Contents

1 Contents: 3
  1.1 Introduction .............................................. 3
  1.2 Install ..................................................... 3
  1.3 Usage ....................................................... 6
  1.4 Internals .................................................... 12

2 Indices and tables 19

Python Module Index 21

Index 23
SLAM (Super LAN Address Manager) is a tool to automatically manage network addresses allocations. It keeps track of address allocations and can automatically generate configuration for DNS and DHCP server software and various other computer and network management software and frameworks.
1.1 Introduction

SLAM is a tool to manage network addresses allocations.

1.1.1 Features

- simple network address manager
- very modular
- output for different configuration format (DNS, DHCP, Quattor...)

1.1.2 Requirements

- python2 ( 2.6)
- django (python-django 1.4)
- a database supported by django
- gettext

Optionally:

- sphinx (python-sphinx) to compile the documentation
- nose (python-nose) to run the test suite

1.2 Install

This section explain how to properly set-up SLAM to be able to use it.
1.2.1 Install

1. Extract archive or clone the repository

2. Configure and setup the database: you can use one of these methods:
   - Use the quick and easy method: just run:
     ```
     $ ./quick_setup.sh
     ```
   - Configure SLAM options and create the database:
     - Edit the file `src/configuration.py` and fill at least the `Database configuration`
     - run this command to create the database:
     ```
     $ python ./src/manage.py syncb
     ```
     - then compile the translation files by running theses commands:
     ```
     $ cd ./src/webinterface
     $ python ../manage.py compilemessages
     ```

3. Use any of the two interface provided (CLI or Web)

1.2.2 Database configuration

The database can be configured in the file `src/configuration.py`.

Example with sqlite:

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/tmp/slam.db',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

Available databases are `sqlite3`, `postgresql_psycopg2`, `mysql`, `oracle` and more, depending on your Django set-up.

1.2.3 Running the Web server

You should refer to this page to choose the last advised method to run a production Web server for Django: https://docs.djangoproject.com/en/1.4/howto/deployment/.

**Test server**

A small web server for testing purpose only can be run with this command:

```
$ python manage.py runserver <port>
```

where `port` is the port to listen connection on. For example if you launch the server with:
$ python manage.py runserver 8080

You will be able to access the interface at this address: http://127.0.0.1:8080.

Production FastCGI server

One solution to launch a production server is to use the Django’s FastCGI server. This allow to use existing HTTP servers already running on a server. It can be run with:

$ python manage.py runfcgi host=127.0.0.1 port=8080

This will spawn a FastCGI server that listens on 127.0.0.1:8080.

You will then need to find how to proxify the requests to this FastCGI server by looking in your HTTP server’s documentation.

For Nginx, the configuration could be:

```
server {
    listen 80 default_server;
    root /srv/www;
    location / {
        fastcgi_pass 127.0.0.1:8080;
    }
}
```

1.2.4 Other configuration

You can either keep the configuration.py file in src/configuration.py or create a new directory /etc/slam and store it there. The file in /etc/slam has the priority over the one stored locally.

Parameters in configuration.py

The DEBUG variable should be set to False in a production environment.

You can the name and email address of the administrator of this application in the ADMINS list.

The TIME_ZONE and LANGUAGE_CODE variables can be changed to the proper values corresponding to your region.

The SECRET_KEY variable must be set to a random and private value. You can generate a proper one with the following command:

```
echo "`</dev/urandom tr -dc '[[:graph:]]' | head -c50`"
```

The ROOT_DIR variable is the root of the SLAM’s directory, which contains src/.

The RELOAD_SCRIPT is the path to a script that will be launched when you go to the generate page on the Web interface. In this script you can create or overwrite the configuration files with SLAM and restart all the network daemons affected.

/etc/slam/users

This file can be created and must contain one UNIX login per line. A really basic check is made in the CLI interface of SLAM and login that are not in this file cannot do any action.
1.3 Usage

SLAM can be used through its Web interface or with a CLI script. Database is centralised so both interfaces can be used at the same time.

1.3.1 Command-line interface

Usage

```
$ ./slam_cli.py
[-h]
-a {list|create|get|delete|modify|setprop|rmprop|createconf|upconf|log|export}
[-pn POOL_NAME]
[-p POOL]
[-A ADDRESS]
[-H HOST]
[-c CAT]
[-m MAC]
[--alias ALIAS]
[-r]
[-g GEN]
[--default]
[--header HDR]
[--footer FTR]
[-o OUTFILE]
[--checkfile CHECK]
[--timeout TIMEOUT]
[--domain DOMAIN]
[--inventory INVENTORY]
[--serial SERIAL]
[--duration DURATION]
[--lastuse LASTUSE]
[--comment COMMENT]
[--nodns]
[ARG...]
```

-h, --help
Print the help message.

