is related to AlertPolicy based operations like AlertPolicy and Alert Rule creation update, and deletion.

**exception** `smc.api.exceptions.CertificateError`

Bases: [\*SMCException\*](#)

Related to certificate based operations like requests, signing, or creation. For example, engines that are not initialized can not respond to certificate creation requests and SMC API will return an error.

**exception** `smc.api.exceptions.CertificateExportError`

Bases: [\*CertificateError\*](#)

Failure to export a certificate

**exception** `smc.api.exceptions.CertificateImportError`

Bases: [\*CertificateError\*](#)

Failure to import a certificate or private key

**exception** `smc.api.exceptions.ConfigLoadError`

Bases: [\*SMCException\*](#)

Thrown when there was a problem reading credential information from file. Typically caused by missing settings.

**exception** `smc.api.exceptions.CreateElementFailed`

Bases: [\*SMCException\*](#)

Generic exception when there was a failure calling a create method

**exception** `smc.api.exceptions.CreateEngineFailed`

Bases: [\*SMCException\*](#)

Thrown when a POST operation returns with a failed response. API based response will be returned as the exception message

**exception** `smc.api.exceptions.CreatePolicyFailed`

Bases: [\*SMCException\*](#)

Thrown when failures occur when creating specific policies like Firewall Policy, IPS, VPN, etc.

**exception** `smc.api.exceptions.CreateRuleFailed`

Bases: [\*SMCException\*](#)

Indicates a failed response when creating a rule of any type.

**exception** `smc.api.exceptions.CreateVPNFailed`

Bases: [\*SMCException\*](#)

Creating a policy or route based VPN failed.

**exception** `smc.api.exceptions.DeleteElementFailed`

Bases: [`SMCException`](#)

Used when deletion fails, typically due to dependencies for the target element

**exception** `smc.api.exceptions.EcaClientConfigExportError`

Bases: [`SMCException`](#)

Failure to export an ECA client config

**exception** `smc.api.exceptions.ElementNotFound`

Bases: [`SMCException`](#)

Generic exception when an attempt is made to load an element that is not found.

**exception** `smc.api.exceptions.EngineCommandFailed`

Bases: [`SMCException`](#)

Engines will have some commands that are specifically executed such as adding blacklist entries, flushing blacklist or adding routes. This exception will be thrown if the SMC API responds with any sort of error and wrap the response

**exception** `smc.api.exceptions.FetchElementFailed`

Bases: [`SMCException`](#)

Failure when fetching results

**exception** `smc.api.exceptions.HaCommandException(response=None)`

Bases: [`SMCOperationFailure`](#)

HAManagement will have some commands that are specifically executed, such as `asset_active`, `set_standby`, and `full_replication`. If the SMC API returns an error and wraps the response, this exception will be thrown.

**exception** `smc.api.exceptions.InterfaceNotFound`

Bases: [`SMCException`](#)

Returned when attempting to fetch an interface directly

**exception** `smc.api.exceptions.InvalidLogSeverity`

Bases: [`SMCException`](#)

Used within log option to prevent invalid submissions.

**exception** `smc.api.exceptions.InvalidRuleValue`

Bases: [`SMCException`](#)

Used within rule creation methods to prevent invalid submissions

**exception** `smc.api.exceptions.InvalidSearchFilter`

Bases: [`SMCException`](#)

Thrown by collections when using invalid search sequences.

**exception** `smc.api.exceptions.LicenseError`

Bases: [`SMCException`](#)

Thrown when operations to perform Node specific license related operations such as `bind license`, `fetch license` or `cancel license fail`. For node licensing specific actions, see: `:py:class: smc.core.node.Node`

**exception** `smc.api.exceptions.LoadElementFailed`

Bases: [\*SMCException\*](#)

Failure when attempting to obtain the settings for a specific element. This is more generic for a broad class of elements.

**exception** `smc.api.exceptions.LoadEngineFailed`

Bases: [\*SMCException\*](#)

Thrown when attempting to load an engine that does not exist

**exception** `smc.api.exceptions.LoadPolicyFailed`

Bases: [\*SMCException\*](#)

Failure when trying to load a specific policy type

**exception** `smc.api.exceptions.MissingDependency`

Bases: [\*SMCException\*](#)

A dependency is missing for the given operation.

**exception** `smc.api.exceptions.MissingRequiredInput`

Bases: [\*SMCException\*](#)

Some functions will flat out fail if certain fields are not provided. This is to ensure that some functions have some protection in case the user doesn't read the doc's.

**exception** `smc.api.exceptions.ModificationAborted`

Bases: [\*SMCException\*](#)

A previous requirement was not met which prevented an attempted change from being executed.

**exception** `smc.api.exceptions.ModificationFailed`

Bases: [\*SMCException\*](#)

Used when making generic modifications to elements.

**exception** `smc.api.exceptions.NodeCommandFailed`

Bases: [\*SMCException\*](#)

Each engine node will have multiple commands that can be executed such as go online, go offline, go standby, locking, etc. When these commands fail, this exception will be thrown and wrap the SMC API response. For all node specific command actions, see: `:py:class: smc.core.node.Node`

**exception** `smc.api.exceptions.NotMonitored`

Bases: [\*SMCException\*](#)

Raised when attempting get monitoring status for not monitored element

**exception** `smc.api.exceptions.PolicyCommandFailed`

Bases: [\*SMCException\*](#)

Generic policy related command failures such as opening or closing a VPN policy.

**exception** `smc.api.exceptions.ResourceNotFound`

Bases: [\*SMCException\*](#)

Used to indicate a resource link is not found on the queried node. For example, the `smc.core.engine.Engine` class will expose available resources but some engines may not have those links.

**exception** `smc.api.exceptions.SMCConnectionError`

Bases: [`SMCException`](#)

Thrown when there are connection related issues with the SMC. This could be that the underlying http requests library could not connect due to wrong IP address, wrong port, or time out

**exception** `smc.api.exceptions.SMCException`

Bases: [`Exception`](#)

Base class for exceptions

**exception** `smc.api.exceptions.SMCOperationFailure(response=None)`

Bases: [`SMCException`](#)

Exception class for storing results from calls to the SMC This is thrown for HTTP methods that do not return the expected HTTP status code. See each `http_*` method in [`smc.api.web`](#) for expected success status

#### Parameters

- **response** – response object returned from HTTP method
- **msg** – optional msg to insert

Instance attributes:

#### Variables

- **response** – http request response object
- **code** – http status code
- **status** – status from SMC API
- **message** – message attribute from SMC API
- **details** – details list from SMC API (may not always exist)
- **smcresult** – [`smc.api.web.SMCResult`](#) object for consistent returns

**exception** `smc.api.exceptions.SessionManagerNotFound(message="")`

Bases: [`Exception`](#)

**exception** `smc.api.exceptions.SessionNotFound`

Bases: [`SMCException`](#)

Retrieving a session by name did not succeed because the session did not already exist

**exception** `smc.api.exceptions.TaskRunFailed`

Bases: [`SMCException`](#)

When running tasks such as policy upload, refresh policy, etc, if the result from SMC is a failure, possibly due to an incorrect input (i.e. missing policy), then this exception will be thrown

**exception** `smc.api.exceptions.UnsupportedAlertChannel`

Bases: [`AlertChainError`](#)

The exception occurs when an unsupported alert channel is used.

**exception** `smc.api.exceptions.UnsupportedAttribute`

Bases: [`SMCException`](#)

The exception occurs when an unsupported attribute is used.

**exception** `smc.api.exceptions.UnsupportedEngineFeature`

Bases: *SMCException*

If an operation is performed on an engine that does not support the functionality, this is thrown. For example, only Master Engine has virtual resources. IPS and Layer 2 Firewall do not have internal gateways (used for VPN).

**exception** `smc.api.exceptions.UnsupportedEntryPoint`

Bases: *SMCException*

An entry point was specified that was not found in this API version. This is likely due to using an older version of the SMC API that does not support that feature. The exception is thrown specifying the entry point specified.

**exception** `smc.api.exceptions.UnsupportedInterfaceType`

Bases: *SMCException*

Some interface types are not supported on certain engines. For example, Virtual Engines only have Virtual Physical Interfaces. Layer 3 Firewalls do not support Capture or Inline Interfaces. This exception will be thrown when an attempt is made to enumerate interfaces for an engine type missing a reference to an unsupported interface type

**exception** `smc.api.exceptions.UpdateElementFailed`

Bases: *SMCException*

Failure when updating element. When failure is due to ETag being invalid, target was modified before change was submitted. A resubmit would be required.

**exception** `smc.api.exceptions.UserElementNotFound`

Bases: *SMCException*

Raised when attempting to find a user element that cannot be found in a mapped database (internal or external LDAP)

## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### S

- `smc.administration.certificates.tls`, 128
- `smc.administration.certificates.tls_common`, 126
- `smc.administration.license`, 139
- `smc.administration.monitoring_status`, 515
- `smc.administration.reports`, 153
- `smc.administration.role`, 122
- `smc.administration.scheduled_tasks`, 139
- `smc.administration.system`, 156
- `smc.administration.tasks`, 165
- `smc.administration.upcoming_event`, 168
- `smc.administration.updates`, 169
- `smc.api.common`, 522
- `smc.api.exceptions`, 525
- `smc.api.web`, 522
- `smc.base.collection`, 502
- `smc.base.structs`, 513
- `smc.core.addon`, 294
- `smc.core.advanced_settings`, 386
- `smc.core.collection`, 310
- `smc.core.contact_address`, 355
- `smc.core.engine`, 266
- `smc.core.engines`, 387
- `smc.core.interfaces`, 326
- `smc.core.node`, 358
- `smc.core.resource`, 370
- `smc.core.route`, 371
- `smc.core.sub_interfaces`, 350
- `smc.core.waiters`, 523
- `smc.elements.group`, 204
- `smc.elements.netlink`, 185
- `smc.elements.network`, 174
- `smc.elements.other`, 235
- `smc.elements.profiles`, 262
- `smc.elements.protocols`, 199
- `smc.elements.servers`, 210
- `smc.elements.service`, 193
- `smc.elements.situations`, 256
- `smc.elements.ssm`, 250
- `smc.elements.user`, 116
- `smc.policy.file_filtering`, 437
- `smc.policy.interface`, 435
- `smc.policy.ips`, 443
- `smc.policy.layer2`, 446
- `smc.policy.layer3`, 439
- `smc.policy.policy`, 433
- `smc.policy.qos`, 449
- `smc.policy.rule_elements`, 464
- `smc.policy.rule_nat`, 475
- `smc.routing.access_list`, 405
- `smc.routing.bgp`, 409
- `smc.routing.ospf`, 424
- `smc.routing.prefix_list`, 408
- `smc.routing.route_map`, 400
- `smc.vpn.elements`, 489
- `smc.vpn.route`, 480
- `smc_monitoring.models`, 51
- `smc_monitoring.models.calendar`, 82
- `smc_monitoring.models.constants`, 61
- `smc_monitoring.models.filters`, 52
- `smc_monitoring.models.formats`, 58
- `smc_monitoring.models.formatters`, 81
- `smc_monitoring.models.query`, 47
- `smc_monitoring.models.values`, 56
- `smc_monitoring.monitors`, 85
- `smc_monitoring.monitors.alerts`, 102
- `smc_monitoring.monitors.blacklist`, 85
- `smc_monitoring.monitors.connections`, 88
- `smc_monitoring.monitors.logs`, 90
- `smc_monitoring.monitors.routes`, 93
- `smc_monitoring.monitors.sslvpn`, 95
- `smc_monitoring.monitors.users`, 97
- `smc_monitoring.monitors.vpns`, 99



## INDEX

### A

- `abort()` (*smc.administration.tasks.Task* method), 166
- `ACCELAPSED` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `AccessControllist` (class in *smc.administration.access\_rights*), 115
- `AccessList` (class in *smc.routing.access\_list*), 405
- `AccessListEntry` (class in *smc.routing.access\_list*), 407
- `ACCRXBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `ACCRXPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `ACCTXBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `ACCTXPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `ACK` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `Action` (class in *smc.policy.rule\_elements*), 466
- `action` (*smc.core.engine.LBFilter* property), 289
- `action` (*smc.policy.rule.Rule* property), 450
- `action` (*smc.policy.rule\_nat.NATRule* property), 459
- `action` (*smc.routing.route\_map.RouteMapRule* property), 403
- `ACTION` (*smc\_monitoring.models.constants.LogField* attribute), 62
- `action` (*smc\_monitoring.monitors.alerts.Alert* property), 102
- `ActionCommandFailed`, 525
- `Actions` (class in *smc\_monitoring.models.constants*), 61
- `activate()` (*smc.administration.scheduled\_tasks.TaskSchedule* method), 149
- `activate()` (*smc.administration.updates.PackageMixin* method), 169
- `activated` (*smc.administration.scheduled\_tasks.TaskSchedule* property), 149
- `activation_date` (*smc.administration.scheduled\_tasks.TaskSchedule* property), 149
- `activation_date` (*smc.administration.updates.UpdatePackage* property), 171
- `active_alerts_ack_all()` (*smc.administration.system.System* method), 158
- `active_policy` (*smc.administration.monitoring\_status.EngineNodeStatus* property), 515
- `active_policy_ref` (*smc.administration.monitoring\_status.EngineNodeStatus* property), 515
- `ActiveAlertQuery` (class in *smc\_monitoring.monitors.alerts*), 102
- `add()` (*smc.core.advanced\_settings.LogModeration* method), 387
- `add()` (*smc.core.collection.PhysicalInterfaceCollection* method), 315
- `add()` (*smc.core.engine.IdleTimeout* method), 289
- `add()` (*smc.core.route.Antispoofing* method), 381
- `add()` (*smc.elements.profiles.DNSAnswerTranslation* method), 263
- `add()` (*smc.elements.profiles.DomainSpecificDNSServer* method), 264
- `add()` (*smc.elements.profiles.FixedDomainAnswer* method), 264
- `add()` (*smc.elements.profiles.HostnameMapping* method), 264
- `add()` (*smc.policy.rule\_elements.RuleElement* method), 464
- `add_access_list()` (*smc.routing.route\_map.MatchCondition* method), 401
- `add_alert_rule()` (*smc.elements.alerts.AlertPolicy* method), 125
- `add_and_filter()` (*smc\_monitoring.models.query.Query* method), 47
- `add_application_port()` (*smc.elements.network.URLListApplication* method), 184
- `add_bgp_peering()` (*smc.core.route.Routing* method), 376
- `add_capture_interface()` (*smc.core.collection.PhysicalInterfaceCollection* method), 315
- `add_category()` (*smc.base.model.Element* method), 109
- `add_category()` (*smc.elements.other.Category* method), 238, 246

`add_category_tag()` (*smc.elements.other.Category method*), 238, 246

`add_central_gateway()` (*smc.vpn.policy.PolicyVPN method*), 477

`add_cluster_interface_on_master_engine()` (*smc.core.collection.PhysicalInterfaceCollection method*), 315

`add_cluster_virtual_interface()` (*smc.core.collection.TunnelInterfaceCollection method*), 323

`add_cluster_virtual_interface()` (*smc.core.collection.VPNBrokerInterfaceCollection method*), 324

`add_contact_address()` (*smc.core.contact\_address.ContactAddressNode method*), 356

`add_contact_address()` (*smc.elements.servers.ContactAddressMixin method*), 211

`add_cvi_loopback()` (*smc.core.sub\_interfaces.LoopbackClusterInterface method*), 337, 351

`add_defined_filter()` (*smc\_monitoring.models.query.Query method*), 48

`add_dhcp_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 316

`add_done_callback()` (*smc.administration.tasks.TaskOperationPoller method*), 167

`add_done_callback()` (*smc.core.waiters.NodeWaiter method*), 525

`add_dynamic_gateway()` (*smc.core.route.Routing method*), 377

`add_element()` (*smc.elements.other.Category method*), 238, 246

`add_entry()` (*smc.elements.other.Blacklist method*), 236, 244

`add_entry()` (*smc.elements.other.Blacklist method*), 237

`add_entry()` (*smc.routing.access\_list.AccessList method*), 405

`add_in_filter()` (*smc\_monitoring.models.query.Query method*), 48

`add_inline_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 316

`add_inline_ips_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 317

`add_inline_l2fw_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 317

`add_interface()` (*smc.core.engine.Engine method*), 266

`add_internal_gateway_to_vpn()` (*smc.vpn.policy.PolicyVPN static method*), 478

`add_ip_address()` (*smc.core.interfaces.Interface method*), 326

`add_layer3_cluster_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 318

`add_layer3_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 319

`add_layer3_interface()` (*smc.core.collection.TunnelInterfaceCollection method*), 324

`add_layer3_interface()` (*smc.core.collection.VirtualPhysicalInterfaceCollection method*), 325

`add_layer3_interface()` (*smc.core.collection.VPNBrokerInterfaceCollection method*), 325

`add_layer3_shared_virtual_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 320

`add_layer3_vlan_cluster_interface()` (*smc.core.collection.PhysicalInterfaceCollection method*), 320

`add_layer3_vlan_interface()` (*smc.core.collection.InterfaceCollection method*), 312

`add_link_usage_exception_rules()` (*smc.core.engine.Engine method*), 267

`add_many()` (*smc.policy.rule\_elements.RuleElement method*), 464

`add_metric()` (*smc.routing.route\_map.MatchCondition method*), 402

`add_mobile_gateway()` (*smc.vpn.policy.PolicyVPN method*), 478

`add_netflow_collector()` (*smc.elements.servers.ManagementLogServerMixin method*), 212

`add_next_hop()` (*smc.routing.route\_map.MatchCondition method*), 402

`add_node_loopback()` (*smc.core.sub\_interfaces.LoopbackInterface method*), 336, 352

`add_not_filter()` (*smc\_monitoring.models.query.Query method*), 48

`add_or_filter()` (*smc\_monitoring.models.query.Query method*), 49

`add_ospf_area()` (*smc.core.route.Routing method*), 377

`add_peer_address()` (*smc.routing.route\_map.MatchCondition method*), 402

`add_permission()` (*smc.administration.access\_rights.AccessControlList*

- method*), 115
- `add_permission()` (*smc.elements.user.UserMixin method*), 118
- `add_port_group_interface()` (*smc.core.collection.SwitchInterfaceCollection method*), 322
- `add_route()` (*smc.core.engine.Engine method*), 267
- `add_satellite_gateway()` (*smc.vpn.policy.PolicyVPN method*), 478
- `add_schedule()` (*smc.administration.scheduled\_tasks.ScheduledTask method*), 147
- `add_single()` (*smc.core.sub\_interfaces.LoopbackInterface method*), 337, 352
- `add_site()` (*smc.core.engine.VPN method*), 291
- `add_site_element()` (*smc.vpn.elements.VPNSite method*), 494
- `add_static_route()` (*smc.core.route.Routing method*), 378
- `add_switch_interface()` (*smc.core.collection.SwitchInterfaceCollection method*), 322
- `add_tls_credential()` (*smc.core.addon.TLSInspection method*), 300
- `add_traffic_handler()` (*smc.core.route.Routing method*), 378
- `add_translated_filter()` (*smc\_monitoring.models.query.Query method*), 49
- `add_tunnel_interface()` (*smc.core.collection.VirtualPhysicalInterfaceCollection method*), 325
- `add_x_forwarded_for` (*smc.elements.servers.ProxyServer property*), 225
- `address` (*smc.elements.servers.HttpProxy property*), 224
- `addresses` (*smc.core.contact\_address.InterfaceContactAddress property*), 357
- `addresses` (*smc.core.interfaces.Interface property*), 326
- `addresses` (*smc.elements.other.ContactAddress property*), 240
- `addresses` (*smc.elements.servers.ElasticsearchCluster property*), 234
- `AddressRange` (*class in smc.elements.network*), 176
- `AdminDomain` (*class in smc.administration.system*), 137, 156
- `AdminUser` (*class in smc.elements.user*), 116
- `adsl_interface` (*smc.core.engine.Engine property*), 267
- `agent` (*smc.core.general.SNMP property*), 304
- `aggregate_mode` (*smc.core.interfaces.PhysicalInterface property*), 338
- `aggregation_entry` (*smc.routing.bgp.BGPProfile property*), 418
- `Alert` (*class in smc\_monitoring.monitors.alerts*), 102
- `ALERT` (*smc\_monitoring.models.constants.LogField attribute*), 62
- `alert_server` (*smc.elements.servers.WebPortalServer property*), 228
- `alert_server_ref` (*smc.elements.servers.ManagementServer property*), 219
- `AlertChain` (*class in smc.elements.alerts*), 124
- `AlertChainError`, 525
- `AlertChainRule` (*class in smc.elements.alerts*), 124
- `ALERTCOUNT` (*smc\_monitoring.models.constants.LogField attribute*), 62
- `ALERTTRACE` (*smc\_monitoring.models.constants.LogField attribute*), 62
- `AlertPolicy` (*class in smc.elements.alerts*), 125
- `AlertPolicyError`, 526
- `AlertRule` (*class in smc.elements.alerts*), 125
- `Alerts` (*class in smc\_monitoring.models.constants*), 61
- `ALERTSEVERITY` (*smc\_monitoring.models.constants.LogField attribute*), 62
- `ALERTSTATUS` (*smc\_monitoring.models.constants.LogField attribute*), 62
- `Alias` (*class in smc.elements.network*), 174
- `alias_resolving()` (*smc.core.engine.Engine method*), 267
- `all()` (*smc.base.collection.CollectionManager method*), 507
- `all()` (*smc.base.collection.ElementCollection method*), 504
- `all()` (*smc.base.collection.SubElementCollection method*), 509
- `all()` (*smc.base.structs.BaseIterable method*), 514
- `all()` (*smc.core.route.RoutingTree method*), 373
- `all()` (*smc.elements.profiles.DNSRule method*), 263
- `all()` (*smc.policy.rule\_elements.RuleElement method*), 464
- `all_as_href()` (*smc.policy.rule\_elements.RuleElement method*), 465
- `all_interfaces` (*smc.core.interfaces.Interface property*), 326
- `all_vpn`s (*smc.core.engine.Engine property*), 268
- `allocated_domain_ref` (*smc.core.engine.VirtualResource property*), 385
- `ALLOW` (*smc\_monitoring.models.constants.Actions attribute*), 61
- `ALLOWEDDATATAG` (*smc\_monitoring.models.constants.LogField attribute*), 62
- `AndFilter` (*class in smc\_monitoring.models.filters*), 52
- `announced_networks` (*smc.routing.bgp.BGP property*), 410
- `announcement_enabled` (*smc.administration.system.AdminDomain property*), 137, 157