-a, --action ACTION
Select any available action among: list, create, get, delete, modify, setprop, rmprop, createconf, upconf, log or export.

This option is required.

-pn, --pool-name POOL_NAME
The name of the address pool to apply actions onto.

-p POOL, --pool POOL
Address definition of the pool to manage (ex: 10.1.2.0/24, fc00::/7, “addr1,addr2,addr3”).

-A, --address ADDRESS
An address to list, allocate or free.

-H, --host HOST
The name of the host to manage.
-c, --category CAT
    One or more category of pool to create or to assign to a host.

-m, --mac MAC
    The mac address to assign to the host.

--alias ALIAS
    Specify one or more alias for the host name.

-r, --random
    Hosts will get random addresses from the pool.

-g, --generator GEN
    The name of a generator object to retrieve or create.

--header HDR
    Header file path for the configuration files to generate. Its content will be inserted before the generated configuration.

--footer FTR
    Footer file path for the configuration files to generate. Its content will be inserted after the generated configuration.

-o, --output OUTFILE
    Output file path for the configuration files to generate, or - for stdout.

--checkfile CHECK
    A list of file to check for duplicates before generating configuration.

--timeout TIMEOUT
    A timeout in the format used by bind that will be used to generate records in bind files.

--domain DOMAIN
    Specify a domain that will be used for every entry in the generated configuration files.

--inventory INVENTORY
    Specify the inventory number of a host.

--serial SERIAL
    Specify the serial number of a host.

--duration DURATION
    Specify the duration of an address allocation, in days.

--lastuse LASTUSE
    Update the time the address was last used with the new timestamp.

--comment COMMENT
    Add a comment to an address.

--nodns
    Specify that no DNS record will be generated for a given host.

ARG
    Specify arguments for additional information required by specific action or options like generate or setprop.

Examples

List

List all pools in the database:
List all addresses allocated in the pool localnet:

$ ./slam_cli.py -a list

List all addresses allocated to the host pc-42:

$ ./slam_cli.py -a list -H pc-42

List all hosts in the database:

$ ./slam_cli.py -a list -H ""

Show which host the address 192.168.10.3 is allocated to and to which pool it belongs:

$ ./slam_cli.py -a list -A 192.168.10.3

Show information about a generator:

$ ./slam_cli.py -a list -g my_dhcp

List all generators in the database:

$ ./slam_cli.py -a list -g ""

Create

Create a new pool named localnet defined by the IPv4 subnet 10.9.8.0/24:

$ ./slam_cli.py -a create -pn localnet -p 10.9.8.0/24

Create a new host named server4 and assign it a new available address from localnet:

$ ./slam_cli.py -a create -pn localnet -H server4

Mark the two given addresses from localnet as allocated but does not bind them to a host:

$ ./slam_cli.py -a create -pn localnet -A 10.9.8.0 -A 10.9.8.255

Specify the mac address and aliases of the new host:

$ ./slam_cli.py -a create -pn localnet -H server5 -m 00:11:22:33:44:55 --alias_webserver --alias mailserver

Create a pool serv of category server and network:

$ ./slam_cli.py -a create -pn serv -p 1.2.3.0/24 -c server, network

Create a new host and assign it to the pool corresponding to the category server (it will take an address from the serv pool):

$ ./slam_cli.py -a create -H server64 -c server

Create a new host which addresses won’t be generated for DNS configuration files:
### Get

Ask for a new address from `localnet` pool for the host `pc42`:

```
$ ./slam_cli.py -a create -H nat -c network --nodns
```

Ask for a new address random from `localnet` pool for the host `pc42`:

```
$ ./slam_cli.py -a get -pn localnet -H pc42
```

Try to assign address `10.9.8.123` form pool to the host `pc42`:

```
$ ./slam_cli.py -a get -pn localnet -H pc42 -A 10.9.8.123
```

### Delete

Delete `localnet` and all the addresses it contained:

```
$ ./slam_cli.py -a delete -pn localnet
```

Delete `server4` and mark all the addresses it had as unallocated:

```
$ ./slam_cli.py -a delete -H server4
```

Unallocate `10.9.8.7`:

```
$ ./slam_cli.py -a delete -A 10.9.8.7
```

### Modify

Modify the name of a pool:

```
$ ./slam_cli.py -a modify -pn localnet3 localnet30
```

Modify the name of a host, from “`pc-1337`” to “`server-1337`”:

```
$ ./slam_cli.py -a modify -H pc-1337 server-1337
```

Modify the mac address of a host:

```
```

Modify the category of a pool:

```
$ ./slam_cli.py -a modify -pn serv -c servercategory
```

Add a comment to an address:

```
$ ./slam_cli.py -a modify -A 10.9.8.6 --comment "Special address."
```
Generate

You can generate configuration entries in format for Bind, Quattor or DHCP, simply specify `bind`, `revbind`, `quattor` or `dhcp` after the generate.

Generate a configuration file for all hosts in the database:

```
$ ./slam_cli.py -a createconf -o out.conf bind
$ ./slam_cli.py -a createconf -o out.conf revbind
$ ./slam_cli.py -a createconf -o out.conf quattor
$ ./slam_cli.py -a createconf -o out.conf dhcp
```

Add additional parameters like timeout or domain:

```
$ ./slam_cli.py -a createconf --timeout 3H -o out.conf bind
$ ./slam_cli.py -a createconf --domain lan.example -o out.conf dhcp
```

Generate a DHCP configuration file for all hosts that have an address in `localnet`:

```
$ ./slam_cli.py -a createconf -pn localnet --domain lan.example -o out.conf dhcp
```

Add content before (`-header`) or after (`-footer`) the generated content:

```
$ ./slam_cli.py -a createconf -pn localnet -o out.conf --header header.zonefile --footer footer.zonefile bind
```

Write the configuration output to stdout:

```
$ ./slam_cli.py -a createconf -o - bind
```

Update an existing configuration file generated by SLAM:

```
$ ./slam_cli.py -a upconf -o ./out.conf bind
```

Check for existing records in `conf.old1` and `conf.old2` before the generation of the configuration:

```
$ ./slam_cli.py -a createconf bind -o --checkfile conf.old1 --checkfile conf.old2
```

A `dhcpgen` generator object can be created this way:

```
$ ./slam_cli.py -a create -g dhcpgen -o /etc/dhcpd/hosts.conf dhcp
```

It is possible to modify it:

```
$ ./slam_cli.py -a modify -g dhcpgen --header /etc/dhcpd/header.conf
```

And then to use it instead of specifying every options:

```
$ ./slam_cli.py -a createconf -g dhcpgen
$ ./slam_cli.py -a updateconf -g dhcpgen
```

When creating a new generator it is possible to set as `default` for the configuration type:

```
$ ./slam_cli.py -a create -g dnsgen dns -o /etc/bind/db.foo.example --default
```

Then, whenever you run a `createconf` or `updateconf` without a generator name or other options, it will run the generators that were set as `default`:
The second line will only run the default generators that generate DNS configuration files.

**Properties**

Add a property to a pool object:

```
$ ./slam_cli.py -a setprop -pn localnet building=200
```

Add a property to a host object:

```
$ ./slam_cli.py -a setprop -H pc-1337 mac=00:12:34:56:78:9a
```

Change the property of an object:

```
```

Delete a given property of an object:

```
$ ./slam_cli.py -a rmprop -H pc-1337 mac
```

List all pool and host that have a `building` property:

```
$ ./slam_cli.py -a list building
```

List all pool and host that have a `building` property of value 200:

```
$ ./slam_cli.py -a list building=200
```

**Alias**

Add one or more alias:

```
$ ./slam_cli.py -a modify -H myhost --alias alias1,alias2,alias3
```

or:

```
$ ./slam_cli.py -a modify -H myhost --alias alias1 --alias alias2 --alias alias3
```

To remove an alias from a host, prefix it with %:

```
$ ./slam_cli.py -a modify -H myhost --alias %alias2
```

To clear all alias of a host, pass it one empty alias:

```
$ ./slam_cli.py -a modify -H myhost --alias ""
```

**Other**

Every action is logged in the database. It is possible to access log entries with:

1.3. Usage
$ ./slam_cli.py -a log

A back-up can be done with the export action which will output all the SLAM’s commands to regenerate the full database:

$ ./slam_cli.py -a export > restore.sh

### 1.3.2 Web interface

#### Running a server

To run a test web server on port port, you can run this command:

```
$ ./src/manage.py runserver <port>
```

To run a production FastCGI server, run this command:

```
$ ./src/manage.py runfcgi
```

Look at the Django documentation for more information about the options of these commands.

### 1.3.3 Test-suite

To run the test-suite you need python-nose.

Then, from the root directory of SLAM, you can run this command:

```
nosetests
```

You can also see the test-suite code coverage if you have nose-cov installed:

```
nosetests --with-cov --cov-report term-missing --cov src/
```

### 1.3.4 Documentation

To generate the documentation, you need python-sphinx, then, from the doc directory you can run this command to see all the available output formats:

```
make help
```