- announcement\_message  
(*smc.administration.system.AdminDomain* property), 137, 157
  - antispam (*smc.policy.rule\_elements.Action* property), 466
  - Antispoofing (class in *smc.core.route*), 381
  - antispoofing (*smc.core.engine.Engine* property), 268
  - AntiVirus (class in *smc.core.addon*), 294
  - antivirus (*smc.core.engine.Engine* property), 268
  - api\_version (*smc.api.session.Session* property), 105
  - ApiClient (class in *smc.elements.user*), 118
  - append() (*smc.core.general.RankedDNSAddress* method), 302
  - appliance\_info() (*smc.core.node.Node* method), 358
  - appliance\_switch\_module  
(*smc.core.interfaces.SwitchPhysicalInterface* property), 347
  - ApplianceInfo (class in *smc.core.node*), 365
  - ApplianceStatus (class in *smc.core.node*), 366
  - ApplianceSwitchModule (class in *smc.core.hardware*), 367
  - APPLICATION (*smc\_monitoring.models.constants.LogField* attribute), 63
  - application\_logging  
(*smc.policy.rule\_elements.LogOptions* property), 470
  - application\_port (*smc.elements.network.URLListApplication* property), 184
  - APPLICATIONCOMBINATIONFLAGS  
(*smc\_monitoring.models.constants.LogField* attribute), 63
  - APPLICATIONDETAIL (*smc\_monitoring.models.constants.LogField* attribute), 63
  - APPLICATIONUSAGE (*smc\_monitoring.models.constants.LogField* attribute), 63
  - approve\_all() (*smc.core.resource.PendingChanges* method), 371
  - ArchiveLogTask (class in *smc.administration.scheduled\_tasks*), 141
  - arp\_entry (*smc.core.interfaces.PhysicalInterface* property), 339
  - as\_number (*smc.routing.bgp.AutonomousSystem* property), 413
  - as\_tree() (*smc.core.route.RoutingTree* method), 374
  - ASPAEMAILMESSAGEID  
(*smc\_monitoring.models.constants.LogField* attribute), 63
  - ASPAEMAILSCORE (*smc\_monitoring.models.constants.LogField* attribute), 63
  - ASPAEMAILSUBJECT (*smc\_monitoring.models.constants.LogField* attribute), 63
  - ASPAMRECEIVEREMAIL (*smc\_monitoring.models.constants.LogField* attribute), 63
  - ASPAMSENDEREMAIL (*smc\_monitoring.models.constants.LogField* attribute), 63
  - ASPAMSENDERMTA (*smc\_monitoring.models.constants.LogField* attribute), 63
  - ASPathAccessList (class in *smc.routing.bgp\_access\_list*), 421
  - ASPathListEntry (class in *smc.routing.bgp\_access\_list*), 421
  - attacker (*smc.elements.situations.Situation* property), 258
  - auth\_request (*smc.core.interfaces.InterfaceOptions* property), 332
  - authentication\_options (*smc.policy.rule.Rule* property), 450
  - authentication\_options  
(*smc.policy.rule\_nat.NATRule* property), 459
  - authentication\_settings  
(*smc.elements.servers.ElasticsearchCluster* property), 234
  - AUTHENTICATIONCOUNTER  
(*smc\_monitoring.models.constants.LogField* attribute), 63
  - AuthenticationOptions (class in *smc.policy.rule\_elements*), 473
  - AuthenticationServerMixin (class in *smc.elements.servers*), 210
  - AUTHMETHOD (*smc\_monitoring.models.constants.LogField* attribute), 63
  - AUTHNAME (*smc\_monitoring.models.constants.LogField* attribute), 63
  - AUTHRULEID (*smc\_monitoring.models.constants.LogField* attribute), 63
  - auto\_update (*smc.core.addon.ZTNACConnector* property), 298
  - autogenerated (*smc.core.route.Antispoofing* property), 382
  - automatic\_proxy (*smc.policy.rule\_nat.NATElement* property), 475
  - automatic\_rules\_settings (*smc.core.engine.Engine* property), 268
  - AutonomousSystem (class in *smc.routing.bgp*), 413
- ## B
- backup\_heartbeat (*smc.core.interfaces.InterfaceOptions* property), 332
  - backup\_log\_server (*smc.elements.servers.LogServer* property), 217
  - backup\_mgt (*smc.core.interfaces.InterfaceOptions* property), 332
  - balancing\_mode (*smc.elements.servers.ProxyServer* property), 225
  - BALANCINGPROBING (*smc\_monitoring.models.constants.LogField* attribute), 63

- BALANCINGSELECTION (*smc\_monitoring.models.constants* attribute), 63
- BaseIterable (class in *smc.base.structs*), 513
- batch() (*smc.base.collection.CollectionManager* method), 507
- batch() (*smc.base.collection.ElementCollection* method), 504
- between() (*smc.base.collection.CollectionManager* method), 507
- between() (*smc.base.collection.ElementCollection* method), 504
- bgkey (*smc.core.addon.ZTNACConnector* property), 298
- BGP (class in *smc.routing.bgp*), 410
- bgp\_peerings (*smc.core.route.Routing* property), 379
- BGPAggregationEntry (class in *smc.routing.bgp*), 412
- BGPBMPSettings (class in *smc.routing.bgp*), 412
- BGPConnectionProfile (class in *smc.routing.bgp*), 420
- BGPPEering (class in *smc.routing.bgp*), 415
- BGPProfile (class in *smc.routing.bgp*), 418
- bind\_license() (*smc.administration.system.System* method), 158
- bind\_license() (*smc.core.node.Node* method), 358
- Blacklist (class in *smc.elements.other*), 235, 244
- blacklist() (*smc.administration.system.System* method), 158
- blacklist() (*smc.core.engine.Engine* method), 269
- blacklist\_bulk() (*smc.core.engine.Engine* method), 269
- blacklist\_entry\_key (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86
- blacklist\_flush() (*smc.core.engine.Engine* method), 269
- blacklist\_id (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86
- blacklist\_show() (*smc.core.engine.Engine* method), 270
- BlacklistEntry (class in *smc\_monitoring.monitors.blacklist*), 85
- BLACKLISTENTRYDESTINATIONIP (*smc\_monitoring.models.constants.LogField* attribute), 63
- BLACKLISTENTRYDESTINATIONIPMASK (*smc\_monitoring.models.constants.LogField* attribute), 63
- BLACKLISTENTRYDESTINATIONPORT (*smc\_monitoring.models.constants.LogField* attribute), 63
- BLACKLISTENTRYDESTINATIONPORTRANGE (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYDURATION (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYID (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYPROTOCOL (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYSOURCEIP (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYSOURCEIPMASK (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYSOURCEIPPREFIXLEN (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYSOURCEPORT (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTENTRYSOURCEPORTRANGE (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLACKLISTER (*smc\_monitoring.models.constants.LogField* attribute), 64
- BlacklistQuery (class in *smc\_monitoring.monitors.blacklist*), 87
- BLOCK (*smc\_monitoring.models.constants.Actions* attribute), 61
- block\_list() (*smc.administration.system.System* method), 159
- block\_list() (*smc.core.engine.Engine* method), 270
- block\_list\_bulk() (*smc.administration.system.System* method), 159
- block\_list\_bulk() (*smc.core.engine.Engine* method), 270
- block\_list\_flush() (*smc.core.engine.Engine* method), 271
- block\_list\_show() (*smc.core.engine.Engine* method), 271
- BLOCK\_LISTENTRYDESTINATIONIP (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLOCK\_LISTENTRYDESTINATIONIPMASK (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLOCK\_LISTENTRYDESTINATIONPORT (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLOCK\_LISTENTRYDESTINATIONPORTRANGE (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLOCK\_LISTENTRYDURATION (*smc\_monitoring.models.constants.LogField* attribute), 64
- BLOCK\_LISTENTRYID (*smc\_monitoring.models.constants.LogField* attribute), 64

BLOCK\_LISTENTRYPROTOCOL  
    (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 64  
BLOCK\_LISTENTRYSOURCEIP  
    (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 64  
BLOCK\_LISTENTRYSOURCEIPMASK  
    (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 64  
BLOCK\_LISTENTRYSOURCEIPPREFIXLEN  
    (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 65  
BLOCK\_LISTENTRYSOURCEPORT  
    (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 65  
BLOCK\_LISTENTRYSOURCEPORTRANGE  
    (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 65  
BLOCK\_LISTER (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 65  
Blocklist (*class in smc.elements.other*), 236  
bmp\_router\_id (*smc.routing.bgp.BGP property*), 411  
bmp\_router\_id\_type (*smc.routing.bgp.BGP property*),  
    411  
bmp\_settings (*smc.routing.bgp.BGPProfile property*),  
    418  
build\_sub\_expression()  
    (*smc.elements.network.Expression static*  
    *method*), 178  
bypass\_on\_overload()  
    (*smc.core.general.Layer2Settings method*),  
    305  
bytes\_received (*smc\_monitoring.monitors.vpns.VPNSecurityMonitor*  
    *property*), 100  
bytes\_sent (*smc\_monitoring.monitors.vpns.VPNSecurityMonitor*  
    *property*), 100  
**C**  
ca\_for\_signing (*smc.core.addon.ClientInspection*  
    *property*), 298  
call\_route\_map() (*smc.routing.route\_map.RouteMapRule*  
    *method*), 403  
cancel\_unbind\_license() (*smc.core.node.Node*  
    *method*), 359  
capabilities (*smc.vpn.elements.GatewayProfile prop-*  
    *erty*), 496  
capabilities (*smc.vpn.elements.VPNProfile property*),  
    173, 497  
CaptureInterface (*class in smc.core.sub\_interfaces*),  
    350  
categories (*smc.base.model.Element property*), 110  
Category (*class in smc.elements.other*), 237, 245  
category\_filter\_system  
    (*smc.administration.system.AdminDomain*  
    *property*), 137, 157  
CategoryTag (*class in smc.elements.other*), 239, 247  
central\_gateway\_node (*smc.vpn.policy.PolicyVPN*  
    *property*), 478  
certificate\_info() (*smc.core.node.Node method*),  
    359  
CertificateError, 526  
CertificateExportError, 526  
CertificateImportError, 526  
change\_engine\_password()  
    (*smc.elements.user.AdminUser method*),  
    117  
change\_interface\_id()  
    (*smc.core.interfaces.Interface method*), 327  
change\_interface\_id()  
    (*smc.core.sub\_interfaces.InlineInterface*  
    *method*), 351  
change\_interface\_id()  
    (*smc.core.sub\_interfaces.SubInterface*  
    *method*), 354  
change\_password() (*smc.elements.user.UserMixin*  
    *method*), 119  
change\_ssh\_pwd() (*smc.core.node.Node method*), 359  
change\_vlan\_id() (*smc.core.interfaces.PhysicalInterface*  
    *method*), 339  
change\_vlan\_id() (*smc.core.interfaces.VlanInterface*  
    *method*), 349  
change\_vlan\_id() (*smc.core.sub\_interfaces.InlineInterface*  
    *method*), 351  
change\_vlan\_id() (*smc.core.sub\_interfaces.SubInterface*  
    *method*), 354  
ChangeRecord (*class in smc.core.resource*), 370  
change\_server\_port (*smc.elements.servers.LogServer prop-*  
    *erty*), 217  
chassis\_id (*smc.core.lldp.LLDPProfile property*), 265  
CILikeFilter (*class in smc\_monitoring.models.filters*),  
    53  
CIPHERALG (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 65  
ciphers() (*smc.administration.certificates.tls.TLSCryptographySuite*  
    *static method*), 134  
clean\_invalid\_filters()  
    (*smc.administration.system.System method*),  
    160  
clear\_text (*smc.elements.servers.TacacsServer prop-*  
    *erty*), 232  
client\_inspection (*smc.core.engine.Engine prop-*  
    *erty*), 271  
ClientInspection (*class in smc.core.addon*), 298  
CLIENTIPADDRESS (*smc\_monitoring.models.constants.LogField*  
    *attribute*), 65  
ClientProtectionCA (*class in*  
    *smc.administration.certificates.tls*), 135  
close() (*smc.vpn.policy.PolicyVPN method*), 479

cloud\_sync (*smc.core.node.HardwareStatus* property), 368

CloudSGSingleFW (class in *smc.core.engines*), 399

ClusterPhysicalInterface (class in *smc.core.interfaces*), 345

ClusterVirtualInterface (class in *smc.core.sub\_interfaces*), 350

CollectionManager (class in *smc.base.collection*), 506

CombinedFormat (class in *smc\_monitoring.models.formats*), 59

comment (*smc.base.model.Element* property), 110

comment (*smc.core.engine.LinkUsageExceptionRules* property), 290

comment (*smc.core.interfaces.Interface* property), 327

comment (*smc.policy.rule.Rule* property), 450

comment (*smc.routing.route\_map.RouteMapRule* property), 403

CommunityAccessList (class in *smc.routing.bgp\_access\_list*), 422

CommunityListEntry (class in *smc.routing.bgp\_access\_list*), 422

compare\_to\_latest() (*smc.core.resource.Snapshot* method), 384

COMPID (*smc\_monitoring.models.constants.LogField* attribute), 65

Condition (class in *smc.routing.route\_map*), 405

ConfigLoadError, 526

configuration\_status (*smc.administration.monitoring\_status.EngineNodeStatus* property), 515

configuration\_status (*smc.core.node.ApplianceStatus* property), 366

ConfigurationStatusWaiter (class in *smc.core.waiters*), 524

CONNDIRECTION (*smc\_monitoring.models.constants.LogField* attribute), 65

connect\_retry (*smc.routing.bgp.BGPConnectionProfile* property), 420

CONNECTEDMACADDR (*smc\_monitoring.models.constants.LogField* attribute), 65

Connection (class in *smc\_monitoring.monitors.connections*), 88

connection\_count (*smc.administration.monitoring\_status.SDWANNNetworkElementStatus* property), 519

connection\_sync\_mode (*smc.core.engine.HAForSingleEngine* property), 288

connection\_timeout (*smc.core.engine.Engine* property), 272

connection\_tracking() (*smc.core.general.Layer2Settings* method), 305

connection\_type\_ref (*smc.vpn.elements.ExternalEndpoint* property), 491

ConnectionQuery (class in *smc\_monitoring.monitors.connections*), 90

ConnectionTracking (class in *smc.policy.rule\_elements*), 469

ConnectionType (class in *smc.vpn.elements*), 501

CONNECTIVITY (*smc\_monitoring.models.constants.LogField* attribute), 65

CONNSTATUS (*smc\_monitoring.models.constants.LogField* attribute), 65

CONNTYPE (*smc\_monitoring.models.constants.LogField* attribute), 65

ConstantValue (class in *smc\_monitoring.models.values*), 56

contact\_addresses (*smc.core.engine.Engine* property), 272

contact\_addresses (*smc.core.interfaces.Interface* property), 327

contact\_addresses (*smc.elements.servers.ContactAddressMixin* property), 211

contact\_addresses (*smc.vpn.elements.ExternalEndpoint* property), 492

contact\_number (*smc.administration.system.AdminDomain* property), 137, 157

ContactAddress (class in *smc.elements.other*), 240

ContactAddressCollection (class in *smc.core.contact\_address*), 355

ContactAddressMixin (class in *smc.elements.servers*), 211

ContactAddressNode (class in *smc.core.contact\_address*), 356

CONTAINEDDATATAG (*smc\_monitoring.models.constants.LogField* attribute), 65

contains() (*smc.core.advanced\_settings.LogModeration* method), 387

context\_filter() (*smc.base.collection.Search* method), 513

CONTROLCOMMANDID (*smc\_monitoring.models.constants.LogField* attribute), 65

CorrelationSituation (class in *smc.elements.situations*), 257

CorrelationSituationContext (class in *smc.elements.situations*), 257

count() (*smc.base.collection.ElementCollection* method), 504

count() (*smc.base.collection.SubElementCollection* method), 509

count() (*smc.base.structs.BaseIterable* method), 514

create() (*smc.administration.access\_rights.AccessControlList* class method), 115

create() (*smc.administration.access\_rights.Permission* class method), 121

create() (*smc.administration.certificates.tls.TLSCertificateAuthority* class method), 121

class method), 135  
 create() (smc.administration.certificates.tls.TLSCryptographySuite class method), 134  
 create() (smc.administration.certificates.tls.TLSProfile class method), 133  
 create() (smc.administration.certificates.tls.TLSServerCertificate class method), 130  
 create() (smc.administration.reports.ReportOperation class method), 155  
 create() (smc.administration.role.Role class method), 123  
 create() (smc.administration.scheduled\_tasks.ArchiveLogTask class method), 141  
 create() (smc.administration.scheduled\_tasks.DeleteLogTask class method), 142  
 create() (smc.administration.scheduled\_tasks.ExportLogTask class method), 143  
 create() (smc.administration.scheduled\_tasks.RefreshMasterEnginePolicyTask class method), 144  
 create() (smc.administration.scheduled\_tasks.RefreshPolicyTask class method), 144  
 create() (smc.administration.scheduled\_tasks.RemoteUpgradeTask class method), 145  
 create() (smc.administration.scheduled\_tasks.ServerBackupTask class method), 148  
 create() (smc.administration.scheduled\_tasks.SGInfoTask class method), 146  
 create() (smc.administration.scheduled\_tasks.TrafficCaptureTask class method), 149  
 create() (smc.administration.scheduled\_tasks.TrafficCaptureTask class method), 150  
 create() (smc.administration.scheduled\_tasks.UploadPolicyTask class method), 151  
 create() (smc.administration.scheduled\_tasks.ValidatePolicyTask class method), 151  
 create() (smc.administration.system.AdminDomain class method), 137, 157  
 create() (smc.base.collection.CreateCollection method), 511  
 create() (smc.core.engine.VirtualResource method), 386  
 create() (smc.core.engines.FirewallCluster class method), 394  
 create() (smc.core.engines.IPS class method), 387  
 create() (smc.core.engines.Layer2Firewall class method), 392  
 create() (smc.core.engines.Layer3Firewall class method), 388  
 create() (smc.core.engines.Layer3VirtualEngine class method), 393  
 create() (smc.core.engines.MasterEngine class method), 397  
 create() (smc.core.engines.MasterEngineCluster class method), 398  
 create() (smc.core.general.NTPSettings class method), 306  
 create() (smc.core.lldp.LLDPProfile class method), 265  
 create() (smc.core.route.PolicyRoute method), 383  
 create() (smc.elements.alerts.AlertChain class method), 124  
 create() (smc.elements.alerts.AlertChainRule static method), 124  
 create() (smc.elements.alerts.AlertPolicy class method), 125  
 create() (smc.elements.alerts.AlertRule static method), 125  
 create() (smc.elements.group.EthernetServiceGroup class method), 209  
 create() (smc.elements.group.Group class method), 207  
 create() (smc.elements.group.ICMPServiceGroup class method), 205  
 create() (smc.elements.group.IPServiceGroup class method), 206  
 create() (smc.elements.group.RpcServiceGroup class method), 206  
 create() (smc.elements.group.ServiceGroup class method), 208  
 create() (smc.elements.group.TCPServiceGroup class method), 208  
 create() (smc.elements.group.UDPServiceGroup class method), 209  
 create() (smc.elements.netlink.DynamicNetlink class method), 187  
 create() (smc.elements.netlink.Multilink class method), 188  
 create() (smc.elements.netlink.MultilinkMember class method), 190  
 create() (smc.elements.netlink.StaticNetlink class method), 191  
 create() (smc.elements.network.AddressRange class method), 177  
 create() (smc.elements.network.Alias class method), 175  
 create() (smc.elements.network.DomainName class method), 177  
 create() (smc.elements.network.Expression class method), 178  
 create() (smc.elements.network.Host class method), 179  
 create() (smc.elements.network.IPList class method), 180  
 create() (smc.elements.network.Network class method), 182  
 create() (smc.elements.network.Router class method), 183  
 create() (smc.elements.network.URLListApplication

*class method*), 184  
`create()` (*smc.elements.network.Zone class method*), 185  
`create()` (*smc.elements.other.Category class method*), 238, 246  
`create()` (*smc.elements.other.CategoryTag class method*), 239, 247  
`create()` (*smc.elements.other.Location class method*), 240, 248  
`create()` (*smc.elements.other.LogicalInterface class method*), 241, 249  
`create()` (*smc.elements.other.MacAddress class method*), 241, 249  
`create()` (*smc.elements.other.UpdateServerProfile class method*), 242  
`create()` (*smc.elements.servers.AuthenticationServerMixin class method*), 210  
`create()` (*smc.elements.servers.DHCP Server class method*), 227  
`create()` (*smc.elements.servers.DNS Server class method*), 223  
`create()` (*smc.elements.servers.ElasticsearchCluster class method*), 234  
`create()` (*smc.elements.servers.Epo Server class method*), 229  
`create()` (*smc.elements.servers.HttpProxy class method*), 224  
`create()` (*smc.elements.servers.Icap Server class method*), 231  
`create()` (*smc.elements.servers.Log Server class method*), 217  
`create()` (*smc.elements.servers.ManagementServer class method*), 219  
`create()` (*smc.elements.servers.NTP Server class method*), 233  
`create()` (*smc.elements.servers.Proxy Server class method*), 225  
`create()` (*smc.elements.servers.SmtP Server class method*), 230  
`create()` (*smc.elements.servers.TlsSettings class method*), 216  
`create()` (*smc.elements.servers.WebApp class method*), 216  
`create()` (*smc.elements.servers.WebPortalServer class method*), 228  
`create()` (*smc.elements.service.EthernetService class method*), 193  
`create()` (*smc.elements.service.ICMP IPv6 Service class method*), 196  
`create()` (*smc.elements.service.ICMP Service class method*), 195  
`create()` (*smc.elements.service.IP Service class method*), 196  
`create()` (*smc.elements.service.RPC Service class method*), 194  
`create()` (*smc.elements.service.TCP Service class method*), 197  
`create()` (*smc.elements.service.UDP Service class method*), 198  
`create()` (*smc.elements.situations.InspectionSituation class method*), 258  
`create()` (*smc.elements.situations.SubTlsMatchSituation class method*), 260  
`create()` (*smc.elements.situations.TlsMatchSituation class method*), 261  
`create()` (*smc.elements.ssm.SSHKnownHosts class method*), 251, 254  
`create()` (*smc.elements.ssm.SSHKnownHostsLists class method*), 251, 255  
`create()` (*smc.elements.ssm.SSHProfile class method*), 252, 253  
`create()` (*smc.elements.user.AdminUser class method*), 117  
`create()` (*smc.elements.user.ApiClient class method*), 118  
`create()` (*smc.elements.user.WebPortalAdminUser class method*), 120  
`create()` (*smc.policy.file\_filtering.FileFilteringPolicy class method*), 437  
`create()` (*smc.policy.file\_filtering.FileFilteringRule class method*), 437  
`create()` (*smc.policy.interface.InterfacePolicy class method*), 435  
`create()` (*smc.policy.interface.InterfaceSubPolicy class method*), 436  
`create()` (*smc.policy.ips.IPSPolicy class method*), 444  
`create()` (*smc.policy.ips.IPSSubPolicy class method*), 445  
`create()` (*smc.policy.layer2.Layer2Policy class method*), 447  
`create()` (*smc.policy.layer3.FirewallPolicy class method*), 440  
`create()` (*smc.policy.layer3.FirewallSubPolicy class method*), 441, 449  
`create()` (*smc.policy.qos.QoSClass class method*), 449  
`create()` (*smc.policy.rule.EthernetRule method*), 458  
`create()` (*smc.policy.rule.IPv4Layer2Rule method*), 456  
`create()` (*smc.policy.rule.IPv4Rule method*), 454  
`create()` (*smc.policy.rule\_elements.MatchExpression class method*), 474  
`create()` (*smc.policy.rule\_nat.IPv4NATRule method*), 462  
`create()` (*smc.routing.access\_list.AccessList class method*), 406  
`create()` (*smc.routing.bgp.AutonomousSystem class method*), 413  
`create()` (*smc.routing.bgp.BGP AggregationEntry class method*), 412

`create()` (*smc.routing.bgp.BGPConnectionProfile* class method), 420

`create()` (*smc.routing.bgp.BGPPeering* class method), 415

`create()` (*smc.routing.bgp.BGPProfile* class method), 418

`create()` (*smc.routing.bgp.ExternalBGPPeer* class method), 415

`create()` (*smc.routing.bgp.RedistributionEntry* class method), 413

`create()` (*smc.routing.ospf.OSPFArea* class method), 427

`create()` (*smc.routing.ospf.OSPFDomainSetting* class method), 431

`create()` (*smc.routing.ospf.OSPFInterfaceSetting* class method), 432

`create()` (*smc.routing.ospf.OSPFKeyChain* class method), 428

`create()` (*smc.routing.ospf.OSPFProfile* class method), 429

`create()` (*smc.routing.route\_map.RouteMap* class method), 402

`create()` (*smc.routing.route\_map.RouteMapRule* class method), 403

`create()` (*smc.vpn.elements.ConnectionType* class method), 501

`create()` (*smc.vpn.elements.ExternalEndpoint* method), 492

`create()` (*smc.vpn.elements.ExternalGateway* class method), 490

`create()` (*smc.vpn.elements.GatewayProfile* class method), 496

`create()` (*smc.vpn.elements.GatewaySettings* class method), 495

`create()` (*smc.vpn.elements.LinkUsageProfile* class method), 172

`create()` (*smc.vpn.elements.VPNBrokerConfigFile* class method), 502

`create()` (*smc.vpn.elements.VPNBrokerDomain* class method), 501

`create()` (*smc.vpn.elements.VPNProfile* class method), 173, 497

`create()` (*smc.vpn.elements.VPNSite* method), 494

`create()` (*smc.vpn.policy.PolicyVPN* class method), 479

`create_bulk()` (*smc.core.engines.FirewallCluster* class method), 397

`create_bulk()` (*smc.core.engines.Layer3Firewall* class method), 389

`create_csr()` (*smc.administration.certificates.tls.TLSServerCertificate* class method), 130

`create_design()` (*smc.administration.reports.ReportTemplate* method), 156

`create_dynamic()` (*smc.core.engines.CloudSGSingleFW* class method), 399

`create_dynamic()` (*smc.core.engines.Layer3Firewall* class method), 391

`create_geneve_mode()` (*smc.vpn.route.RouteVPN* class method), 483

`create_gre_transport_endpoint()` (*smc.vpn.route.TunnelEndpoint* class method), 487

`create_gre_transport_mode()` (*smc.vpn.route.RouteVPN* class method), 484

`create_gre_tunnel_endpoint()` (*smc.vpn.route.TunnelEndpoint* class method), 487

`create_gre_tunnel_mode()` (*smc.vpn.route.RouteVPN* class method), 484

`create_gre_tunnel_no_encryption()` (*smc.vpn.route.RouteVPN* class method), 485

`create_interface_setting_for_all_node()` (*smc.administration.scheduled\_tasks.TrafficCaptureInterfaceSetting* class method), 150

`create_internal_gateway()` (*smc.core.engine.Engine* method), 273

`create_ipsec_endpoint()` (*smc.vpn.route.TunnelEndpoint* class method), 488

`create_ipsec_tunnel()` (*smc.vpn.route.RouteVPN* class method), 485

`create_regular_expression()` (*smc.elements.situations.InspectionSituation* method), 258

`create_rule_section()` (*smc.policy.rule.EthernetRule* method), 458

`create_rule_section()` (*smc.policy.rule.IPv4Layer2Rule* method), 457

`create_rule_section()` (*smc.policy.rule.IPv4Rule* method), 455

`create_rule_section()` (*smc.policy.rule\_nat.IPv4NATRule* method), 463

`create_self_signed()` (*smc.administration.certificates.tls.ClientProtectionCA* class method), 136

`create_self_signed()` (*smc.administration.certificates.tls.TLSServerCredential* class method), 130

`create_tunnel_endpoint()` (*smc.vpn.route.TunnelEndpoint* class method), 488

`create_with_netlinks()` (*smc.elements.netlink.Multilink* class method), 189

- CreateCollection (class in *smc.base.collection*), 511  
 created\_by (*smc.core.resource.History* property), 114  
 CreateElementFailed, 526  
 CreateEngineFailed, 526  
 CreatePolicyFailed, 526  
 CreateRuleFailed, 526  
 CreateVPNFailed, 526  
 creation\_time (*smc.administration.reports.Report* property), 154  
 CRITICAL (*smc\_monitoring.models.constants.Alerts* attribute), 61  
 CSLikeFilter (class in *smc\_monitoring.models.filters*), 53  
 CSVFormat (class in *smc\_monitoring.models.formatters*), 82  
 custom\_range() (*smc\_monitoring.models.calendar.TimeFormat* method), 83  
 cvi\_mode (*smc.core.interfaces.ClusterPhysicalInterface* property), 346
- ## D
- data\_context (*smc.elements.servers.NetflowCollector* property), 215  
 datacenter (*smc.core.addon.ZTNACConnector* property), 298  
 DataContext (class in *smc.elements.servers*), 212  
 DATATAG (*smc\_monitoring.models.constants.LogField* attribute), 65  
 DATATAGS (*smc\_monitoring.models.constants.LogField* attribute), 65  
 DataType (class in *smc\_monitoring.models.constants*), 62  
 DATATYPE (*smc\_monitoring.models.constants.LogField* attribute), 65  
 datetime\_from\_ms() (in module *smc\_monitoring.models.calendar*), 84  
 datetime\_to\_ms() (in module *smc\_monitoring.models.calendar*), 84  
 db\_replication (*smc.elements.servers.ManagementServer* property), 221  
 Debug (class in *smc.core.node*), 369  
 debug() (*smc.core.node.Node* method), 359  
 decrypting (*smc.policy.rule\_elements.Action* property), 466  
 deep\_inspection (*smc.policy.rule\_elements.Action* property), 467  
 default\_link\_balancing\_preference (*smc.vpn.elements.LinkUsageProfile* property), 172  
 default\_nat (*smc.core.engine.Engine* property), 273  
 DefaultNAT (class in *smc.core.general*), 301  
 DefinedFilter (class in *smc\_monitoring.models.filters*), 53  
 delete() (*smc.base.model.ElementBase* method), 108  
 delete() (*smc.core.contact\_address.ContactAddressNode* method), 356  
 delete() (*smc.core.interfaces.Interface* method), 328  
 delete() (*smc.core.interfaces.VlanInterface* method), 349  
 delete() (*smc.core.route.PolicyRoute* method), 384  
 delete() (*smc.core.route.RoutingTree* method), 374  
 delete() (*smc.core.sub\_interfaces.LoopbackClusterInterface* method), 338, 352  
 delete() (*smc.core.sub\_interfaces.LoopbackInterface* method), 337, 353  
 delete() (*smc.elements.servers.MultiContactAddress* method), 214  
 delete() (*smc\_monitoring.monitors.blacklist.BlacklistEntry* method), 86  
 delete\_invalid\_route() (*smc.core.interfaces.Interface* method), 328  
 delete\_license() (*smc.administration.system.System* method), 160  
 delete\_user\_authentication\_certificate() (*smc.core.engine.Engine* method), 274  
 delete\_user\_authentication\_certificate\_request() (*smc.core.engine.Engine* method), 274  
 DeleteElementFailed, 526  
 DeleteLogTask (class in *smc.administration.scheduled\_tasks*), 142  
 DeleteOldRunTask (class in *smc.administration.scheduled\_tasks*), 142  
 DeleteOldSnapshotsTask (class in *smc.administration.scheduled\_tasks*), 142  
 description (*smc.elements.protocols.ProtocolParameterValue* property), 203  
 description (*smc.elements.situations.Situation* property), 259  
 description (*smc.elements.situations.SituationContext* property), 259  
 dest\_addr (*smc\_monitoring.monitors.connections.Connection* property), 88  
 dest\_if (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
 dest\_port (*smc\_monitoring.monitors.connections.Connection* property), 88  
 dest\_ports (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86  
 dest\_vlan (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
 dest\_zone (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
 Destination (class in *smc.policy.rule\_elements*), 466  
 destination (*smc\_monitoring.monitors.alerts.Alert* property), 102  
 destination (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86  
 destination\_port (*smc\_monitoring.monitors.alerts.Alert*

- property*), 103
- destinations* (*smc.policy.rule.Rule* *property*), 450
- DetailedFormat (class in *smc\_monitoring.models.formats*), 59
- DeviceToolsProfile (class in *smc.elements.ssm*), 250
- DHCLEASEEXPIRES (*smc\_monitoring.models.constants.LogField* *attribute*), 65
- DHCLEASEGW (*smc\_monitoring.models.constants.LogField* *attribute*), 65
- DHCLEASEIP (*smc\_monitoring.models.constants.LogField* *attribute*), 66
- DHCLEASENETMASK (*smc\_monitoring.models.constants.LogField* *attribute*), 66
- DHCLEASEPREFIXLEN (*smc\_monitoring.models.constants.LogField* *attribute*), 66
- DHCLEASERECEIVED (*smc\_monitoring.models.constants.LogField* *attribute*), 66
- DHCLEASES (*smc\_monitoring.models.constants.LogField* *attribute*), 66
- DHCPServer (class in *smc.elements.servers*), 227
- disable() (*smc.administration.role.Role* *method*), 123
- disable() (*smc.core.addon.AntiVirus* *method*), 295
- disable() (*smc.core.addon.ClientInspection* *method*), 298
- disable() (*smc.core.addon.FileReputation* *method*), 296
- disable() (*smc.core.addon.Sandbox* *method*), 299
- disable() (*smc.core.addon.SidewinderProxy* *method*), 296
- disable() (*smc.core.addon.UrlFiltering* *method*), 297
- disable() (*smc.core.addon.ZTNAConnector* *method*), 298
- disable() (*smc.core.general.DefaultNAT* *method*), 301
- disable() (*smc.core.general.DNSRelay* *method*), 303
- disable() (*smc.core.general.Layer2Settings* *method*), 305
- disable() (*smc.core.general.SNMP* *method*), 304
- disable() (*smc.core.interfaces.QoS* *method*), 336
- disable() (*smc.policy.rule.Rule* *method*), 451
- disable() (*smc.routing.bgp.BGP* *method*), 411
- disable() (*smc.routing.ospf.OSPF* *method*), 425
- disable() (*smc.vpn.route.RouteVPN* *method*), 486
- disable\_connection\_sync() (*smc.core.engine.HAForSingleEngine* *method*), 288
- disable\_ha\_backup\_unit() (*smc.core.engine.HAForSingleEngine* *method*), 288
- disabled\_sites (*smc.vpn.policy.GatewayNode* *property*), 496
- DisableUnusedAdminTask (class in *smc.administration.scheduled\_tasks*), 142
- disapprove\_all() (*smc.core.resource.PendingChanges* *method*), 371
- DISCARD (*smc\_monitoring.models.constants.Actions* *attribute*), 61
- DISCARD\_PASSIVE (*smc\_monitoring.models.constants.Actions* *attribute*), 61
- discard\_quic\_if\_cant\_inspect (*smc.core.engine.Engine* *property*), 274
- display\_name (*smc.elements.situations.SituationParameter* *property*), 260
- dns (*smc.core.engine.Engine* *property*), 274
- dns\_answer\_translation (*smc.elements.profiles.DNSRelayProfile* *property*), 263
- dns\_relay (*smc.core.engine.Engine* *property*), 274
- DNSAnswerTranslation (class in *smc.elements.profiles*), 262
- DNSEntry (class in *smc.core.general*), 302
- DNSRelay (class in *smc.core.general*), 303
- DNSRelayProfile (class in *smc.elements.profiles*), 263
- DNSRule (class in *smc.elements.profiles*), 263
- DNSServer (class in *smc.elements.servers*), 223
- domain (*smc.administration.access\_rights.Permission* *property*), 121
- domain (*smc.api.session.Session* *property*), 105
- domain (*smc\_monitoring.monitors.users.User* *property*), 97
- domain\_name\_entry (*smc.elements.network.DomainName* *property*), 177
- domain\_server\_address (*smc.elements.netlink.StaticNetlink* *property*), 192
- domain\_specific\_dns\_server (*smc.elements.profiles.DNSRelayProfile* *property*), 263
- DomainName (class in *smc.elements.network*), 177
- DomainSpecificDNSServer (class in *smc.elements.profiles*), 264
- done() (*smc.administration.tasks.TaskOperationPoller* *method*), 167
- done() (*smc.core.waiters.NodeWaiter* *method*), 525
- dos\_protection (*smc.policy.rule\_elements.Action* *property*), 467
- download() (*smc.administration.updates.PackageMixin* *method*), 170
- download() (*smc.core.resource.Snapshot* *method*), 384
- download() (*smc.elements.network.IPList* *method*), 181
- download\_html() (*smc.core.resource.Snapshot* *method*), 385
- download\_only() (*smc.administration.tasks.Task* *method*), 166
- DownloadTask (class in *smc.administration.tasks*), 165
- DPD (*smc\_monitoring.models.constants.LogField* *attribute*), 66
- DPORT (*smc\_monitoring.models.constants.LogField* *attribute*), 66

[dscp\\_marking\\_and\\_throttling\(\)](#)  
 ([smc.core.interfaces.QoS](#) method), 336  
[DSCP\\_MARK](#) ([smc\\_monitoring.models.constants.LogField](#)  
[attribute](#)), 66  
[dst](#) ([smc.policy.rule\\_elements.Destination](#) property), 466  
[DST](#) ([smc\\_monitoring.models.constants.LogField](#) [attribute](#)), 66  
[DSTADDRS](#) ([smc\\_monitoring.models.constants.LogField](#)  
[attribute](#)), 66  
[DSTIF](#) ([smc\\_monitoring.models.constants.LogField](#) [attribute](#)), 66  
[DSTIPRANGE](#) ([smc\\_monitoring.models.constants.LogField](#)  
[attribute](#)), 66  
[DSTVLAN](#) ([smc\\_monitoring.models.constants.LogField](#) [attribute](#)), 66  
[DSTZONE](#) ([smc\\_monitoring.models.constants.LogField](#) [attribute](#)), 66  
[duplicate\(\)](#) ([smc.base.model.Element](#) method), 110  
[duplicates\(\)](#) ([smc.base.collection.Search](#) method),  
 513  
[duration](#) ([smc\\_monitoring.monitors.blacklist.BlacklistEntry](#)  
[property](#)), 86  
[dyn\\_up](#) ([smc.administration.monitoring\\_status.EngineNode](#)  
[property](#)), 515  
[dyn\\_up](#) ([smc.core.node.ApplianceStatus](#) property), 366  
[dynamic\\_nicid](#) ([smc.core.route.RoutingTree](#) property),  
 374  
[dynamic\\_routing](#) ([smc.core.engine.Engine](#) property),  
 274  
[dynamic\\_src\\_nat](#) ([smc.policy.rule\\_nat.NATRule](#) prop-  
[erty](#)), 460  
[DynamicNetlink](#) (class in [smc.elements.netlink](#)), 186  
[DynamicSourceNAT](#) (class in [smc.policy.rule\\_nat](#)), 476  
**E**  
[EcaClientConfigExportError](#), 527  
[EcaOperatingSystemSituation](#) (class in  
[smc.elements.situations](#)), 257  
[elasticsearch\\_authentication\\_settings](#)  
 ([smc.elements.servers.ManagementLogServerMixin](#)  
[property](#)), 212  
[ElasticsearchCluster](#) (class in  
[smc.elements.servers](#)), 234  
[Element](#) (class in [smc.base.model](#)), 109  
[ElementBase](#) (class in [smc.base.model](#)), 108  
[ElementCollection](#) (class in [smc.base.collection](#)), 503  
[ELEMENTDOMAIN](#) ([smc\\_monitoring.models.constants.LogField](#)  
[attribute](#)), 66  
[ElementFormat](#) (class in  
[smc\\_monitoring.models.formatters](#)), 82  
[ElementNotFound](#), 527  
[ElementValue](#) (class in [smc\\_monitoring.models.values](#)),  
 56  
[email\\_sender\\_address](#)  
 ([smc.elements.servers.SmtServer](#) property),  
 230  
[email\\_sender\\_name](#) ([smc.elements.servers.SmtServer](#)  
[property](#)), 230  
[empty\\_members\(\)](#) ([smc.elements.group.GroupMixin](#)  
[method](#)), 204  
[empty\\_trash\\_bin\(\)](#) ([smc.administration.system.System](#)  
[method](#)), 160  
[enable\(\)](#) ([smc.administration.role.Role](#) method), 123  
[enable\(\)](#) ([smc.core.addon.AntiVirus](#) method), 295  
[enable\(\)](#) ([smc.core.addon.ClientInspection](#) method),  
 298  
[enable\(\)](#) ([smc.core.addon.FileReputation](#) method), 296  
[enable\(\)](#) ([smc.core.addon.Sandbox](#) method), 299  
[enable\(\)](#) ([smc.core.addon.SidewinderProxy](#) method),  
 296  
[enable\(\)](#) ([smc.core.addon.UrlFiltering](#) method), 297  
[enable\(\)](#) ([smc.core.addon.ZTNAConnector](#) method),  
 298  
[enable\(\)](#) ([smc.core.general.DefaultNAT](#) method), 301  
[enable\(\)](#) ([smc.core.general.DNSRelay](#) method), 303  
[enable\(\)](#) ([smc.core.general.Layer2Settings](#) method),  
 306  
[enable\(\)](#) ([smc.core.general.SNMP](#) method), 304  
[enable\(\)](#) ([smc.policy.rule.Rule](#) method), 451  
[enable\(\)](#) ([smc.routing.bgp.BGP](#) method), 411  
[enable\(\)](#) ([smc.routing.ospf.OSPF](#) method), 425  
[enable\(\)](#) ([smc.vpn.route.RouteVPN](#) method), 486  
[enable\\_aggregate\\_mode\(\)](#)  
 ([smc.core.interfaces.PhysicalInterface](#)  
[method](#)), 339  
[enable\\_disable\(\)](#) ([smc.elements.user.UserMixin](#)  
[method](#)), 119  
[enable\\_disable\(\)](#) ([smc.vpn.elements.ExternalEndpoint](#)  
[method](#)), 492  
[enable\\_disable\(\)](#) ([smc.vpn.policy.GatewayTreeNode](#)  
[method](#)), 499  
[enable\\_disable\(\)](#) ([smc.vpn.policy.GatewayTunnel](#)  
[method](#)), 499  
[enable\\_disable\(\)](#) ([smc.vpn.route.EndpointTunnel](#)  
[method](#)), 483  
[enable\\_disable\\_force\\_nat\\_t\(\)](#)  
 ([smc.vpn.elements.ExternalEndpoint](#) method),  
 493  
[enable\\_disable\\_nat\(\)](#) ([smc.vpn.policy.PolicyVPN](#)  
[method](#)), 479  
[enabled](#) ([smc.administration.upcoming\\_event.UpcomingEventsPolicyEntry](#)  
[property](#)), 169  
[enabled](#) ([smc.elements.user.AdminUser](#) property), 118  
[enabled](#) ([smc.vpn.elements.ExternalEndpoint](#) property),  
 493  
[enabled](#) ([smc.vpn.policy.GatewayTunnel](#) property), 499  
[enabled](#) ([smc.vpn.route.EndpointTunnel](#) property), 483

enabled\_sites (*smc.vpn.policy.GatewayNode* property), 496  
 end\_port (*smc.policy.rule\_nat.DynamicSourceNAT* property), 476  
 end\_time (*smc.administration.tasks.Task* property), 166  
 end\_time (*smc\_monitoring.models.calendar.TimeFormat* property), 83  
 endpoint (*smc.vpn.route.TunnelEndpoint* property), 488  
 ENDPOINT (*smc\_monitoring.models.constants.LogField* attribute), 66  
 endpoint\_executable\_logging (*smc.policy.rule\_elements.LogOptions* property), 470  
 endpoint\_integration (*smc.core.engine.Engine* property), 274  
 endpoint\_tunnels (*smc.vpn.policy.GatewayTunnel* property), 500  
 endpointA (*smc.administration.monitoring\_status.VPNEndpointStatus* property), 520  
 endpointB (*smc.administration.monitoring\_status.VPNEndpointStatus* property), 520  
 EndpointTunnel (class in *smc.vpn.route*), 483  
 Engine (class in *smc.core.engine*), 266  
 engine (*smc\_monitoring.monitors.alerts.Alert* property), 103  
 engine (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86  
 engine (*smc\_monitoring.monitors.connections.Connection* property), 88  
 engine (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
 engine (*smc\_monitoring.monitors.sslvpn.SSLVPNUser* property), 96  
 engine (*smc\_monitoring.monitors.users.User* property), 97  
 engine (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc* property), 100  
 engine\_node\_status (*smc.administration.monitoring\_status.EngineNodeStatus* property), 515  
 engine\_upgrade() (*smc.administration.system.System* method), 160  
 engine\_upgrade\_import() (*smc.administration.system.System* method), 160  
 EngineCommandFailed, 527  
 EngineNodeStatus (class in *smc.administration.monitoring\_status*), 515  
 EngineUpgrade (class in *smc.administration.updates*), 170  
 ENTERPRISEOID (*smc\_monitoring.models.constants.LogField* attribute), 66  
 entries (*smc.administration.upcoming\_event.UpcomingEventIgnoreSettings* property), 169  
 entry\_point() (*smc.base.collection.Search* method), 513  
 entry\_points (*smc.api.session.Session* property), 105  
 epo\_login (*smc.elements.servers.EpoServer* property), 229  
 epo\_port (*smc.elements.servers.EpoServer* property), 229  
 EpoServer (class in *smc.elements.servers*), 229  
 es\_enable\_cluster\_sniffer (*smc.elements.servers.ElasticsearchCluster* property), 234  
 es\_retention\_period (*smc.elements.servers.ElasticsearchCluster* property), 234  
 es\_shard\_number (*smc.elements.servers.ElasticsearchCluster* property), 235  
 es\_tls\_settings (*smc.elements.servers.ElasticsearchCluster* property), 235  
 es\_index\_settings (*smc.elements.servers.ManagementLogServerMixin* property), 212  
 event\_timestamp (*smc\_monitoring.SMCRequest* attribute), 522  
 EthernetRule (class in *smc.policy.rule*), 457  
 EthernetService (class in *smc.elements.service*), 193  
 EthernetServiceGroup (class in *smc.elements.group*), 209  
 EVENT (*smc\_monitoring.models.constants.LogField* attribute), 66  
 event\_date (*smc.administration.upcoming\_event.UpcomingEvent* property), 168  
 EVENTADDRESS (*smc\_monitoring.models.constants.LogField* attribute), 66  
 EVENTINFO (*smc\_monitoring.models.constants.LogField* attribute), 66  
 EVENTLOGID (*smc\_monitoring.models.constants.LogField* attribute), 67  
 EVENTTIME (*smc\_monitoring.models.constants.LogField* attribute), 67  
 EVENTTYPE (*smc\_monitoring.models.constants.LogField* attribute), 67  
 EVENTUSER (*smc\_monitoring.models.constants.LogField* attribute), 67  
 exact\_ipv4\_match() (*smc\_monitoring.models.filters.TranslatedFilter* method), 55  
 execute() (*smc\_monitoring.models.query.Query* method), 49  
 exists() (*smc.base.collection.ElementCollection* method), 505  
 expiration (*smc\_monitoring.monitors.users.User* property), 98  
 expiration (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc* property), 100  
 EXPIRATIONTIME (*smc\_monitoring.models.constants.LogField* attribute), 67  
 export() (*smc.base.model.Element* method), 110  
 export() (*smc.policy.policy.InspectionPolicy* method), 110

- 442
- `export_certificate()` (`smc.administration.certificates.tls_common.ImportExportCertificate` method), 126
- `export_elements()` (`smc.administration.system.System` method), 161
- `export_intermediate_certificate()` (`smc.administration.certificates.tls_common.ImportExportIntermediateCertificate` method), 127
- `export_ldif_elements()` (`smc.administration.system.System` method), 161
- `export_pdf()` (`smc.administration.reports.Report` method), 154
- `export_text()` (`smc.administration.reports.Report` method), 154
- `export_user_authentication_certificate()` (`smc.core.engine.Engine` method), 275
- `export_user_authentication_certificate_request()` (`smc.core.engine.Engine` method), 275
- `ExportLogTask` (class in `smc.administration.scheduled_tasks`), 143
- `Expression` (class in `smc.elements.network`), 178
- `ExtCommunityListEntry` (class in `smc.routing.bgp_access_list`), 423
- `ExtendedCommunityAccessList` (class in `smc.routing.bgp_access_list`), 423
- `external_distance` (`smc.routing.bgp.BGPProfile` property), 419
- `external_endpoint` (`smc.vpn.elements.ExternalGateway` property), 490
- `external_pki_certificate_settings` (`smc.elements.servers.ManagementLogServerMixin` property), 212
- `external_pki_certificate_settings` (`smc.elements.servers.WebPortalServer` property), 229
- `ExternalBGPPeer` (class in `smc.routing.bgp`), 414
- `ExternalEndpoint` (class in `smc.vpn.elements`), 491
- `ExternalGateway` (class in `smc.vpn.elements`), 490
- ## F
- `FACILITY` (`smc_monitoring.models.constants.LogField` attribute), 67
- `fetch_as_element()` (`smc_monitoring.models.query.Query` method), 50
- `fetch_as_element()` (`smc_monitoring.monitors.alerts.ActiveAlertQuery` method), 102
- `fetch_as_element()` (`smc_monitoring.monitors.blacklist.BlacklistQuery` method), 87
- `fetch_as_element()` (`smc_monitoring.monitors.connections.ConnectionQuery` method), 90
- `fetch_as_element()` (`smc_monitoring.monitors.routes.RoutingQuery` method), 93
- `fetch_as_element()` (`smc_monitoring.monitors.sslvpn.SSLVPNQuery` method), 96
- `fetch_as_element()` (`smc_monitoring.monitors.users.UserQuery` method), 98
- `fetch_as_element()` (`smc_monitoring.monitors.vpn.vpn.VPNQuery` method), 99
- `fetch_batch()` (`smc_monitoring.models.query.Query` method), 50
- `fetch_batch()` (`smc_monitoring.monitors.logs.LogQuery` method), 92
- `fetch_license()` (`smc.core.node.Node` method), 359
- `fetch_live()` (`smc_monitoring.models.query.Query` method), 50
- `fetch_live()` (`smc_monitoring.monitors.logs.LogQuery` method), 92
- `fetch_raw()` (`smc_monitoring.models.query.Query` method), 50
- `fetch_raw()` (`smc_monitoring.monitors.logs.LogQuery` method), 92
- `fetch_size` (`smc_monitoring.monitors.logs.LogQuery` property), 92
- `FetchCertificateRevocationTask` (class in `smc.administration.scheduled_tasks`), 144
- `FetchElementFailed`, 527
- `field_format()` (`smc_monitoring.models.formats.FormatFieldMixin` method), 60
- `field_ids()` (`smc_monitoring.models.formats.FormatFieldMixin` method), 60
- `field_names()` (`smc_monitoring.models.formats.FormatFieldMixin` method), 60
- `FieldValue` (class in `smc_monitoring.models.values`), 57
- `file_filtering` (`smc.policy.rule_elements.Action` property), 467
- `file_filtering_rules` (`smc.policy.file_filtering.FileFilteringPolicy` property), 437
- `file_reputation` (`smc.core.engine.Engine` property), 275
- `FileFilteringPolicy` (class in `smc.policy.file_filtering`), 437
- `FileFilteringRule` (class in `smc.policy.file_filtering`), 437
- `filename` (`smc.api.common.SMCRequest` attribute), 522
- `FileReputation` (class in `smc.core.addon`), 296
- `filesystem` (`smc.core.node.HardwareStatus` property), 368
- `FILETYPECOMPAT` (`smc_monitoring.models.constants.LogField` attribute), 67
- `filter` (`smc.elements.servers.NetflowCollector` property), 215
- `filter()` (`smc.base.collection.CollectionManager` method), 507
- `filter()` (`smc.base.collection.ElementCollection` method), 505

- FilterExpression (class in *smc.elements.other*), 240, 248
- find\_system\_element() (*smc.administration.system.System* method), 161
- finish (*smc.routing.route\_map.RouteMapRule* property), 404
- FirewallCluster (class in *smc.core.engines*), 394
- FirewallIPv6SubPolicy (class in *smc.policy.layer3*), 439
- FirewallPolicy (class in *smc.policy.layer3*), 439
- FirewallRule (class in *smc.policy.layer3*), 440
- FirewallSubPolicy (class in *smc.policy.layer3*), 441, 449
- FirewallTemplatePolicy (class in *smc.policy.layer3*), 441
- first() (*smc.base.collection.CollectionManager* method), 508
- first() (*smc.base.collection.ElementCollection* method), 505
- first\_fetch(*smc\_monitoring.monitors.blacklist.Blacklist* property), 86
- first\_fetch(*smc\_monitoring.monitors.connections.Connection* property), 89
- first\_fetch(*smc\_monitoring.monitors.routes.RoutingView* property), 94
- first\_fetch(*smc\_monitoring.monitors.sslvpn.SSLVPNUser* property), 96
- first\_fetch (*smc\_monitoring.monitors.users.User* property), 98
- first\_fetch(*smc\_monitoring.monitors.vpns.VPNSecurity* property), 100
- fixed\_domain\_answer (*smc.elements.profiles.DNSRelayProfile* property), 263
- FixedDomainAnswer (class in *smc.elements.profiles*), 264
- FLAG (*smc\_monitoring.models.constants.LogField* attribute), 67
- force\_nat\_t (*smc.vpn.elements.ExternalEndpoint* property), 493
- force\_unlock() (*smc.policy.policy.Policy* method), 433
- forced\_next\_hop\_element (*smc.policy.rule\_elements.Action* property), 467
- forced\_next\_hop\_ip (*smc.policy.rule\_elements.Action* property), 467
- FormatFieldMixin (class in *smc\_monitoring.models.formats*), 59
- forwarding\_tls\_settings (*smc.elements.servers.ManagementLogServerMixin* property), 212
- fp\_proxy\_key (*smc.elements.servers.ProxyServer* property), 226
- fp\_proxy\_key\_id (*smc.elements.servers.ProxyServer* property), 226
- fp\_proxy\_user\_id (*smc.elements.servers.ProxyServer* property), 226
- FPCACHED (*smc\_monitoring.models.constants.LogField* attribute), 67
- full\_qos() (*smc.core.interfaces.QoS* method), 336
- full\_resolve() (*smc.elements.network.Alias* method), 175
- FW100INTERFACE (*smc\_monitoring.models.constants.LogField* attribute), 67
- FW100TRAFFICCOUNTERS (*smc\_monitoring.models.constants.LogField* attribute), 67
- fw\_ipv4\_access\_rules (*smc.policy.layer3.FirewallRule* property), 440
- fw\_ipv4\_access\_rules (*smc.policy.layer3.FirewallSubPolicy* property), 441, 450
- fw\_ipv4\_nat\_rules (*smc.policy.layer3.FirewallRule* property), 440
- fw\_ipv6\_access\_rules (*smc.policy.layer3.FirewallIPv6SubPolicy* property), 439
- fw\_ipv6\_access\_rules (*smc.policy.layer3.FirewallRule* property), 440
- fw\_ipv6\_nat\_rules (*smc.policy.layer3.FirewallRule* property), 441
- FWACCEPTEDBYTES (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWACCEPTEDPACKETS (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWACCOUNTEDBYTES (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWACCOUNTEDPACKETS (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWADSLRXBYTES (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWADSLTXBYTES (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWDECRYPTEDBYTES (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWDECRYPTEDPACKETS (*smc\_monitoring.models.constants.LogField* attribute), 67
- FWDROPPEDBYTES (*smc\_monitoring.models.constants.LogField* attribute), 68
- FWDROPPEDPACKETS (*smc\_monitoring.models.constants.LogField* attribute), 68
- FWENCRYPTEDBYTES (*smc\_monitoring.models.constants.LogField* attribute), 68
- FWENCRYPTEDPACKETS (*smc\_monitoring.models.constants.LogField* attribute), 68

- FWFORWARDEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWFORWARDEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWINTERFACEKEY (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWNATTEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWNATTEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWRECEIVEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWRECEIVEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWSENTBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWSENTPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFIC (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFICACCOUNTEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFICACCOUNTEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFICALLOWEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFICALLOWEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFICDISCARDEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 68
- FWTRAFFICDISCARDEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 69
- FWTRAFFICENCRYPTEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 69
- FWTRAFFICENCRYPTEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 69
- FWTRAFFICLOGGEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 69
- FWTRAFFICLOGGEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 69
- FWTRAFFICNATTEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 69
- FWTRAFFICNATTEDPACKETS (*smc\_monitoring.models.constants.LogField attribute*), 69
- gateway\_certificate (*smc.core.engine.VPN property*), 291
- gateway\_certificate\_request (*smc.core.engine.VPN property*), 291
- gateway\_profile (*smc.core.engine.VPN property*), 292
- gateway\_settings (*smc.core.engine.VPN property*), 292
- gatewayA (*smc.administration.monitoring\_status.VPNGatewayTunnelStatus property*), 520
- gatewayB (*smc.administration.monitoring\_status.VPNGatewayTunnelStatus property*), 520
- GatewayNode (*class in smc.vpn.policy*), 496
- GatewayProfile (*class in smc.vpn.elements*), 496
- GatewaySettings (*class in smc.vpn.elements*), 495
- GatewayTreeNode (*class in smc.vpn.policy*), 499
- GatewayTunnel (*class in smc.vpn.policy*), 499
- generate() (*smc.administration.reports.ReportDesign method*), 155
- generate\_and\_sign\_user\_authentication\_certificate() (*smc.core.engine.Engine method*), 275
- generate\_certificate() (*smc.core.engine.VPN method*), 292
- generate\_password() (*smc.elements.user.UserMixin method*), 119
- generate\_snapshot() (*smc.core.engine.Engine method*), 275
- generate\_user\_authentication\_certificate\_request() (*smc.core.engine.Engine method*), 275
- GENERICTRAPTYPE (*smc\_monitoring.models.constants.LogField attribute*), 69
- Geolocation (*class in smc.elements.other*), 240
- geolocation (*smc.core.engine.Engine property*), 275
- get() (*smc.base.collection.SubElementCollection method*), 509
- get() (*smc.base.model.Element class method*), 111
- get() (*smc.base.structs.BaseIterable method*), 514
- get() (*smc.core.advanced\_settings.LogModeration method*), 387
- get() (*smc.core.collection.InterfaceCollection method*), 313
- get() (*smc.core.collection.LoopbackCollection method*), 314
- get() (*smc.core.contact\_address.ContactAddressCollection method*), 356
- get() (*smc.core.node.InterfaceStatus method*), 369
- get() (*smc.core.route.RoutingTree method*), 374
- get() (*smc.elements.protocols.ProtocolAgentValues method*), 201
- get() (*smc.elements.servers.MultiContactAddress method*), 214

[get\\_action\(\)](#) (*smc.policy.file\_filtering.FileFilteringRule* method), 438  
[get\\_active\\_alerts\(\)](#) (*smc.administration.system.AdminDomain* method), 138, 158  
[get\\_all\\_contains\(\)](#) (*smc.base.collection.SubElementCollection* method), 510  
[get\\_boolean\(\)](#) (*smc.core.interfaces.Interface* method), 328  
[get\\_contains\(\)](#) (*smc.base.collection.SubElementCollection* method), 510  
[get\\_exact\(\)](#) (*smc.base.collection.SubElementCollection* method), 510  
[get\\_or\\_create\(\)](#) (*smc.base.model.Element* class method), 111  
[get\\_result\(\)](#) (*smc.administration.tasks.Task* method), 166  
[get\\_session\\_monitoring\(\)](#) (*smc.core.engine.Engine* method), 276  
[get\\_status\(\)](#) (*smc.administration.monitoring\_status.MonitoringStatus* static method), 517  
[get\\_task\\_poller\(\)](#) (*smc.administration.tasks.Task* method), 166  
[go\\_offline\(\)](#) (*smc.core.node.Node* method), 359  
[go\\_online\(\)](#) (*smc.core.node.Node* method), 360  
[go\\_standby\(\)](#) (*smc.core.node.Node* method), 360  
[goto](#) (*smc.routing.route\_map.RouteMapRule* property), 404  
[goto\\_rule\\_section\(\)](#) (*smc.routing.route\_map.RouteMapRule* method), 404  
[granted\\_elements](#) (*smc.administration.access\_rights.Permission* property), 121  
[granted\\_permissions](#) (*smc.core.engine.Engine* property), 276  
[Group](#) (class in *smc.elements.group*), 207  
[GroupMixin](#) (class in *smc.elements.group*), 204  

## H

[ha\\_backup\\_unit\\_mode](#) (*smc.core.engine.HAForSingleEngine* property), 288  
[ha\\_settings](#) (*smc.core.engines.Layer3Firewall* property), 392  
[HaCommandException](#), 527  
[HAForSingleEngine](#) (class in *smc.core.engine*), 288  
[hardware\\_status](#) (*smc.core.node.Node* property), 360  
[HardwareStatus](#) (class in *smc.core.node*), 367  
[has\\_interfaces](#) (*smc.core.interfaces.Interface* property), 328  
[has\\_nat](#) (*smc.policy.rule\_nat.NATElement* property), 475  
[has\\_vlan](#) (*smc.core.interfaces.Interface* property), 328  
[HASHALG](#) (*smc\_monitoring.models.constants.LogField* attribute), 69  
[headers](#) (*smc.api.common.SMCRequest* attribute), 522  
[health](#) (*smc.core.node.Node* property), 360  
[health\\_in\\_percent](#) (*smc.administration.monitoring\_status.SDWANBranch* property), 518  
[health\\_in\\_percent](#) (*smc.administration.monitoring\_status.SDWANBranch* property), 519  
[health\\_in\\_percent](#) (*smc.administration.monitoring\_status.VPNStatus* property), 521  
[HIGH](#) (*smc\_monitoring.models.constants.Alerts* attribute), 61  
[History](#) (class in *smc.core.resource*), 114  
[history](#) (*smc.administration.monitoring\_status.MonitoringStatus* property), 517  
[history](#) (*smc.base.model.Element* property), 112  
[history](#) (*smc.policy.rule.Rule* property), 451  
[HITS](#) (*smc\_monitoring.models.constants.LogField* attribute), 69  
[hold\\_time\\_multiplier](#) (*smc.core.lldp.LLDPProfile* property), 265  
[Host](#) (class in *smc.elements.network*), 179  
[host](#) (*smc.elements.servers.NetflowCollector* property), 215  
[hostname\\_mapping](#) (*smc.elements.profiles.DNSRelayProfile* property), 263  
[HostnameMapping](#) (class in *smc.elements.profiles*), 264  
[href](#) (*smc.api.common.SMCRequest* attribute), 522  
[href](#) (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86  
[http\\_proxy](#) (*smc.core.addon.FileReputation* property), 296  
[http\\_proxy](#) (*smc.core.addon.Sandbox* property), 300  
[http\\_proxy](#) (*smc.core.addon.UrlFiltering* property), 297  
[http\\_proxy\(\)](#) (*smc.core.addon.AntiVirus* method), 295  
[http\\_proxy\\_port](#) (*smc.elements.servers.HttpProxy* property), 224  
[http\\_proxy\\_username](#) (*smc.elements.servers.HttpProxy* property), 224  
[HttpProxy](#) (class in *smc.elements.servers*), 224  
[HTTPREQUESTHOST](#) (*smc\_monitoring.models.constants.LogField* attribute), 69  
[HTTPSInspectionExceptions](#) (class in *smc.elements.other*), 240, 250  
[hw\\_info](#) (*smc.administration.monitoring\_status.EngineNodeStatus* property), 516  

## I

[icap\\_include\\_xhdrs](#) (*smc.elements.servers.IcapServer* property), 231  
[icap\\_path](#) (*smc.elements.servers.IcapServer* property), 231

- `icap_port` (`smc.elements.servers.IcapServer` property), 232
- `icap_secure` (`smc.elements.servers.IcapServer` property), 232
- `icap_xhdr_clientip` (`smc.elements.servers.IcapServer` property), 232
- `icap_xhdr_serverip` (`smc.elements.servers.IcapServer` property), 232
- `icap_xhdr_username` (`smc.elements.servers.IcapServer` property), 232
- `IcapServer` (class in `smc.elements.servers`), 231
- `ICMP_CODE` (`smc_monitoring.models.constants.LogField` attribute), 69
- `ICMPID` (`smc_monitoring.models.constants.LogField` attribute), 69
- `ICMPIPv6Service` (class in `smc.elements.service`), 195
- `ICMPService` (class in `smc.elements.service`), 195
- `ICMPServiceGroup` (class in `smc.elements.group`), 205
- `ICMP_TYPE` (`smc_monitoring.models.constants.LogField` attribute), 69
- `IdleTimeout` (class in `smc.core.engine`), 289
- `ignore_other` (`smc.core.engine.LBFilter` property), 290
- `IKEDHGROUP` (`smc_monitoring.models.constants.LogField` attribute), 69
- `IKELOCALID` (`smc_monitoring.models.constants.LogField` attribute), 69
- `IKEREMOTEID` (`smc_monitoring.models.constants.LogField` attribute), 69
- `IKEV1MODE` (`smc_monitoring.models.constants.LogField` attribute), 69
- `impacted_element` (`smc.administration.upcoming_event.UpcomingEvent` property), 168
- `impacted_resources` (`smc.administration.upcoming_event.UpcomingEvent` property), 168
- `import_certificate()` (`smc.administration.certificates.tls_common.ImportExportCertificate` method), 126
- `import_elements()` (`smc.administration.system.System` method), 161
- `import_from_chain()` (`smc.administration.certificates.tls.TLSServerCredential` class method), 131
- `import_intermediate_certificate()` (`smc.administration.certificates.tls_common.ImportExportCertificate` method), 127
- `import_ldif_elements()` (`smc.administration.system.System` method), 161
- `import_new_certificate_authority_certificate()` (`smc.administration.system.System` method), 161
- `import_private_key()` (`smc.administration.certificates.tls_common.ImportPrivateKey` method), 127
- `import_signed()` (`smc.administration.certificates.tls.ClientProtectionCA` class method), 136
- `import_signed()` (`smc.administration.certificates.tls.TLSServerCredential` class method), 131
- `ImportExportCertificate` (class in `smc.administration.certificates.tls_common`), 126
- `ImportExportIntermediate` (class in `smc.administration.certificates.tls_common`), 127
- `ImportPrivateKey` (class in `smc.administration.certificates.tls_common`), 127
- `inactive` (`smc.elements.servers.LogServer` property), 218
- `inbound_traffic_in_bits_per_sec` (`smc.administration.monitoring_status.SDWANNetlinkElementStatus` property), 519
- `INCIDENTCASE` (`smc_monitoring.models.constants.LogField` attribute), 69
- `InFilter` (class in `smc_monitoring.models.filters`), 53
- `info` (`smc.administration.upcoming_event.UpcomingEvent` property), 168
- `INFO` (`smc_monitoring.models.constants.Alerts` attribute), 62
- `INFOMSG` (`smc_monitoring.models.constants.LogField` attribute), 69
- `initial_contact()` (`smc.core.node.Node` method), 360
- `InlineInterface` (class in `smc.core.sub_interfaces`), 350
- `InlineIPSInterface` (class in `smc.core.sub_interfaces`), 350
- `InlineVLANInterface` (class in `smc.core.sub_interfaces`), 351
- `inspected_services` (`smc.elements.servers.ProxyServer` property), 226
- `inspection_policy()` (`smc.policy.interface.InterfacePolicy` method), 435
- `inspection_policy()` (`smc.policy.interface.InterfaceTemplatePolicy` method), 436
- `InspectionPolicy` (class in `smc.policy.policy`), 442
- `InspectionSituation` (class in `smc.elements.situations`), 257
- `InspectionSituationContext` (class in `smc.elements.situations`), 258
- `installed_policy` (`smc.core.engine.Engine` property), 277
- `installed_policy` (`smc.core.node.ApplianceStatus` property), 366
- `installed_policy_ref` (`smc.administration.monitoring_status.EngineNodeStatus` property), 516

- Interface (class in *smc.core.interfaces*), 326
- interface (*smc.core.engine.Engine* property), 277
- interface (*smc.core.general.SNMP* property), 304
- INTERFACE (*smc\_monitoring.models.constants.LogField* attribute), 70
- interface\_id (*smc.core.contact\_address.ContactAddressNode* property), 356
- interface\_id (*smc.core.engine.InternalEndpoint* property), 308
- interface\_id (*smc.core.interfaces.Interface* property), 328
- interface\_ip (*smc.core.contact\_address.ContactAddressNode* property), 357
- interface\_options (*smc.core.engine.Engine* property), 277
- interface\_status (*smc.core.node.Node* property), 361
- InterfaceCollection (class in *smc.core.collection*), 311
- InterfaceContactAddress (class in *smc.core.contact\_address*), 357
- InterfaceNotFound, 527
- InterfaceOptions (class in *smc.core.interfaces*), 332
- InterfacePolicy (class in *smc.policy.interface*), 435
- InterfaceRule (class in *smc.policy.interface*), 435
- interfaces (*smc.core.interfaces.Interface* property), 329
- InterfaceStatus (class in *smc.core.node*), 369
- InterfaceSubPolicy (class in *smc.policy.interface*), 436
- InterfaceTemplatePolicy (class in *smc.policy.interface*), 436
- internal\_distance (*smc.routing.bgp.BGPPProfile* property), 419
- internal\_endpoint (*smc.core.engine.InternalGateway* property), 309
- internal\_endpoint (*smc.core.engine.VPN* property), 292
- internal\_gateway (*smc.core.engine.Engine* property), 277
- internal\_gateway (*smc.core.engine.VPNMapping* property), 293
- InternalEndpoint (class in *smc.core.engine*), 308
- InternalGateway (class in *smc.core.engine*), 309
- InvalidFieldFormat, 82
- InvalidLogSeverity, 527
- InvalidRuleValue, 527
- InvalidSearchFilter, 527
- ip (*smc.core.route.RoutingTree* property), 374
- ip\_address (*smc.elements.servers.ProxyServer* property), 226
- ip\_descriptor (*smc.core.engine.LBFilter* property), 290
- ip\_range (*smc.elements.netlink.MultilinkMember* property), 191
- IPAccessList (class in *smc.routing.access\_list*), 406
- ipaddress (*smc\_monitoring.monitors.users.User* property), 98
- IPCOMPRESSION (*smc\_monitoring.models.constants.LogField* attribute), 70
- IPList (class in *smc.elements.network*), 180
- iplist (*smc.elements.network.IPList* property), 181
- IPPrefixList (class in *smc.routing.prefix\_list*), 408
- IPS (class in *smc.core.engines*), 387
- ips\_ethernet\_rules (*smc.policy.ips.IPSPolicyRule* property), 444
- ips\_ipv4\_access\_rules (*smc.policy.ips.IPSPolicyRule* property), 444
- ips\_ipv4\_access\_rules (*smc.policy.ips.IPSSubPolicy* property), 445
- IPSAPPID (*smc\_monitoring.models.constants.LogField* attribute), 70
- IPSECSSPI (*smc\_monitoring.models.constants.LogField* attribute), 70
- IPService (class in *smc.elements.service*), 196
- IPServiceGroup (class in *smc.elements.group*), 206
- IPSPolicy (class in *smc.policy.ips*), 444
- IPSPolicyRule (class in *smc.policy.ips*), 444
- IPSSubPolicy (class in *smc.policy.ips*), 445
- IPSTemplatePolicy (class in *smc.policy.ips*), 445
- IPv4Layer2Rule (class in *smc.policy.rule*), 456
- IPv4NATRule (class in *smc.policy.rule\_nat*), 461
- IPv4Rule (class in *smc.policy.rule*), 453
- IPv6AccessList (class in *smc.routing.access\_list*), 407
- IPv6NATRule (class in *smc.policy.rule\_nat*), 464
- IPv6PrefixList (class in *smc.routing.prefix\_list*), 408
- IPv6Rule (class in *smc.policy.rule*), 459
- IPValue (class in *smc\_monitoring.models.values*), 57
- is\_active (*smc.api.session.Session* property), 105
- is\_any (*smc.policy.rule\_elements.RuleElement* property), 465
- is\_auth\_request (*smc.core.interfaces.PhysicalInterface* property), 340
- is\_backup\_heartbeat (*smc.core.interfaces.PhysicalInterface* property), 340
- is\_backup\_mgt (*smc.core.interfaces.PhysicalInterface* property), 340
- is\_central\_gateway (*smc.core.engine.VPNMapping* property), 293
- is\_disabled (*smc.policy.rule.Rule* property), 451
- is\_disabled (*smc.routing.route\_map.RouteMapRule* property), 405
- is\_locked() (*smc.base.model.Element* method), 112
- is\_mobile\_gateway (*smc.core.engine.VPNMapping* property), 293
- is\_none (*smc.policy.rule\_elements.RuleElement* property), 465

[is\\_outgoing](#) ([smc.core.interfaces.PhysicalInterface](#) property), 340  
[is\\_primary\\_heartbeat](#) ([smc.core.interfaces.PhysicalInterface](#) property), 340  
[is\\_primary\\_mgt](#) ([smc.core.interfaces.PhysicalInterface](#) property), 340  
[is\\_rule\\_section](#) ([smc.policy.rule.Rule](#) property), 451  
[is\\_satellite\\_gateway](#) ([smc.core.engine.VPNMapping](#) property), 293  
[is\\_ssl](#) ([smc.api.session.Session](#) property), 105  
[isp\\_link\\_ref](#) ([smc.core.engine.LinkUsageExceptionRules](#) property), 290  
[iterator\(\)](#) ([smc.base.collection.CollectionManager](#) method), 508  
**J**  
[jitter\\_in\\_ms](#) ([smc.administration.monitoring\\_status.SDWANBranchToB](#) property), 519  
[jitter\\_in\\_ms](#) ([smc.administration.monitoring\\_status.VPNStatus](#) property), 521  
**K**  
[kafka\\_topic](#) ([smc.elements.servers.NetflowCollector](#) property), 215  
[known\\_host\\_lists](#) ([smc.core.engine.Engine](#) property), 278  
**L**  
[l2fw\\_settings](#) ([smc.core.engine.Engine](#) property), 278  
[last\(\)](#) ([smc.base.collection.ElementCollection](#) method), 506  
[last\\_activated\\_package](#) ([smc.administration.system.System](#) property), 161  
[last\\_day\(\)](#) ([smc\\_monitoring.models.calendar.TimeFormat](#) method), 84  
[last\\_fifteen\\_minutes\(\)](#) ([smc\\_monitoring.models.calendar.TimeFormat](#) method), 84  
[last\\_five\\_minutes\(\)](#) ([smc\\_monitoring.models.calendar.TimeFormat](#) method), 84  
[last\\_hour\(\)](#) ([smc\\_monitoring.models.calendar.TimeFormat](#) method), 84  
[last\\_message](#) ([smc.administration.tasks.Task](#) property), 166  
[last\\_message\(\)](#) ([smc.administration.tasks.TaskOperationPoller](#) method), 167  
[last\\_modified](#) ([smc.core.resource.History](#) property), 114  
[last\\_thirty\\_minutes\(\)](#) ([smc\\_monitoring.models.calendar.TimeFormat](#) method), 84  
[last\\_upload\\_time](#) ([smc.administration.monitoring\\_status.EngineNodeSta](#) property), 516  
[last\\_week\(\)](#) ([smc\\_monitoring.models.calendar.TimeFormat](#) method), 84  
[latency\\_in\\_ms](#) ([smc.administration.monitoring\\_status.SDWANBranchToB](#) property), 519  
[latency\\_in\\_ms](#) ([smc.administration.monitoring\\_status.VPNStatus](#) property), 521  
[layer2\\_ethernet\\_rules](#) ([smc.policy.interface.InterfaceRule](#) property), 435  
[layer2\\_ethernet\\_rules](#) ([smc.policy.layer2.Layer2Rule](#) property), 447  
[layer2\\_ipv4\\_access\\_rules](#) ([smc.policy.interface.InterfaceRule](#) property), 435  
[layer2\\_ipv4\\_access\\_rules](#) ([smc.policy.interface.InterfaceSubPolicy](#) property), 436  
[layer2\\_ipv4\\_access\\_rules](#) ([smc.policy.layer2.Layer2Rule](#) property), 448  
[layer2\\_ipv6\\_access\\_rules](#) ([smc.policy.interface.InterfaceRule](#) property), 435  
[layer2\\_ipv6\\_access\\_rules](#) ([smc.policy.layer2.Layer2Rule](#) property), 448  
[Layer2Firewall](#) (class in [smc.core.engines](#)), 392  
[Layer2Policy](#) (class in [smc.policy.layer2](#)), 447  
[Layer2Rule](#) (class in [smc.policy.layer2](#)), 447  
[Layer2Settings](#) (class in [smc.core.general](#)), 305  
[Layer2TemplatePolicy](#) (class in [smc.policy.layer2](#)), 448  
[Layer3Firewall](#) (class in [smc.core.engines](#)), 388  
[Layer3PhysicalInterface](#) (class in [smc.core.interfaces](#)), 343, 344  
[Layer3VirtualEngine](#) (class in [smc.core.engines](#)), 393  
[LBFilter](#) (class in [smc.core.engine](#)), 289  
[lbfilter\\_useports](#) ([smc.core.engine.Engine](#) property), 278  
[lbfilters](#) ([smc.core.engine.Engine](#) property), 278  
[ldap\\_replication\(\)](#) ([smc.core.engine.Engine](#) method), 278  
[level](#) ([smc.core.route.RoutingTree](#) property), 374  
[License](#) (class in [smc.administration.license](#)), 139  
[license\\_check\\_for\\_new\(\)](#) ([smc.administration.system.System](#) method), 162  
[license\\_details\(\)](#) ([smc.administration.system.System](#) method), 162

*method*), 162

`license_fetch()` (*smc.administration.system.System method*), 162

`license_install()` (*smc.administration.system.System method*), 162

`license_update_enabled` (*smc.elements.servers.ManagementServer property*), 221

`LicenseError`, 527

`Licenses` (*class in smc.administration.license*), 139

`licenses` (*smc.administration.system.System property*), 162

`limit()` (*smc.base.collection.CollectionManager method*), 508

`limit()` (*smc.base.collection.ElementCollection method*), 506

`link_selection` (*smc.policy.qos.QoSClass property*), 449

`link_usage_exception` (*smc.vpn.elements.LinkUsageProfile property*), 172

`link_usage_exception_rules` (*smc.core.engine.Engine property*), 278

`link_usage_profile` (*smc.core.engine.Engine property*), 278

`LinkType` (*class in smc.elements.netlink*), 187

`LinkUsageExceptionRules` (*class in smc.core.engine*), 290

`LinkUsageProfile` (*class in smc.vpn.elements*), 172

`lldp_mode` (*smc.core.interfaces.PhysicalInterface property*), 341

`lldp_profile` (*smc.core.engine.Engine property*), 278

`LLDPProfile` (*class in smc.core.lldp*), 265

`lls_guaranteed_free_percent` (*smc.core.engine.LocalLogStorageSettings property*), 290

`lls_guaranteed_free_size_in_mb` (*smc.core.engine.LocalLogStorageSettings property*), 290

`lls_max_time` (*smc.core.engine.LocalLogStorageSettings property*), 290

`LoadElementFailed`, 527

`LoadEngineFailed`, 528

`LoadPolicyFailed`, 528

`local_alternative_policies` (*smc.administration.monitoring\_status.EngineNodeStatus property*), 516

`local_alternative_policy_activation_time` (*smc.administration.monitoring\_status.EngineNodeStatus property*), 516

`local_distance` (*smc.routing.bgp.BGPProfile property*), 419

`local_endpoint` (*smc.vpn.route.RouteVPN property*), 486

`local_endpoint` (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc property*), 100

`local_gateway` (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc property*), 100

`local_log_storage` (*smc.core.engine.Engine property*), 279

`local_log_storage_activated` (*smc.core.engine.LocalLogStorageSettings property*), 291

`local_networks` (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc property*), 100

`LocalLogStorageSettings` (*class in smc.core.engine*), 290

`Location` (*class in smc.elements.other*), 240, 248

`location` (*smc.core.engine.Engine property*), 279

`location` (*smc.core.general.SNMP property*), 304

`lock()` (*smc.base.model.Element method*), 112

`lock_offline()` (*smc.core.node.Node method*), 361

`lock_online()` (*smc.core.node.Node method*), 362

`log_accounting_info_mode` (*smc.policy.rule\_elements.LogOptions property*), 470

`log_alert` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_closing_mode` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_compression` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_compression_max_burst_size` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_compression_max_log_rate` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_disk_space_handling_mode` (*smc.elements.servers.ManagementLogServerMixin property*), 212

`log_level` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_level()` (*smc.core.addon.AntiVirus method*), 295

`log_moderation` (*smc.core.engine.Engine property*), 279

`log_moderation` (*smc.core.interfaces.Interface property*), 329

`log_payload_excerpt` (*smc.policy.rule\_elements.LogOptions property*), 471

`log_payload_record` (*smc.policy.rule\_elements.LogOptions property*), 472

`log_server` (*smc.core.engine.Engine property*), 280

`log_severity` (*smc.policy.rule\_elements.LogOptions property*), 472

`log_target_types()` (*in smc.administration.scheduled\_tasks*), 152

- LogField (class in *smc\_monitoring.models.constants*), 62
- logging\_subsystem (*smc.core.node.HardwareStatus* property), 368
- LoggingProfile (class in *smc.elements.ssm*), 250
- LogicalInterface (class in *smc.elements.other*), 241, 249
- LOGID (*smc\_monitoring.models.constants.LogField* attribute), 70
- LOGIFTOPDESTINATIONIPADDRS (*smc\_monitoring.models.constants.LogField* attribute), 70
- LOGIFTOPSOURCEIPADDRS (*smc\_monitoring.models.constants.LogField* attribute), 70
- login() (*smc.api.session.Session* method), 105
- LogModeration (class in *smc.core.advanced\_settings*), 386
- LogOptions (class in *smc.policy.rule\_elements*), 470
- logout() (*smc.api.session.Session* method), 107
- LogQuery (class in *smc\_monitoring.monitors.logs*), 91
- LogServer (class in *smc.elements.servers*), 217
- LOGSEVERITY (*smc\_monitoring.models.constants.LogField* attribute), 70
- LONGMSG (*smc\_monitoring.models.constants.LogField* attribute), 70
- loopback\_endpoint (*smc.core.engine.VPN* property), 292
- loopback\_interface (*smc.core.engine.Engine* property), 280
- loopback\_interface (*smc.core.node.Node* property), 362
- LoopbackClusterInterface (class in *smc.core.sub\_interfaces*), 337, 351
- LoopbackCollection (class in *smc.core.collection*), 314
- LoopbackInterface (class in *smc.core.sub\_interfaces*), 336, 352
- LOW (*smc\_monitoring.models.constants.Alerts* attribute), 62
- ## M
- MacAddress (class in *smc.elements.other*), 241, 249
- macaddress (*smc.core.interfaces.ClusterPhysicalInterface* property), 346
- MACALG (*smc\_monitoring.models.constants.LogField* attribute), 70
- management\_address (*smc.core.lldp.LLDPProfile* property), 265
- ManagementLogServerMixin (class in *smc.elements.servers*), 212
- ManagementServer (class in *smc.elements.servers*), 219
- manager (*smc.api.session.Session* property), 107
- massive\_delete() (*smc.administration.system.System* method), 162
- massive\_license\_bind (*smc.administration.system.System* property), 163
- master\_node (*smc.administration.monitoring\_status.EngineNodeStatus* property), 516
- master\_node (*smc.core.node.ApplianceStatus* property), 366
- MasterEngine (class in *smc.core.engines*), 397
- MasterEngineCluster (class in *smc.core.engines*), 398
- MasterNode (class in *smc.core.node*), 358
- match\_condition (*smc.routing.route\_map.RouteMapRule* property), 405
- match\_vpn\_options (*smc.policy.rule.Rule* property), 451
- MatchCondition (class in *smc.routing.route\_map*), 401
- MatchExpression (class in *smc.policy.rule\_elements*), 473
- members (*smc.elements.group.GroupMixin* property), 204
- members (*smc.elements.netlink.Multilink* property), 189
- MESSAGEID (*smc\_monitoring.models.constants.LogField* attribute), 70
- methods (*smc.policy.rule\_elements.AuthenticationOptions* property), 473
- Metric (class in *smc.routing.route\_map*), 405
- mgt\_integration\_configuration (*smc.administration.system.System* property), 163
- mgt\_integration\_container (*smc.elements.servers.ManagementServer* property), 221
- MissingDependency, 528
- MissingRequiredInput, 528
- mobile\_gateway\_node (*smc.vpn.policy.PolicyVPN* property), 479
- mobile\_vpn (*smc.policy.rule\_elements.Action* property), 467
- mobile\_vpn\_topology (*smc.vpn.policy.PolicyVPN* property), 479
- modem\_interface (*smc.core.engine.Engine* property), 280
- ModificationAborted, 528
- ModificationFailed, 528
- modified\_by (*smc.core.resource.History* property), 114
- module
- smc.administration.certificates.tls*, 128
  - smc.administration.certificates.tls\_common*, 126
  - smc.administration.license*, 139
  - smc.administration.monitoring\_status*, 515
  - smc.administration.reports*, 153
  - smc.administration.role*, 122

- smc.administration.scheduled\_tasks, 139
- smc.administration.system, 156
- smc.administration.tasks, 165
- smc.administration.upcoming\_event, 168
- smc.administration.updates, 169
- smc.api.common, 522
- smc.api.exceptions, 525
- smc.api.web, 522
- smc.base.collection, 502
- smc.base.structs, 513
- smc.core.addon, 294
- smc.core.advanced\_settings, 386
- smc.core.collection, 310
- smc.core.contact\_address, 355
- smc.core.engine, 266
- smc.core.engines, 387
- smc.core.interfaces, 326
- smc.core.node, 358
- smc.core.resource, 370
- smc.core.route, 371
- smc.core.sub\_interfaces, 350
- smc.core.waiters, 523
- smc.elements.group, 204
- smc.elements.netlink, 185
- smc.elements.network, 174
- smc.elements.other, 235
- smc.elements.profiles, 262
- smc.elements.protocols, 199
- smc.elements.servers, 210
- smc.elements.service, 193
- smc.elements.situations, 256
- smc.elements.ssm, 250
- smc.elements.user, 116
- smc.policy.file\_filtering, 437
- smc.policy.interface, 435
- smc.policy.ips, 443
- smc.policy.layer2, 446
- smc.policy.layer3, 439
- smc.policy.policy, 433
- smc.policy.qos, 449
- smc.policy.rule\_elements, 464
- smc.policy.rule\_nat, 475
- smc.routing.access\_list, 405
- smc.routing.bgp, 409
- smc.routing.ospf, 424
- smc.routing.prefix\_list, 408
- smc.routing.route\_map, 400
- smc.vpn.elements, 171, 489
- smc.vpn.route, 480
- smc\_monitoring.models, 51
- smc\_monitoring.models.calendar, 82
- smc\_monitoring.models.constants, 61
- smc\_monitoring.models.filters, 52
- smc\_monitoring.models.formats, 58

- smc\_monitoring.models.formatters, 81
- smc\_monitoring.models.query, 47
- smc\_monitoring.models.values, 56
- smc\_monitoring.monitors, 85
- smc\_monitoring.monitors.alerts, 102
- smc\_monitoring.monitors.blacklist, 85
- smc\_monitoring.monitors.connections, 88
- smc\_monitoring.monitors.logs, 90
- smc\_monitoring.monitors.routes, 93
- smc\_monitoring.monitors.sslvpn, 95
- smc\_monitoring.monitors.users, 97
- smc\_monitoring.monitors.vpns, 99
- monitoring\_state (*smc.administration.monitoring\_status.MonitoringStatus* property), 517
- monitoring\_status (*smc.administration.monitoring\_status.MonitoringStatus* property), 518
- MonitoringStatus (class in *smc.administration.monitoring\_status*), 517
- move\_rule\_after() (*smc.policy.rule.Rule* method), 451
- move\_rule\_before() (*smc.policy.rule.Rule* method), 451
- mss\_enforced (*smc.policy.rule\_elements.ConnectionTracking* property), 469
- mss\_enforced\_min\_max (*smc.policy.rule\_elements.ConnectionTracking* property), 469
- mtu (*smc.core.interfaces.PhysicalInterface* property), 341
- multicast\_ip (*smc.core.interfaces.PhysicalInterface* property), 341
- MultiContactAddress (class in *smc.elements.servers*), 214
- Multilink (class in *smc.elements.netlink*), 187
- MultilinkMember (class in *smc.elements.netlink*), 190

## N

- name (*smc.administration.monitoring\_status.MonitoringStatus* property), 518
- name (*smc.api.session.Session* property), 107
- name (*smc.base.model.Element* property), 112
- name (*smc.base.model.UserElement* property), 113
- name (*smc.core.engine.InternalEndpoint* property), 308
- name (*smc.core.interfaces.Interface* property), 330
- name (*smc.core.node.ApplianceStatus* property), 366
- name (*smc.core.route.RoutingTree* property), 375
- name (*smc.elements.other.ContactAddress* property), 240
- name (*smc.elements.protocols.ProtocolParameterValue* property), 203
- name (*smc.policy.rule.Rule* property), 452
- name (*smc.vpn.policy.GatewayNode* property), 496
- nat (*smc.vpn.policy.PolicyVPN* property), 479
- nat\_enforce (*smc.core.engine.LBFilter* property), 290
- NATBALANCEID (*smc\_monitoring.models.constants.LogField* attribute), 70

- NATDPORT (*smc\_monitoring.models.constants.LogField attribute*), 70
- NATDST (*smc\_monitoring.models.constants.LogField attribute*), 70
- NATElement (*class in smc.policy.rule\_nat*), 475
- NATMAPID (*smc\_monitoring.models.constants.LogField attribute*), 70
- NATRule (*class in smc.policy.rule\_nat*), 459
- NATRULEID (*smc\_monitoring.models.constants.LogField attribute*), 70
- NATSPORT (*smc\_monitoring.models.constants.LogField attribute*), 70
- NATSRC (*smc\_monitoring.models.constants.LogField attribute*), 70
- NATT (*smc\_monitoring.models.constants.LogField attribute*), 70
- ndi\_interfaces (*smc.core.interfaces.PhysicalInterface property*), 341
- negotiation\_role (*smc\_monitoring.monitors.vpns.VPNSNodeInterface property*), 101
- NEGOTIATIONROLE (*smc\_monitoring.models.constants.LogField attribute*), 70
- neighbor\_ip (*smc.routing.bgp.ExternalBGPPeer property*), 415
- neighbor\_port (*smc.routing.bgp.ExternalBGPPeer property*), 415
- NEIGHBORINTERFACE (*smc\_monitoring.models.constants.LogField attribute*), 71
- NEIGHBORL2DATA (*smc\_monitoring.models.constants.LogField attribute*), 71
- NEIGHBORL3DATA (*smc\_monitoring.models.constants.LogField attribute*), 71
- NEIGHBORPROTOCOL (*smc\_monitoring.models.constants.LogField attribute*), 71
- NEIGHBORSTATE (*smc\_monitoring.models.constants.LogField attribute*), 71
- netflow\_collector (*smc.elements.servers.ManagementLogServer property*), 212
- netflow\_collector\_port (*smc.elements.servers.NetflowCollector property*), 215
- netflow\_collector\_service (*smc.elements.servers.NetflowCollector property*), 215
- netflow\_collector\_version (*smc.elements.servers.NetflowCollector property*), 215
- NetflowCollector (*class in smc.elements.servers*), 215
- netlink\_role (*smc.elements.netlink.MultilinkMember property*), 191
- netlinks (*smc.core.route.Routing property*), 380
- Network (*class in smc.elements.network*), 182
- network\_application\_latency\_monitoring (*smc.policy.rule\_elements.Action property*), 467
- nicid (*smc.core.route.RoutingTree property*), 375
- Node (*class in smc.core.node*), 358
- NODECAPACITY (*smc\_monitoring.models.constants.LogField attribute*), 71
- NodeCommandFailed, 528
- NODECONFIGURATION (*smc\_monitoring.models.constants.LogField attribute*), 71
- NODECONFIGURATIONTIMESTAMP (*smc\_monitoring.models.constants.LogField attribute*), 71
- NODEDYNUP (*smc\_monitoring.models.constants.LogField attribute*), 71
- NODEHWSTATUS (*smc\_monitoring.models.constants.LogField attribute*), 71
- nodeid (*smc.core.node.Node property*), 362
- NODEID (*smc\_monitoring.models.constants.LogField attribute*), 71
- NodeInterface (*class in smc.core.sub\_interfaces*), 353
- NODELOAD (*smc\_monitoring.models.constants.LogField attribute*), 71
- nodes (*smc.core.engine.Engine property*), 280
- NodeStateWaiter (*class in smc.core.waiters*), 524
- NODESTATUS (*smc\_monitoring.models.constants.LogField attribute*), 71
- NodeStatusWaiter (*class in smc.core.waiters*), 524
- NODEVERSION (*smc\_monitoring.models.constants.LogField attribute*), 71
- NodeWaiter (*class in smc.core.waiters*), 524
- NONCONTAINEDDATATAG (*smc\_monitoring.models.constants.LogField attribute*), 71
- NotFilter (*class in smc\_monitoring.models.filters*), 54
- NotMonitored, 528
- ntp\_auth\_key (*smc.elements.servers.NTPServer property*), 233
- ntp\_auth\_key\_id (*smc.elements.servers.NTPServer property*), 233
- ntp\_auth\_key\_type (*smc.elements.servers.NTPServer property*), 233
- ntp\_enable (*smc.core.general.NTPSettings property*), 306
- ntp\_host\_name (*smc.elements.servers.NTPServer property*), 233
- ntp\_servers (*smc.core.general.NTPSettings property*), 306
- ntp\_settings (*smc.core.engine.Engine property*), 280
- NTPServer (*class in smc.elements.servers*), 233
- NTPSettings (*class in smc.core.general*), 306
- NUMALERTRESPONSES (*smc\_monitoring.models.constants.LogField attribute*), 71
- NumberValue (*class in smc\_monitoring.models.values*), 57
- NUMBLACKLISTRESPONSES

- (smc\_monitoring.models.constants.LogField attribute)*, 71
- NUMBLOCK\_LISTRESPONSES (*smc\_monitoring.models.constants.LogField attribute*), 71
- NUMBYTESRECEIVED (*smc\_monitoring.models.constants.LogField attribute*), 71
- NUMBYTESENT (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMDISCARDRESPONSES (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMLOGEVENTS (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMLOGRESPONSES (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMPACKETSRECEIVED (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMPACKETSENT (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMRECORDRESPONSES (*smc\_monitoring.models.constants.LogField attribute*), 72
- NUMRESETRESPONSES (*smc\_monitoring.models.constants.LogField attribute*), 72
- O**
- object\_types() (*smc.base.collection.Search static method*), 513
- OBJECTDN (*smc\_monitoring.models.constants.LogField attribute*), 72
- OBJECTKEY (*smc\_monitoring.models.constants.LogField attribute*), 72
- OBJECTNAME (*smc\_monitoring.models.constants.LogField attribute*), 72
- OBJECTTYPE (*smc\_monitoring.models.constants.LogField attribute*), 72
- obtain\_members() (*smc.elements.group.GroupMixin method*), 204
- open() (*smc.vpn.policy.PolicyVPN method*), 479
- options (*smc.policy.rule.Rule property*), 452
- order (*smc.elements.situations.SituationParameter property*), 260
- ordered\_url (*smc.elements.other.UpdateServerProfile property*), 242
- OrFilter (*class in smc\_monitoring.models.filters*), 54
- ORIGINNAME (*smc\_monitoring.models.constants.LogField attribute*), 72
- OSPF (*class in smc.routing.ospf*), 424
- ospf\_areas (*smc.core.route.Routing property*), 380
- OSPFArea (*class in smc.routing.ospf*), 426
- OSPFDomainSetting (*class in smc.routing.ospf*), 431
- OSPFInterfaceSetting (*class in smc.routing.ospf*), 432
- OSPFKeyChain (*class in smc.routing.ospf*), 428
- OSPFProfile (*class in smc.routing.ospf*), 429
- outbound\_traffic\_in\_bits\_per\_sec (*smc.administration.monitoring\_status.SDWANNetlinkElementStatus property*), 520
- OUTBOUNDSPI (*smc\_monitoring.models.constants.LogField attribute*), 72
- outgoing (*smc.core.interfaces.InterfaceOptions property*), 332
- P**
- package\_id (*smc.administration.updates.UpdatePackage property*), 171
- PackageMixin (*class in smc.administration.updates*), 169
- packet\_loss\_in\_permyriad (*smc.administration.monitoring\_status.SDWANBranchToBranchStatus property*), 519
- packet\_loss\_in\_permyriad (*smc.administration.monitoring\_status.SDWANNetlinkElementStatus property*), 520
- packet\_loss\_in\_permyriad (*smc.administration.monitoring\_status.VPNStatus property*), 521
- parameter\_values (*smc.elements.situations.Situation property*), 259
- params (*smc.api.common.SMCRequest attribute*), 522
- parent\_policy (*smc.policy.rule.Rule property*), 452
- PASSEDBYTES (*smc\_monitoring.models.constants.LogField attribute*), 72
- password\_meta\_data (*smc.elements.user.AdminUser property*), 118
- PasswordMetaData (*class in smc.elements.user*), 118
- peer\_endpoint (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc property*), 101
- peer\_gateway (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc property*), 101
- peer\_networks (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc property*), 101
- PEERCOMPONENTID (*smc\_monitoring.models.constants.LogField attribute*), 72
- PEERENDPOINT (*smc\_monitoring.models.constants.LogField attribute*), 72
- PEERSECURITYGATEWAY (*smc\_monitoring.models.constants.LogField attribute*), 72
- pending\_changes (*smc.core.engine.Engine property*), 280
- PendingChanges (*class in smc.core.resource*), 370
- period\_begin (*smc.administration.reports.Report property*), 154
- period\_end (*smc.administration.reports.Report property*), 154
- Permission (*class in smc.administration.access\_rights*), 121

- permissions (*smc.administration.access\_rights.AccessControlList* property), 115
- permissions (*smc.administration.role.Role* property), 123
- permissions (*smc.core.engine.Engine* property), 281
- permissions (*smc.elements.user.UserMixin* property), 119
- PERMIT (*smc\_monitoring.models.constants.Actions* attribute), 61
- PFSDHGROUP (*smc\_monitoring.models.constants.LogField* attribute), 72
- PHASE1FAIL (*smc\_monitoring.models.constants.LogField* attribute), 73
- PHASE1SUCC (*smc\_monitoring.models.constants.LogField* attribute), 73
- PHASE2FAIL (*smc\_monitoring.models.constants.LogField* attribute), 73
- PHASE2SUCC (*smc\_monitoring.models.constants.LogField* attribute), 73
- physical\_interface (*smc.core.engine.Engine* property), 281
- physical\_interface (*smc.core.engine.InternalEndpoint* property), 308
- PhysicalInterface (class in *smc.core.interfaces*), 338
- PhysicalInterfaceCollection (class in *smc.core.collection*), 315
- pki\_certificate\_info() (*smc.core.node.Node* method), 362
- pki\_certificate\_info() (*smc.elements.servers.ManagementLogServerMixin* method), 213
- pki\_certificate\_settings() (*smc.core.node.Node* method), 362
- pki\_certificate\_settings() (*smc.elements.servers.ManagementLogServerMixin* method), 213
- pki\_delete\_certificate\_request() (*smc.core.node.Node* method), 362
- pki\_delete\_certificate\_request() (*smc.elements.servers.ManagementLogServerMixin* method), 213
- pki\_export\_certificate\_request() (*smc.core.node.Node* method), 363
- pki\_export\_certificate\_request() (*smc.elements.servers.ManagementLogServerMixin* method), 213
- pki\_import\_certificate() (*smc.core.node.Node* method), 363
- pki\_import\_certificate() (*smc.elements.servers.ManagementLogServerMixin* method), 213
- pki\_renew\_certificate() (*smc.core.node.Node* method), 363
- pki\_renew\_certificate() (*smc.elements.servers.ManagementLogServerMixin* method), 213
- platform (*smc.administration.access\_rights.AccessControlList* property), 115
- platform (*smc.administration.monitoring\_status.EngineNodeStatus* property), 516
- platform (*smc.administration.updates.EngineUpgrade* property), 170
- platform (*smc.core.node.ApplianceStatus* property), 367
- Policy (class in *smc.policy.policy*), 433
- policy (*smc.core.general.Layer2Settings* property), 306
- policy\_route (*smc.core.engine.Engine* property), 281
- policy\_validation\_settings() (in module *smc.administration.scheduled\_tasks*), 152
- PolicyCommandFailed, 528
- PolicyRoute (class in *smc.core.route*), 383
- PolicyVPN (class in *smc.vpn.policy*), 477
- port (*smc.elements.servers.AuthenticationServerMixin* property), 211
- port (*smc.elements.servers.ElasticsearchCluster* property), 235
- port (*smc.elements.servers.SmtServer* property), 230
- port (*smc.routing.bgp.BGPPProfile* property), 419
- port\_description (*smc.corelldp.LLDPProfile* property), 266
- port\_group\_interface (*smc.core.interfaces.Interface* property), 330
- port\_id (*smc.corelldp.LLDPProfile* property), 266
- pos (*smc.core.engine.HAForSingleEngine* property), 289
- POTENTIALLYDUPLICATERESPONSE (*smc\_monitoring.models.constants.LogField* attribute), 73
- power\_off() (*smc.core.node.Node* method), 363
- PrefixListEntry (class in *smc.routing.prefix\_list*), 409
- prepare\_blacklist() (in module *smc.elements.other*), 243
- prepare\_block\_list() (in module *smc.elements.other*), 243
- prepend() (*smc.core.general.RankedDNSAddress* method), 302
- preshared\_key() (*smc.vpn.policy.GatewayTunnel* method), 500
- primary\_heartbeat (*smc.core.interfaces.InterfaceOptions* property), 332
- primary\_mgt (*smc.core.interfaces.InterfaceOptions* property), 333
- probe\_ecmp (*smc.core.route.RoutingTree* property), 375
- probe\_interval (*smc.core.route.RoutingTree* property), 375
- probe\_ipaddress (*smc.core.route.RoutingTree* property), 375
- probe\_metric (*smc.core.route.RoutingTree* property), 375
- probe\_retry\_count (*smc.core.route.RoutingTree* property), 375

- probe\_test (*smc.core.route.RoutingTree* property), 375
- PROBEFAIL (*smc\_monitoring.models.constants.LogField* attribute), 73
- PROBEOK (*smc\_monitoring.models.constants.LogField* attribute), 73
- ProbingProfile (class in *smc.elements.ssm*), 250
- program\_number (*smc.elements.service.RPCService* property), 194
- progress (*smc.administration.tasks.Task* property), 166
- PROTOCOL (*smc\_monitoring.models.constants.LogField* attribute), 73
- protocol (*smc\_monitoring.monitors.alerts.Alert* property), 103
- protocol (*smc\_monitoring.monitors.blacklist.BlacklistEntry* property), 86
- protocol (*smc\_monitoring.monitors.connections.Connection* property), 89
- protocol (*smc\_monitoring.monitors.vpns.VPNSecurityAssessment* property), 101
- protocol\_agent (*smc.elements.protocols.ProtocolAgentMixin* property), 200
- protocol\_agent (*smc.elements.service.ProtocolAgentMixin* property), 193
- protocol\_agent\_ref (*smc.elements.service.EthernetService* property), 194
- protocol\_agent\_values (*smc.elements.protocols.ProtocolAgentMixin* property), 200
- protocol\_agent\_values (*smc.elements.service.ProtocolAgentMixin* property), 193
- protocol\_number (*smc.elements.service.IPService* property), 197
- ProtocolAgent (class in *smc.elements.protocols*), 200
- ProtocolAgentMixin (class in *smc.elements.protocols*), 200
- ProtocolAgentMixin (class in *smc.elements.service*), 193
- ProtocolAgentValues (class in *smc.elements.protocols*), 201
- ProtocolParameterValue (class in *smc.elements.protocols*), 202
- provided\_method (*smc.elements.servers.AuthenticationServerMixin* property), 211
- proxy\_server (*smc.elements.protocols.ProxyServiceValue* property), 203
- proxy\_service (*smc.elements.servers.ProxyServer* property), 226
- ProxyServer (class in *smc.elements.servers*), 225
- ProxyServiceValue (class in *smc.elements.protocols*), 203
- Q
- QoS (class in *smc.core.interfaces*), 335
- qos (*smc.core.interfaces.PhysicalInterface* property), 341
- qos (*smc.core.interfaces.TunnelInterface* property), 348
- qos (*smc.core.interfaces.VlanInterface* property), 349
- qos\_class (*smc.policy.rule\_elements.LogOptions* property), 472
- qos\_limit (*smc.core.interfaces.QoS* property), 336
- qos\_mode (*smc.core.interfaces.QoS* property), 336
- qos\_policy (*smc.core.interfaces.QoS* property), 336
- QoSClass (class in *smc.policy.qos*), 449
- QOSCLASS (*smc\_monitoring.models.constants.LogField* attribute), 73
- QoSPolicy (class in *smc.policy.qos*), 449
- QOSPRIORITY (*smc\_monitoring.models.constants.LogField* attribute), 73
- Query (class in *smc\_monitoring.models.query*), 47
- query\_route() (*smc.core.engine.Engine* method), 281
- QueryDataFilter (class in *smc.elements.other*), 242
- quic\_enabled (*smc.core.engines.FirewallCluster* property), 397
- quic\_enabled (*smc.core.engines.Layer3Firewall* property), 392
- quic\_enabled (*smc.core.engines.Layer3VirtualEngine* property), 394
- R
- RADIUSACCOUNTINGTYPE (*smc\_monitoring.models.constants.LogField* attribute), 73
- RadiusServer (class in *smc.elements.servers*), 232
- RankedDNSAddress (class in *smc.core.general*), 302
- RawDictFormat (class in *smc\_monitoring.models.formatters*), 82
- RawFormat (class in *smc\_monitoring.models.formats*), 60
- reboot() (*smc.core.node.Node* method), 363
- RECEIVEDLOGEVENTS (*smc\_monitoring.models.constants.LogField* attribute), 73
- RECEPTIONTIME (*smc\_monitoring.models.constants.LogField* attribute), 73
- redistribution\_entry (*smc.routing.bgp.BGPProfile* property), 419
- RedistributionEntry (class in *smc.routing.bgp*), 412
- referenced\_by (*smc.base.model.Element* property), 112
- references\_by\_element() (*smc.administration.system.System* method), 163
- refresh() (*smc.api.session.Session* method), 107
- refresh() (*smc.core.engine.Engine* method), 282
- RefreshMasterEnginePolicyTask (class in *smc.administration.scheduled\_tasks*), 144
- RefreshPolicyTask (class in *smc.administration.scheduled\_tasks*), 144
- REFUSE (*smc\_monitoring.models.constants.Actions* attribute), 61

[related\\_element\\_type](#) (*smc.core.route.RoutingTree* property), 375  
[release\\_date](#) (*smc.administration.updates.EngineUpgradeTask* property), 170  
[release\\_date](#) (*smc.administration.updates.UpdatePackageTask* property), 171  
[release\\_notes](#) (*smc.administration.updates.PackageMixin* property), 170  
[remote\\_address](#) (*smc.vpn.route.TunnelEndpoint* property), 488  
[remote\\_endpoint](#) (*smc.vpn.route.RouteVPN* property), 486  
[RemoteUpgradeTask](#) (class in *smc.administration.scheduled\_tasks*), 145  
[remove\(\)](#) (*smc.core.advanced\_settings.LogModeration* method), 387  
[remove\(\)](#) (*smc.core.engine.IdleTimeout* method), 289  
[remove\(\)](#) (*smc.core.engine.InternalGateway* method), 309  
[remove\(\)](#) (*smc.core.engine.VPN* method), 292  
[remove\(\)](#) (*smc.core.general.RankedDNSAddress* method), 302  
[remove\(\)](#) (*smc.core.route.Antispoofing* method), 382  
[remove\\_alternative\\_policies\(\)](#) (*smc.core.engine.Engine* method), 282  
[remove\\_category\(\)](#) (*smc.elements.other.CategoryTag* method), 240, 248  
[remove\\_condition\(\)](#) (*smc.routing.route\_map.MatchCondition* method), 402  
[remove\\_contact\\_address\(\)](#) (*smc.core.contact\_address.ContactAddressNode* method), 357  
[remove\\_contact\\_address\(\)](#) (*smc.elements.servers.ContactAddressMixin* method), 212  
[remove\\_element\(\)](#) (*smc.elements.other.Category* method), 238, 246  
[remove\\_entry\(\)](#) (*smc.routing.access\_list.AccessList* method), 406  
[remove\\_link\\_usage\\_exception\\_rules\(\)](#) (*smc.core.engine.Engine* method), 282  
[remove\\_netflow\\_collector\(\)](#) (*smc.elements.servers.ManagementLogServerMixin* method), 213  
[remove\\_permission\(\)](#) (*smc.administration.access\_rights.AccessControlList* method), 115  
[remove\\_route\\_gateway\(\)](#) (*smc.core.route.Routing* method), 380  
[remove\\_tls\\_credential\(\)](#) (*smc.core.addon.TLSInspection* method), 300  
[rename\(\)](#) (*smc.base.model.Element* method), 112  
[rename\(\)](#) (*smc.core.engine.Engine* method), 282  
[rename\(\)](#) (*smc.core.engine.VPN* method), 293  
[rename\(\)](#) (*smc.core.node.Node* method), 363  
[RenewGatewayCertificatesTask](#) (class in *smc.administration.scheduled\_tasks*), 145  
[RenewInternalCATask](#) (class in *smc.administration.scheduled\_tasks*), 146  
[RenewInternalCertificatesTask](#) (class in *smc.administration.scheduled\_tasks*), 146  
[replace\\_ip](#) (*smc.core.engine.LBFilter* property), 290  
[Report](#) (class in *smc.administration.reports*), 154  
[report\\_files](#) (*smc.administration.reports.ReportDesign* property), 155  
[ReportDesign](#) (class in *smc.administration.reports*), 154  
[ReportOperation](#) (class in *smc.administration.reports*), 155  
[ReportTemplate](#) (class in *smc.administration.reports*), 156  
[require\\_auth](#) (*smc.policy.rule\_elements.AuthenticationOptions* property), 473  
[reset\\_interface\(\)](#) (*smc.core.interfaces.Interface* method), 330  
[reset\\_to\\_factory\(\)](#) (*smc.core.node.Node* method), 363  
[reset\\_user\\_db\(\)](#) (*smc.core.node.Node* method), 364  
[resolve\(\)](#) (*smc.elements.network.Alias* method), 175  
[resolve\\_field\\_ids\(\)](#) (*smc\_monitoring.models.query.Query* static method), 51  
[resource](#) (*smc.administration.tasks.Task* property), 166  
[RESOURCE](#) (*smc\_monitoring.models.constants.LogField* attribute), 73  
[ResourceNotFound](#), 528  
[resources](#) (*smc.administration.scheduled\_tasks.ScheduledTaskMixin* property), 147  
[restart\\_web\\_access\(\)](#) (*smc.elements.servers.ManagementServer* method), 221  
[result](#) (*smc.administration.monitoring\_status.MonitoringStatus* property), 518  
[RESULT](#) (*smc\_monitoring.models.constants.LogField* attribute), 73  
[result\(\)](#) (*smc.administration.tasks.TaskOperationPoller* method), 167  
[result\(\)](#) (*smc.core.waiters.NodeWaiter* method), 525  
[result\\_url](#) (*smc.administration.tasks.Task* property), 166  
[retry](#) (*smc.elements.other.UpdateServerProfile* property), 242  
[RETSRCIF](#) (*smc\_monitoring.models.constants.LogField* attribute), 73  
[Role](#) (class in *smc.administration.role*), 123  
[role](#) (*smc.administration.access\_rights.Permission* property), 121  
[Route](#) (class in *smc.core.route*), 382

[route\\_gw](#) (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
[route\\_map\\_rules](#) (*smc.routing.route\_map.RouteMap* property), 403  
[route\\_metric](#) (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
[route\\_network](#) (*smc\_monitoring.monitors.routes.RoutingView* property), 94  
[route\\_type](#) (*smc\_monitoring.monitors.routes.RoutingView* property), 95  
[ROUTEBGPPATH](#) (*smc\_monitoring.models.constants.LogField* attribute), 73  
[ROUTEDISTANCE](#) (*smc\_monitoring.models.constants.LogField* attribute), 73  
[ROUTEGATEWAY](#) (*smc\_monitoring.models.constants.LogField* attribute), 73  
[RouteMap](#) (class in *smc.routing.route\_map*), 402  
[RouteMapRule](#) (class in *smc.routing.route\_map*), 403  
[ROUTEMETRIC](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[ROUTENETMASK](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[ROUTENETWORK](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[ROUTEOSPFLSATYPE](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[Router](#) (class in *smc.elements.network*), 183  
[router\\_id](#) (*smc.routing.bgp.BGP* property), 412  
[router\\_id](#) (*smc.routing.ospf.OSPF* property), 425  
[ROUTETYPE](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[RouteVPN](#) (class in *smc.vpn.route*), 483  
[RouteVPNTunnelMonitoringGroup](#) (class in *smc.vpn.route*), 487  
[Routing](#) (class in *smc.core.route*), 376  
[routing](#) (*smc.core.engine.Engine* property), 283  
[routing\\_monitoring](#) (*smc.core.engine.Engine* property), 283  
[routing\\_node\\_element](#) (*smc.core.route.Routing* property), 381  
[RoutingQuery](#) (class in *smc\_monitoring.monitors.routes*), 93  
[RoutingTree](#) (class in *smc.core.route*), 373  
[RoutingView](#) (class in *smc\_monitoring.monitors.routes*), 94  
[rpc\\_version](#) (*smc.elements.service.RPCService* property), 194  
[RPCService](#) (class in *smc.elements.service*), 194  
[RpcServiceGroup](#) (class in *smc.elements.group*), 206  
[RTT](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[Rule](#) (class in *smc.policy.rule*), 450  
[rule\\_collection\(\)](#) (in module *smc.base.collection*), 512  
[rule\\_counters\(\)](#) (*smc.policy.policy.Policy* method), 433  
[RULECOUNTERS](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[RuleElement](#) (class in *smc.policy.rule\_elements*), 464  
[RULEHITS](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[RULEID](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[RuleValidityTime](#) (class in *smc.elements.other*), 242  
[run\(\)](#) (*smc.core.waiters.NodeWaiter* method), 525  
[RWPHTTPPREFERRER](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[RWPHTTPUSERAGENT](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[RWPSERVICENAME](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  

## S

[sa\\_type](#) (*smc\_monitoring.monitors.vpns.VPNSecurityAssoc* property), 101  
[SAAUTHALG](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SABUNDLE](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SACIPHERALG](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SACCLASS](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SACOMPRESSIONALG](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SAEXPPIREHARDLIMIT](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SAEXPPIRESOFTLIMIT](#) (*smc\_monitoring.models.constants.LogField* attribute), 74  
[SAINCOMING](#) (*smc\_monitoring.models.constants.LogField* attribute), 75  
[SAKBHARDLIMIT](#) (*smc\_monitoring.models.constants.LogField* attribute), 75  
[SAKBSOFTLIMIT](#) (*smc\_monitoring.models.constants.LogField* attribute), 75  
[Sandbox](#) (class in *smc.core.addon*), 299  
[sandbox](#) (*smc.core.engine.Engine* property), 283  
[sandbox\\_subsystem](#) (*smc.core.node.HardwareStatus* property), 368  
[SARESPONDER](#) (*smc\_monitoring.models.constants.LogField* attribute), 75  
[satellite\\_gateway\\_node](#) (*smc.vpn.policy.PolicyVPN* property), 480  
[SATYPE](#) (*smc\_monitoring.models.constants.LogField* attribute), 75  
[save\(\)](#) (*smc.policy.rule.Rule* method), 452  
[save\(\)](#) (*smc.vpn.policy.PolicyVPN* method), 480

`scan_detection` (*smc.policy.rule\_elements.Action* property), 468  
`ScheduledTaskMixin` (class in *smc.administration.scheduled\_tasks*), 147  
`script_path` (*smc.elements.servers.ManagementServer* property), 221  
`SDWANBranchStatus` (class in *smc.administration.monitoring\_status*), 518  
`SDWANBranchToBranchStatus` (class in *smc.administration.monitoring\_status*), 518  
`SDWANNetlinkElementStatus` (class in *smc.administration.monitoring\_status*), 519  
`Search` (class in *smc.base.collection*), 512  
`search_elements()` (*smc.elements.other.Category* method), 239, 247  
`search_rule()` (*smc.policy.policy.Policy* method), 434  
`search_rule()` (*smc.routing.route\_map.RouteMap* method), 403  
`second_interface_id` (*smc.core.interfaces.PhysicalInterface* property), 341  
`SECURITYGATEWAY` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SELECTEDCACHE` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SELECTEDRTT` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`self_sign()` (*smc.administration.certificates.tls.TLSServerCertificate* method), 132  
`SENDER` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`sender_address` (*smc.elements.servers.ManagementServer* property), 221  
`sender_name` (*smc.elements.servers.ManagementServer* property), 221  
`SENDERDOMAIN` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENDERTYPE` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORALLOWEDINSPECTEDTCPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORALLOWEDINSPECTEDUDPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORALLOWEDUNINSPECTEDTCPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORALLOWEDUNINSPECTEDUDPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORDISCARDEDTCPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORDISCARDEDUDPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORINSPECTEDBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORINSPECTEDPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORINTERFACEKEY` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORLOSTBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORLOSTPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 75  
`SENSORPROCESSEDBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSORPROCESSEDPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSORRECEIVEDBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSORRECEIVEDPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFIC` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICCLOSEDTCPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICINSPECTEDPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICLOSTPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICNEWTCPCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICNUMBEROFALERTS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICOKCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICPROCESSEDBYTES` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICPROCESSEDPACKETS` (*smc\_monitoring.models.constants.LogField* attribute), 76  
`SENSETRAFFICSTATSOFPACKETS`

- `(smc_monitoring.models.constants.LogField attribute)`, 76
- `SENSETRAFFICSUSPICIOUSCONNECTIONS` (`(smc_monitoring.models.constants.LogField attribute)`), 76
- `SENSETRAFFICTCPHANDSHAKES` (`(smc_monitoring.models.constants.LogField attribute)`), 76
- `SENSETRAFFICTCPTIMEOUTS` (`(smc_monitoring.models.constants.LogField attribute)`), 76
- `SENTLOGEVENTS` (`(smc_monitoring.models.constants.LogField attribute)`), 76
- `SerializedIterable` (class in `smc.base.structs`), 514
- `server_credentials` (`(smc.core.addon.TLSInspection property)`), 301
- `server_directories_settings()` (in module `smc.administration.scheduled_tasks`), 152
- `server_directory()` (in module `smc.administration.scheduled_tasks`), 153
- `ServerBackupTask` (class in `smc.administration.scheduled_tasks`), 148
- `ServerStatus` (class in `smc.administration.monitoring_status`), 520
- `Service` (class in `smc.policy.rule_elements`), 466
- `service` (`(smc.policy.rule_elements.Service property)`), 466
- `service` (`(smc_monitoring.monitors.alerts.Alert property)`), 103
- `service` (`(smc_monitoring.monitors.connections.Connection property)`), 89
- `ServiceGroup` (class in `smc.elements.group`), 207
- `SERVICEKEY` (`(smc_monitoring.models.constants.LogField attribute)`), 76
- `services` (`(smc.policy.rule.Rule property)`), 452
- `ServiceValue` (class in `smc_monitoring.models.values`), 57
- `Session` (class in `smc.api.session`), 105
- `session_expiration` (`(smc_monitoring.monitors.sslvpn.SSLVPN property)`), 96
- `session_hold_timer` (`(smc.routing.bgp.BGPConnectionProfile property)`), 420
- `session_id` (`(smc.api.session.Session property)`), 107
- `session_keep_alive` (`(smc.routing.bgp.BGPConnectionProfile property)`), 420
- `session_start` (`(smc_monitoring.monitors.sslvpn.SSLVPN property)`), 96
- `SESSIONDOMAIN` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `SESSIONEVENT` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `SESSIONID` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `SessionManagerNotFound`, 529
- `SessionNotFound`, 529
- `set_admin_domain()` (`(smc.core.engine.VirtualResource method)`), 386
- `set_any()` (`(smc.policy.rule_elements.RuleElement method)`), 465
- `set_auth_request()` (`(smc.core.interfaces.InterfaceOptions method)`), 333
- `set_backup_heartbeat()` (`(smc.core.interfaces.InterfaceOptions method)`), 333
- `set_backup_mgt()` (`(smc.core.interfaces.InterfaceOptions method)`), 333
- `set_connection_sync()` (`(smc.core.engine.HAForSingleEngine method)`), 289
- `set_debug()` (`(smc.core.node.Node method)`), 364
- `set_ha_backup_unit()` (`(smc.core.engine.HAForSingleEngine method)`), 289
- `set_none()` (`(smc.policy.rule_elements.RuleElement method)`), 465
- `set_none()` (`(smc.policy.rule_nat.NATElement method)`), 475
- `set_outgoing()` (`(smc.core.interfaces.InterfaceOptions method)`), 334
- `set_preshared_key()` (`(smc.vpn.route.RouteVPN method)`), 486
- `set_primary_heartbeat()` (`(smc.core.interfaces.InterfaceOptions method)`), 334
- `set_primary_mgt()` (`(smc.core.interfaces.InterfaceOptions method)`), 334
- `set_resolving()` (`(smc_monitoring.models.formats.TextFormat method)`), 60
- `set_retry_on_busy()` (`(smc.api.session.Session method)`), 107
- `set_tunnel_group()` (`(smc.vpn.route.RouteVPN method)`), 486
- `Severity` (`(smc.elements.situations.Situation property)`), 259
- `Severity` (`(smc_monitoring.monitors.alerts.Alert property)`), 103
- `SFPINGRESS` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `sginfo()` (`(smc.core.node.Node method)`), 364
- `SGInfoTask` (class in `smc.administration.scheduled_tasks`), 146
- `SHAPINGCLASS` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `SHAPINGGUARANTEE` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `SHAPINGLIMIT` (`(smc_monitoring.models.constants.LogField attribute)`), 77
- `SHAPINGPRIORITY` (`(smc_monitoring.models.constants.LogField attribute)`), 77

[attribute](#)), 77  
[shared\\_secret](#) (*smc.elements.servers.AuthenticationServerMixin* module, 122  
[property](#)), 211  
[show\\_not\\_categorized](#)  
*(smc.administration.system.AdminDomain*  
*property)*, 138, 158  
[sidewinder\\_proxy](#) (*smc.core.engine.Engine* property),  
284  
[SidewinderProxy](#) (class in *smc.core.addon*), 296  
[single\\_ip\\_ha](#) (*smc.core.node.HardwareStatus* prop-  
erty), 369  
[SingleNodeInterface](#) (class in  
*smc.core.sub\_interfaces*), 354  
[SITCATEGORY](#) (*smc\_monitoring.models.constants.LogField*  
attribute), 77  
[site\\_element](#) (*smc.vpn.elements.VPNSite* property),  
494  
[sites](#) (*smc.core.engine.VPN* property), 293  
[Situation](#) (class in *smc.elements.situations*), 258  
[situation](#) (*smc.administration.upcoming\_event.UpcomingEvent*  
property), 168  
[situation](#) (*smc.administration.upcoming\_event.UpcomingEventPublicEntry*  
property), 169  
[SITUATION](#) (*smc\_monitoring.models.constants.LogField*  
attribute), 77  
[situation](#) (*smc\_monitoring.monitors.alerts.Alert* prop-  
erty), 103  
[situation\\_parameters](#)  
*(smc.elements.situations.CorrelationSituationContext*  
*property)*, 257  
[situation\\_parameters](#)  
*(smc.elements.situations.SituationContext*  
*property)*, 259  
[SituationContext](#) (class in *smc.elements.situations*),  
259  
[SituationContextGroup](#) (class in  
*smc.elements.situations*), 259  
[SituationParameter](#) (class in  
*smc.elements.situations*), 260  
[SituationParameterValue](#) (class in  
*smc.elements.situations*), 260  
[situations](#) (*smc.policy.file\_filtering.FileFilteringRule*  
property), 438  
[SituationTag](#) (class in *smc.elements.other*), 242  
[smc.administration.certificates.tls](#)  
module, 128  
[smc.administration.certificates.tls\\_common](#)  
module, 126  
[smc.administration.license](#)  
module, 139  
[smc.administration.monitoring\\_status](#)  
module, 515  
[smc.administration.reports](#)  
module, 153  
[smc.administration.role](#)  
module, 122  
[smc.administration.scheduled\\_tasks](#)  
module, 139  
[smc.administration.system](#)  
module, 156  
[smc.administration.tasks](#)  
module, 165  
[smc.administration.upcoming\\_event](#)  
module, 168  
[smc.administration.updates](#)  
module, 169  
[smc.api.common](#)  
module, 522  
[smc.api.exceptions](#)  
module, 525  
[smc.api.web](#)  
module, 522  
[smc.base.collection](#)  
module, 502  
[smc.base.structs](#)  
module, 502  
[smc.core.addon](#)  
module, 294  
[smc.core.advanced\\_settings](#)  
module, 386  
[smc.core.collection](#)  
module, 310  
[smc.core.contact\\_address](#)  
module, 355  
[smc.core.engine](#)  
module, 266  
[smc.core.engines](#)  
module, 387  
[smc.core.interfaces](#)  
module, 326  
[smc.core.node](#)  
module, 358  
[smc.core.resource](#)  
module, 370  
[smc.core.route](#)  
module, 371  
[smc.core.sub\\_interfaces](#)  
module, 350  
[smc.core.waiters](#)  
module, 523  
[smc.elements.group](#)  
module, 204  
[smc.elements.netlink](#)  
module, 185  
[smc.elements.network](#)  
module, 174  
[smc.elements.other](#)  
module, 235

- smc.elements.profiles
  - module, 262
- smc.elements.protocols
  - module, 199
- smc.elements.servers
  - module, 210
- smc.elements.service
  - module, 193
- smc.elements.situations
  - module, 256
- smc.elements.ssm
  - module, 250
- smc.elements.user
  - module, 116
- smc.policy.file\_filtering
  - module, 437
- smc.policy.interface
  - module, 435
- smc.policy.ips
  - module, 443
- smc.policy.layer2
  - module, 446
- smc.policy.layer3
  - module, 439
- smc.policy.policy
  - module, 433
- smc.policy.qos
  - module, 449
- smc.policy.rule\_elements
  - module, 464
- smc.policy.rule\_nat
  - module, 475
- smc.routing.access\_list
  - module, 405
- smc.routing.bgp
  - module, 409
- smc.routing.ospf
  - module, 424
- smc.routing.prefix\_list
  - module, 408
- smc.routing.route\_map
  - module, 400
- smc.vpn.elements
  - module, 171, 489
- smc.vpn.route
  - module, 480
- smc\_certificate\_authority()
  - (*smc.administration.system.System* method), 163
- smc\_monitoring.models
  - module, 51
- smc\_monitoring.models.calendar
  - module, 82
- smc\_monitoring.models.constants
  - module, 61
- smc\_monitoring.models.filters
  - module, 52
- smc\_monitoring.models.formats
  - module, 58
- smc\_monitoring.models.formatters
  - module, 81
- smc\_monitoring.models.query
  - module, 47
- smc\_monitoring.models.values
  - module, 56
- smc\_monitoring.monitors
  - module, 85
- smc\_monitoring.monitors.alerts
  - module, 102
- smc\_monitoring.monitors.blacklist
  - module, 85
- smc\_monitoring.monitors.connections
  - module, 88
- smc\_monitoring.monitors.logs
  - module, 90
- smc\_monitoring.monitors.routes
  - module, 93
- smc\_monitoring.monitors.sslvpn
  - module, 95
- smc\_monitoring.monitors.users
  - module, 97
- smc\_monitoring.monitors.vpns
  - module, 99
- smc\_time (*smc.administration.system.System* property), 163
- smc\_version (*smc.administration.system.System* property), 163
- SMCConnectionError, 528
- SMCException, 529
- SMCOperationFailure, 529
- SMCRequest (class in *smc.api.common*), 522
- SMCResult (class in *smc.api.web*), 522
- sms\_http\_channel (*smc.elements.servers.ManagementServer* property), 221
- sms\_script\_channel (*smc.elements.servers.ManagementServer* property), 221
- sms\_smtp\_channel (*smc.elements.servers.ManagementServer* property), 221
- smtp\_server\_ref (*smc.elements.servers.ManagementServer* property), 221
- SmtpServer (class in *smc.elements.servers*), 230
- Snapshot (class in *smc.core.resource*), 384
- snapshots (*smc.core.engine.Engine* property), 284
- SNMP (class in *smc.core.general*), 304
- snmp (*smc.core.engine.Engine* property), 284
- snmp\_gateways (*smc.elements.servers.ManagementServer* property), 222
- SNMPAgent (class in *smc.elements.profiles*), 265

SNMPPRETSRCIF (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SNMPSRCIF (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SNMPTRAPMAP (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SNMPTRAPOID (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SNMPTRAPVALUE (*smc\_monitoring.models.constants.LogField attribute*), 77  
 sock (*smc.api.session.Session property*), 108  
 Source (*class in smc.policy.rule\_elements*), 466  
 source (*smc\_monitoring.monitors.alerts.Alert property*), 103  
 source (*smc\_monitoring.monitors.blacklist.BlacklistEntry property*), 87  
 source\_addr (*smc\_monitoring.monitors.connections.Connection property*), 89  
 source\_addr (*smc\_monitoring.monitors.sslvpn.SSLVPNUser property*), 97  
 source\_port (*smc\_monitoring.monitors.alerts.Alert property*), 103  
 source\_port (*smc\_monitoring.monitors.connections.Connection property*), 89  
 source\_ports (*smc\_monitoring.monitors.blacklist.BlacklistEntry property*), 87  
 sources (*smc.policy.rule.Rule property*), 452  
 SPORT (*smc\_monitoring.models.constants.LogField attribute*), 77  
 src (*smc.policy.rule\_elements.Source property*), 466  
 SRC (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SRCADDRESS (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SRCADDRS (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SRCIF (*smc\_monitoring.models.constants.LogField attribute*), 77  
 SRCIPRANGE (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SRCVLAN (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SRCZONE (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SRVHELPERID (*smc\_monitoring.models.constants.LogField attribute*), 78  
 ssh() (*smc.core.node.Node method*), 364  
 SSHKnownHosts (*class in smc.elements.ssm*), 250, 254  
 SSHKnownHostsLists (*class in smc.elements.ssm*), 251, 255  
 SSHProfile (*class in smc.elements.ssm*), 252, 253  
 SSLVPNQuery (*class in smc\_monitoring.monitors.sslvpn*), 95  
 SSLVPNSSESSIONMONID (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SSLVPNSSESSIONMONRECEIVED (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SSLVPNSSESSIONMONTIMEOUT (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SSLVPNSSESSIONTYPETYPE (*smc\_monitoring.models.constants.LogField attribute*), 78  
 SSLVPNUser (*class in smc\_monitoring.monitors.sslvpn*), 96  
 SSMLoggingProfile (*class in smc.elements.ssm*), 253  
 start() (*smc.administration.scheduled\_tasks.ScheduledTaskMixin method*), 147  
 start\_port (*smc.policy.rule\_nat.DynamicSourceNAT property*), 476, 477  
 start\_time (*smc.administration.tasks.Task property*), 166  
 start\_time (*smc\_monitoring.models.calendar.TimeFormat property*), 84  
 STATE (*in module smc.core.waiters*), 525  
 state (*smc.administration.updates.UpdatePackage property*), 171  
 state (*smc.core.node.ApplianceStatus property*), 367  
 state (*smc.policy.rule\_elements.ConnectionTracking property*), 469  
 STATE (*smc\_monitoring.models.constants.LogField attribute*), 78  
 state (*smc\_monitoring.monitors.connections.Connection property*), 89  
 static\_arp\_entry() (*smc.core.interfaces.PhysicalInterface method*), 341  
 static\_dst\_nat (*smc.policy.rule\_nat.NATRule property*), 460  
 static\_src\_nat (*smc.policy.rule\_nat.NATRule property*), 460  
 StaticNetlink (*class in smc.elements.netlink*), 191  
 StaticSourceNAT (*class in smc.policy.rule\_nat*), 476  
 statistics\_only() (*smc.core.interfaces.QoS method*), 336  
 STATUS (*in module smc.core.waiters*), 525  
 status (*smc.core.addon.AntiVirus property*), 295  
 status (*smc.core.addon.ClientInspection property*), 299  
 status (*smc.core.addon.FileReputation property*), 296  
 status (*smc.core.addon.Sandbox property*), 300  
 status (*smc.core.addon.SidewinderProxy property*), 297  
 status (*smc.core.addon.UrlFiltering property*), 297  
 status (*smc.core.addon.ZTNAConnector property*), 298  
 status (*smc.core.general.DefaultNAT property*), 301  
 status (*smc.core.general.DNSRelay property*), 303  
 status (*smc.core.general.SNMP property*), 304  
 status (*smc.core.node.ApplianceStatus property*), 367  
 status (*smc.routing.bgp.BGP property*), 412

`status` (*smc.routing.ospf.OSPF* property), 425  
`status()` (*smc.core.node.Node* method), 365  
`STATUSTYPE` (*smc\_monitoring.models.constants.LogField* attribute), 78  
`stop()` (*smc.administration.tasks.TaskOperationPoller* method), 167  
`stop()` (*smc.core.waiters.NodeWaiter* method), 525  
`STORAGESERVERID` (*smc\_monitoring.models.constants.LogField* attribute), 78  
`StringValue` (class in *smc\_monitoring.models.values*), 58  
`sub_interfaces()` (*smc.core.interfaces.Interface* method), 330  
`sub_policy` (*smc.policy.rule\_elements.Action* property), 468  
`SubElement` (class in *smc.base.model*), 113  
`SubElementCollection` (class in *smc.base.collection*), 509  
`SubInterface` (class in *smc.core.sub\_interfaces*), 354  
`SubInterfaceCollection` (class in *smc.core.sub\_interfaces*), 354  
`subnet_distance` (*smc.routing.bgp.BGPProfile* property), 419  
`SubTlSMatchSituation` (class in *smc.elements.situations*), 260  
`subtract_from_now()` (in module *smc\_monitoring.models.calendar*), 84  
`success` (*smc.administration.tasks.Task* property), 167  
`suspend()` (*smc.administration.scheduled\_tasks.TaskSchedule* method), 149  
`switch_domain()` (*smc.api.session.Session* method), 108  
`switch_physical_interface` (*smc.core.engine.Engine* property), 284  
`SwitchInterfaceCollection` (class in *smc.core.collection*), 321  
`SwitchPhysicalInterface` (class in *smc.core.interfaces*), 347  
`sync_connections` (*smc.policy.rule\_elements.ConnectionTracking* property), 470  
`sync_interface` (*smc.core.engine.HAForSingleEngine* property), 289  
`SYSLOGTYPE` (*smc\_monitoring.models.constants.LogField* attribute), 78  
`System` (class in *smc.administration.system*), 158  
`system_capabilities` (*smc.core.lldp.LLDPProfile* property), 266  
`system_description` (*smc.core.lldp.LLDPProfile* property), 266  
`system_name` (*smc.core.lldp.LLDPProfile* property), 266  
`system_properties()` (*smc.administration.system.System* method), 163  
`system_property()` (*smc.administration.system.System* method), 163  
`SystemSnapshotTask` (class in *smc.administration.scheduled\_tasks*), 148  
**T**  
`TableFormat` (class in *smc\_monitoring.models.formatters*), 82  
`TacacsServer` (class in *smc.elements.servers*), 232  
`tag` (*smc.policy.rule.Rule* property), 452  
`TAGINFO` (*smc\_monitoring.models.constants.LogField* attribute), 78  
`target` (*smc.elements.situations.Situation* property), 259  
`Task` (class in *smc.administration.tasks*), 165  
`task` (*smc.administration.tasks.TaskOperationPoller* property), 167  
`task` (*smc.administration.tasks.TaskProgress* property), 168  
`task_schedule` (*smc.administration.scheduled\_tasks.ScheduledTaskMixin* property), 148  
`TaskHistory()` (in module *smc.administration.tasks*), 167  
`TaskOperationPoller` (class in *smc.administration.tasks*), 167  
`TaskProgress` (class in *smc.administration.tasks*), 168  
`TaskRunFailed`, 529  
`TaskSchedule` (class in *smc.administration.scheduled\_tasks*), 149  
`TCPDUMPSTATUS` (*smc\_monitoring.models.constants.LogField* attribute), 78  
`TCPENCAPSULATION` (*smc\_monitoring.models.constants.LogField* attribute), 78  
`TCPService` (class in *smc.elements.service*), 197  
`TCPServiceGroup` (class in *smc.elements.group*), 208  
`TERMINATE` (*smc\_monitoring.models.constants.Actions* attribute), 61  
`TERMINATE_FAILED` (*smc\_monitoring.models.constants.Actions* attribute), 61  
`TERMINATE_PASSIVE` (*smc\_monitoring.models.constants.Actions* attribute), 61  
`TERMINATE_RESET` (*smc\_monitoring.models.constants.Actions* attribute), 61  
`TextFormat` (class in *smc\_monitoring.models.formats*), 60  
`threshold_in_days` (*smc.administration.upcoming\_event.UpcomingEvent* property), 169  
`time_sync()` (*smc.core.node.Node* method), 365  
`time_to_live` (*smc.core.lldp.LLDPProfile* property), 266  
`TimeFormat` (class in *smc\_monitoring.models.calendar*), 82  
`timeout` (*smc.api.session.Session* property), 108  
`timeout` (*smc.elements.other.UpdateServerProfile* property), 242

`timeout` (`smc.elements.servers.AuthenticationServerMixin` attribute), 211  
`timeout` (`smc.policy.rule_elements.AuthenticationOptions` property), 473  
`timeout` (`smc.policy.rule_elements.ConnectionTracking` property), 470  
`TIMEOUT` (`smc_monitoring.models.constants.LogField` attribute), 78  
`TIMESTAMP` (`smc_monitoring.models.constants.LogField` attribute), 78  
`timestamp` (`smc_monitoring.monitors.alerts.Alert` property), 103  
`timestamp` (`smc_monitoring.monitors.blacklist.BlacklistEntry` property), 87  
`timestamp` (`smc_monitoring.monitors.connections.Connection` property), 89  
`timestamp` (`smc_monitoring.monitors.routes.RoutingView` property), 95  
`timestamp` (`smc_monitoring.monitors.users.User` property), 98  
`timestamp` (`smc_monitoring.monitors.vpns.VPNSecurityAssoc` property), 101  
`timezone()` (`smc_monitoring.models.formats.TextFormat` method), 61  
`tls_credentials` (`smc.elements.servers.ManagementServer` property), 222  
`tls_inspection` (`smc.core.engine.Engine` property), 284  
`tls_policy` (`smc.elements.protocols.TlsInspectionPolicyValue` property), 204  
`tls_profile` (`smc.elements.other.UpdateServerProfile` property), 243  
`tls_profile` (`smc.elements.servers.ManagementServer` property), 222  
`tls_profile` (`smc.elements.servers.NetflowCollector` property), 215  
`tls_profile_ref` (`smc.elements.servers.IcapServer` property), 232  
`tls_trusted_ca_tags` (`smc.core.addon.ClientInspection` property), 299  
`tls_trusted_cas` (`smc.core.addon.ClientInspection` property), 299  
`TLSALERTDESCRIPTION` (`smc_monitoring.models.constants.LogField` attribute), 78  
`TLSALERTLEVEL` (`smc_monitoring.models.constants.LogField` attribute), 78  
`TlSCertificateAuthority` (class in `smc.administration.certificates.tls`), 135  
`TlSCERTIFICATEVERIFYERRORCODE` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TlSCIPHERSUITE` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TLSCOMPRESSIONMETHOD` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TlSCryptographySuite` (class in `smc.administration.certificates.tls`), 134  
`TLSDetected` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TLSDOMAIN` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TlsIdentity` (class in `smc.administration.certificates.tls`), 134  
`TlsIdentity` (`smc.elements.servers.NetflowCollector` property), 215  
`TlsInspection` (class in `smc.core.addon`), 300  
`TlsInspectionPolicyValue` (class in `smc.elements.protocols`), 203  
`TlsMatchSituation` (class in `smc.elements.situations`), 261  
`TlsMatchSituationContext` (class in `smc.elements.situations`), 262  
`TlsProfile` (class in `smc.administration.certificates.tls`), 133  
`TLSPROTOCOLVERSION` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TlsServerCredential` (class in `smc.administration.certificates.tls`), 129  
`TlsSettings` (class in `smc.elements.servers`), 215  
`TOTALBYTES` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPACCEPTEDBYTES` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPACCEPTEDPACKETS` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPDROPPEDBYTES` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPDROPPEDPACKETS` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPMEMUSAGE` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPNODELOAD` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPRECEIVEDBYTES` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPRECEIVEDPACKETS` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPSENTBYTES` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPSENTPACKETS` (`smc_monitoring.models.constants.LogField` attribute), 79  
`TPTRAFFICCOUNTERS` (`smc_monitoring.models.constants.LogField` attribute), 79  
`traffic_in_bits_per_sec` (`smc.administration.monitoring_status.SDWANBranchToBranchS`

property), 519

traffic\_in\_bits\_per\_sec (smc.administration.monitoring\_status.VPNStatus property), 521

TrafficCaptureInterfaceSettings (class in smc.administration.scheduled\_tasks), 149

TrafficCaptureTask (class in smc.administration.scheduled\_tasks), 150

TRAFFICCOUNTERS (smc\_monitoring.models.constants.LogField attribute), 79

TRAFFICSHAPING (smc\_monitoring.models.constants.LogField attribute), 79

TRANSIENT (smc\_monitoring.models.constants.LogField attribute), 80

translated\_value (smc.policy.rule\_nat.DynamicSourceNAT property), 476, 477

translated\_value (smc.policy.rule\_nat.NATElement property), 475

TranslatedFilter (class in smc\_monitoring.models.filters), 54

TranslatedValue (class in smc\_monitoring.models.values), 58

transmit\_delay (smc.core.lldp.LLDPProfile property), 266

transport (smc.elements.service.RPCService property), 194

trust\_all\_cas (smc.vpn.elements.ExternalGateway property), 490

trust\_host\_header (smc.elements.servers.ProxyServer property), 226

tunnel\_interface (smc.core.engine.Engine property), 285

tunnel\_interface (smc.vpn.route.TunnelEndpoint property), 488

tunnel\_mode (smc.vpn.route.RouteVPN property), 486

tunnel\_side\_a (smc.vpn.policy.GatewayTunnel property), 500

tunnel\_side\_b (smc.vpn.policy.GatewayTunnel property), 500

TunnelEndpoint (class in smc.vpn.route), 487

TUNNELINGLEVEL (smc\_monitoring.models.constants.LogField attribute), 80

TunnelInterface (class in smc.core.interfaces), 348

TunnelInterfaceCollection (class in smc.core.collection), 323

TunnelMonitoringGroup (class in smc.vpn.route), 489

tunnels (smc.vpn.policy.PolicyVPN property), 480

tunnels (smc.vpn.route.RouteVPN property), 486

type (smc.core.node.Node property), 365

type (smc.elements.protocols.ProtocolParameterValue property), 203

type (smc.elements.situations.SituationParameter property), 260

TYPE (smc\_monitoring.models.constants.LogField attribute), 80

TYPEDESCRIPTION (smc\_monitoring.models.constants.LogField attribute), 80

## U

UDPService (class in smc.elements.service), 198

UDPServiceGroup (class in smc.elements.group), 209

uuid (smc.elements.servers.ManagementLogServerMixin property), 213

unbind\_license() (smc.core.node.Node method), 365

unique\_id (smc.base.model.UserElement property), 113

unlicensed\_components() (smc.administration.system.System method), 163

unlock() (smc.base.model.Element method), 112

unset\_any() (smc.policy.rule\_elements.RuleElement method), 465

UnsupportedAlertChannel, 529

UnsupportedAttribute, 529

UnsupportedEngineFeature, 529

UnsupportedEntryPoint, 530

UnsupportedInterfaceType, 530

unused() (smc.base.collection.Search method), 513

upcoming\_event() (smc.administration.system.System method), 164

upcoming\_event\_ignore\_settings() (smc.administration.system.System method), 164

upcoming\_event\_policy() (smc.administration.system.System method), 164

UpcomingEvent (class in smc.administration.upcoming\_event), 168

UpcomingEventIgnoreSettings (class in smc.administration.upcoming\_event), 169

UpcomingEvents (class in smc.administration.upcoming\_event), 169

UpcomingEventsPolicy (class in smc.administration.upcoming\_event), 169

UpcomingEventsPolicyEntry (class in smc.administration.upcoming\_event), 169

update() (smc.base.model.ElementBase method), 109

update() (smc.core.interfaces.Interface method), 330

update() (smc.core.route.RoutingTree method), 376

update() (smc.elements.protocols.ProtocolAgentValues method), 201

update() (smc.elements.servers.ElasticsearchCluster method), 235

update() (smc.elements.servers.LogServer method), 218

update() (smc.elements.servers.ManagementServer method), 222

update() (smc.policy.layer3.FirewallPolicy method), 440

update() (smc.policy.rule.Rule method), 452

- `update()` (*smc.policy.rule\_nat.NATRule* method), 460
- `update_configuration()` (*smc.core.general.SNMP* method), 305
- `update_configuration()` (*smc.routing.bgp.BGP* method), 412
- `update_configuration()` (*smc.routing.ospf.OSPF* method), 425
- `update_day()` (*smc.core.addon.AntiVirus* method), 295
- `update_field()` (*smc.policy.rule\_elements.RuleElement* method), 465
- `update_field()` (*smc.policy.rule\_nat.NATElement* method), 475
- `update_filter()` (*smc\_monitoring.models.query.Query* method), 51
- `update_format()` (*smc\_monitoring.models.query.Query* method), 51
- `update_frequency()` (*smc.core.addon.AntiVirus* method), 295
- `update_interface()` (*smc.core.interfaces.Interface* method), 330
- `update_interface()` (*smc.core.interfaces.SwitchPhysicalInterface* method), 347
- `update_members()` (*smc.elements.group.GroupMixin* method), 204
- `update_or_create()` (*smc.base.model.Element* class method), 113
- `update_or_create()` (*smc.core.collection.InterfaceCollection* method), 314
- `update_or_create()` (*smc.core.contact\_address.ContactAddressNode* method), 357
- `update_or_create()` (*smc.elements.group.GroupMixin* class method), 205
- `update_or_create()` (*smc.elements.netlink.Multilink* class method), 189
- `update_or_create()` (*smc.elements.netlink.StaticNetlink* class method), 192
- `update_or_create()` (*smc.elements.network.Alias* class method), 176
- `update_or_create()` (*smc.elements.network.IPList* class method), 181
- `update_or_create()` (*smc.elements.servers.MultiContactAddress* method), 214
- `update_or_create()` (*smc.elements.servers.ProxyServer* class method), 226
- `update_or_create()` (*smc.routing.access\_list.AccessList* class method), 406
- `update_or_create()` (*smc.routing.bgp.AutonomousSystem* class method), 414
- `update_or_create()` (*smc.routing.ospf.OSPFArea* class method), 427
- `update_or_create()` (*smc.routing.ospf.OSPFProfile* class method), 430
- `update_or_create()` (*smc.vpn.elements.ExternalEndpoint* class method), 493
- `update_or_create()` (*smc.vpn.elements.ExternalGateway* class method), 490
- `update_or_create()` (*smc.vpn.elements.VPNSite* class method), 494
- `update_package()` (*smc.administration.system.System* method), 164
- `update_package_import()` (*smc.administration.system.System* method), 164
- `update_protocol_agent()` (*smc.elements.protocols.ProtocolAgentMixin* method), 200
- `update_protocol_agent()` (*smc.elements.service.ProtocolAgentMixin* method), 193
- `update_status()` (*smc.administration.tasks.Task* method), 167
- `update_system_property()` (*smc.administration.system.System* method), 164
- `update_upcoming_event_ignore_settings()` (*smc.administration.system.System* method), 164
- `update_upcoming_event_policy()` (*smc.administration.system.System* method), 164
- `UpdateElementFailed`, 530
- `UpdatePackage` (class in *smc.administration.updates*), 171
- `updates_check_enabled` (*smc.elements.servers.ManagementServer* property), 222
- `updates_proxy_address` (*smc.elements.servers.ManagementServer* property), 222
- `updates_proxy_authentication_enabled` (*smc.elements.servers.ManagementServer* property), 222
- `updates_proxy_enabled` (*smc.elements.servers.ManagementServer* property), 222
- `updates_proxy_password` (*smc.elements.servers.ManagementServer* property), 223
- `updates_proxy_port` (*smc.elements.servers.ManagementServer* property), 223
- `updates_proxy_username` (*smc.elements.servers.ManagementServer* property), 223
- `UpdateServerProfile` (class in *smc.elements.other*), 242
- `upload()` (*smc.core.engine.Engine* method), 285
- `upload()` (*smc.elements.network.IPList* method), 181
- `upload()` (*smc.policy.interface.InterfaceTemplatePolicy*

- method*), 436
  - `upload()` (*smc.policy.ips.IPSTemplatePolicy method*), 445
  - `upload()` (*smc.policy.layer2.Layer2TemplatePolicy method*), 448
  - `upload()` (*smc.policy.layer3.FirewallTemplatePolicy method*), 442
  - `upload()` (*smc.policy.policy.InspectionPolicy method*), 443
  - `upload()` (*smc.policy.policy.Policy method*), 434
  - `upload_alternative_slot()` (*smc.core.engine.Engine method*), 285
  - `UploadPolicyTask` (*class in smc.administration.scheduled\_tasks*), 150
  - `url` (*smc.api.session.Session property*), 108
  - `url_category_logging` (*smc.policy.rule\_elements.LogOptions property*), 472
  - `url_entry` (*smc.elements.network.URLListApplication property*), 185
  - `url_filtering` (*smc.core.engine.Engine property*), 286
  - `URLCategory` (*class in smc.elements.service*), 199
  - `URLCategoryGroup` (*class in smc.elements.group*), 209
  - `URLCATEGORYGROUP` (*smc\_monitoring.models.constants.LogField attribute*), 80
  - `URLCATEGORYRISK` (*smc\_monitoring.models.constants.LogField attribute*), 80
  - `UrlFiltering` (*class in smc.core.addon*), 297
  - `URLListApplication` (*class in smc.elements.network*), 184
  - `use_ipsec` (*smc.core.engine.LBFilter property*), 290
  - `use_ports` (*smc.core.engine.LBFilter property*), 290
  - `used_on` (*smc.elements.other.Location property*), 241, 248
  - `used_on` (*smc.policy.rule\_nat.NATRule property*), 460
  - `User` (*class in smc\_monitoring.monitors.users*), 97
  - `user_alert_check` (*smc.administration.system.AdminDomain property*), 138, 158
  - `user_authentication_import_certificate()` (*smc.core.engine.Engine method*), 286
  - `user_logging` (*smc.policy.rule\_elements.LogOptions property*), 472
  - `user_response` (*smc.policy.rule\_elements.Action property*), 468
  - `UserElement` (*class in smc.base.model*), 113
  - `UserElementNotFound`, 530
  - `UserMixin` (*class in smc.elements.user*), 118
  - `USERNAME` (*smc\_monitoring.models.constants.LogField attribute*), 80
  - `username` (*smc\_monitoring.monitors.sslvpn.SSLVPNUser property*), 97
  - `username` (*smc\_monitoring.monitors.users.User property*), 98
  - `USERORIGINATOR` (*smc\_monitoring.models.constants.LogField attribute*), 80
  - `UserQuery` (*class in smc\_monitoring.monitors.users*), 98
  - `USERROLE` (*smc\_monitoring.models.constants.LogField attribute*), 80
  - `users` (*smc.policy.rule\_elements.AuthenticationOptions property*), 473
- ## V
- `valid_blacklist` (*smc.policy.rule\_elements.Action property*), 468
  - `valid_block_list` (*smc.policy.rule\_elements.Action property*), 468
  - `valid_from` (*smc.administration.certificates.tls.TLSServerCredential property*), 132
  - `valid_to` (*smc.administration.certificates.tls.TLSServerCredential property*), 132
  - `validate()` (*smc.vpn.policy.PolicyVPN method*), 480
  - `ValidatePolicyTask` (*class in smc.administration.scheduled\_tasks*), 151
  - `validity` (*smc.core.route.Antispoofing property*), 382
  - `Value` (*class in smc\_monitoring.models.values*), 58
  - `value` (*smc.elements.protocols.ProtocolParameterValue property*), 203
  - `version` (*smc.administration.monitoring\_status.EngineNodeStatus property*), 516
  - `version` (*smc.administration.updates.EngineUpgrade property*), 170
  - `version` (*smc.core.engine.Engine property*), 286
  - `version` (*smc.core.node.ApplianceStatus property*), 367
  - `version` (*smc.core.node.Node property*), 365
  - `vfw_id` (*smc.core.engine.VirtualResource property*), 386
  - `virtual_engine_vlan_ok` (*smc.core.interfaces.PhysicalInterface property*), 342
  - `virtual_mapping` (*smc.core.interfaces.PhysicalInterface property*), 342
  - `virtual_physical_interface` (*smc.core.engine.Engine property*), 286
  - `virtual_resource` (*smc.core.engine.Engine property*), 287
  - `virtual_resource_name` (*smc.core.interfaces.PhysicalInterface property*), 342
  - `VirtualPhysicalInterface` (*class in smc.core.interfaces*), 346
  - `VirtualPhysicalInterfaceCollection` (*class in smc.core.collection*), 325
  - `VirtualResource` (*class in smc.core.engine*), 385
  - `visible_security_group_mapping()` (*smc.administration.system.System method*), 165
  - `visible_virtual_engine_mapping()` (*smc.administration.system.System method*), 165

- `vlan_id` (*smc.core.interfaces.VlanInterface* property), 349
- `vlan_id` (*smc.core.sub\_interfaces.ClusterVirtualInterface* property), 350
- `vlan_id` (*smc.core.sub\_interfaces.InlineInterface* property), 351
- `vlan_id` (*smc.core.sub\_interfaces.NodeInterface* property), 354
- `vlan_interface` (*smc.core.interfaces.Interface* property), 331
- `VlanInterface` (class in *smc.core.interfaces*), 349
- `VPN` (class in *smc.core.engine*), 291
- `vpn` (*smc.core.engine.Engine* property), 287
- `vpn` (*smc.core.engine.VPNMapping* property), 294
- `vpn` (*smc.policy.rule\_elements.Action* property), 469
- `vpn_broker_interface` (*smc.core.engine.Engine* property), 287
- `vpn_client` (*smc.core.engine.VPN* property), 293
- `vpn_endpoint` (*smc.core.engine.Engine* property), 287
- `vpn_mappings` (*smc.core.engine.Engine* property), 287
- `vpn_site` (*smc.vpn.elements.ExternalGateway* property), 491
- `vpn_site` (*smc.vpn.policy.GatewayTreeNode* property), 499
- `vpn_status_code` (*smc.administration.monitoring\_status* property), 521
- `VPNBrokerConfigFile` (class in *smc.vpn.elements*), 502
- `VPNBrokerDomain` (class in *smc.vpn.elements*), 501
- `VPNBrokerInterface` (class in *smc.core.interfaces*), 349
- `VPNBrokerInterfaceCollection` (class in *smc.core.collection*), 324
- `VPNBYTESRECEIVED` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `VPNBYTESENT` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `VPNEndpointTunnelStatus` (class in *smc.administration.monitoring\_status*), 520
- `VPNGatewayTunnelStatus` (class in *smc.administration.monitoring\_status*), 520
- `VPNID` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `VPNMapping` (class in *smc.core.engine*), 293
- `VPNMappingCollection` (class in *smc.core.engine*), 294
- `VPNProfile` (class in *smc.vpn.elements*), 173, 497
- `VPNSAQuery` (class in *smc\_monitoring.monitors.vpns*), 99
- `VPNSecurityAssoc` (class in *smc\_monitoring.monitors.vpns*), 100
- `VPNSite` (class in *smc.vpn.elements*), 493
- `VPNSRCID` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `VPNSTATISTICS` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `VPNStatus` (class in *smc.administration.monitoring\_status*), 521
- `VPNSTATUS` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `VPNTYPE` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `vulnerability_references` (*smc.elements.situations.InspectionSituation* property), 258
- `vulnerability_refs` (*smc\_monitoring.monitors.alerts.Alert* property), 104
- `VULNERABILITYREFERENCES` (*smc\_monitoring.models.constants.LogField* attribute), 80
- W**
- `wait()` (*smc.administration.tasks.TaskOperationPoller* method), 168
- `wait()` (*smc.core.waiters.NodeWaiter* method), 525
- `web_app` (*smc.elements.servers.ManagementServer* property), 223
- `web_app` (*smc.elements.servers.WebPortalServer* property), 229
- `WebApp` (class in *smc.elements.servers*), 216
- `WebPortalAdminUser` (class in *smc.elements.user*), 119
- `WebPortalServer` (class in *smc.elements.servers*), 228
- `when_created` (*smc.core.resource.History* property), 114
- `wireless_interface` (*smc.core.engine.Engine* property), 288
- `WIRELESSCHANNEL` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `WIRELESSCONNECTIONS` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `WIRELESSMONITORING` (*smc\_monitoring.models.constants.LogField* attribute), 80
- `WIRELESSSECURITY` (*smc\_monitoring.models.constants.LogField* attribute), 81
- `WIRELESSSSID` (*smc\_monitoring.models.constants.LogField* attribute), 81
- `WIRELESSSTATUS` (*smc\_monitoring.models.constants.LogField* attribute), 81
- `within_ipv4_network()` (*smc\_monitoring.models.filters.TranslatedFilter* method), 55
- `within_ipv4_range()` (*smc\_monitoring.models.filters.TranslatedFilter* method), 55
- Z**
- `ZIPEXPORTFILE` (*smc\_monitoring.models.constants.LogField* attribute), 81
- `Zone` (class in *smc.elements.network*), 185

`zone` (*smc.core.interfaces.Interface property*), [331](#)  
`zone_ref` (*smc.core.interfaces.Interface property*), [331](#)  
`ZTNAConnector` (*class in smc.core.addon*), [298](#)