And choose the one you prefer:

```
make html
```

### 1.4 Internals

#### 1.4.1 Technologies

This is a summary of the technologies SLAM uses and what they are used for:
• SLAM, the actual tool
  – **python** (2.6) as the main programming language
  – **Django** (*python-django*): framework used as Web interface and as an ORM for the application
  – **SQLite** as the database backend, can be changed by users by only changing one configuration variable
• SLAM’s versioning system
  – **Git**: distributed versioning system
• SLAM’s installation
  – custom **shell** script: automatic configuration of the project
  – **python** script provided by Django: initialize the database, start the Web server...
• SLAM’s testing and integration, used for code-style, unit, functional testing
  – **pylint**: very customizable coding-style and programming mistakes checker
  – **NOSE** (*python-nose*): simple and complete test framework for Python applications
  – **NOSE’s coverage plugin** (*nose-cov*): used to get a complete and precise report on the test-suite code coverage
  – **Jenkins**: continuous integration software that poll the repository and automatically launch the full style and test-suite each day or for any change in the repository
• SLAM’s documentation and project management
  – **SPHINX** (*python-sphinx*): generate the documentation in multiple formats (HTML, PDF, epub, texinfo...) from *reStructuredText* files
  – **Trac**: the Wiki and the bug tracker are the main tools we use

### 1.4.2 Global Architecture

#### Core

The core of the project is composed of models: Host, Pool, Address, Property that are the structures used by Django to represent data in the database; the address ranges and collections: Ip4Range for IPv4, Ip6Range for IPv6 and AddressSet for arbitrary addresses; and generators that output configuration file for a specific software or service.

Most of the core processing happens in the address ranges and in the **Pool** class. Address ranges store a collection of address and **Pool** keeps track of the status of the address of one of these collection (allocated or available).

Properties can be added to a **Host** or a **Pool** and can be used to add any information of type (key: value).

#### Interfaces

Two interfaces are provided by default: CLI and Web with Django. They interact directly with the SLAM Django application through the **slam.interface** module. This module provides several helper function to perform various usual task. New interfaces should use these functions as much as possible but can very well interact directly with the SLAM Django application.
REST and Web interfaces

It was chosen to merge these two interfaces because a lot of code would have been duplicated between these two interfaces if they were split. Furthermore, thanks to Django’s template system and its REST middleware, the distinction is really no that strong. It also allow to only test the REST interface with the JSON output and consider the Web interface is tested as well because the treatments are exactly the same and parsing the JSON output is far easier than HTML.

For user input, the problem was that HTML does not allow the generation of query with PUT and DELETE method which are required to provide a sane REST interface (it was added to the HTML5 specification at some point but has been removed since). However a really simple Django middleware can be used to transparently make the conversion between POST request with hidden fields and HTTP methods which allow us to receive only proper REST query without any effort.

For output, a simple in-url setting (format) allow the user to specify that he expect a specific output format such as json. This is implemented really easily thanks to Django’s template system and the only difference between the HTML and JSON format is the template used.

Interface Translation

We chose to only translate the Web interface. This is done easily with the internationalization module of Django which generate automatically .po files for easy translation.

Generators

Generators inherit from Config in slam.generator and can overwrite gen_header, gen_footer and generate to customize the behavior of the generator. For example, the Bind generator uses gen_header to parse the SOA record and increment it. They can also declare their format of comment that will be used to generate the SLAM headers in the generated configuration file.

Tests

Tests are stored in their own files in the /test directory, regrouped by tested python modules. They are executed on an independant database. Tests that need to compare output use StringIO and test that need to provide an input stream uses a simple pipe.

1.4.3 Evolution

New category model

A new category model is in progress on the branch new_category_model. The idea is to remove the concept of category and to replace it by HostType and AddrType.

The aimed model is that a pool can be linked to several AddrType which defines the way addresses will be generated (it embeds several attributes such as the way aliases are handled, DNS TTL, DNS zone, etc.). Each Host has a HostType which defines its category (pc, laptop, server...). The HostType has a list of AddrTypes that will be automatically allocated at the host’s creation and which is represented by a custom relation (HostTypeAddrType). This relation allow to specify that a given HostType will have automatically at its creation.

HostType:

• name,
• name_format: would be a special formatted string to allow a host name to be automatically generated from their IP, category, etc.

AddrType:

• name,
• dns_zone: the domain of which this host is a sub-domain,
• dns_alias: describes how the alias are treated when dns records are generated. Aliases could be ignore, or generated as CNAME, or as alternative A records,
• dhcp_alias: describes how the alias are treated when dhcp records are generated.
• dns_timeout: the TTL for the generated DNS records.

HostTypeAddrType:

• hosttype,
• addrtype: AddrType to automatically allocate when the host is created,
• count: number of this AddrType to allocate to the host.

Still todo on the new_category_model branch

The models are finished, common interface to add and modify HostType and AddrType is done in slam/interface.py. It needs to be integrated in the CLI interface and the web interface.

The three interfaces (common, CLI and Web) need to be modified to integrate the new types, and to allocate automatically addresses related to the types, write a new interface to list objects from the two new classes, and so on. Generators also need to be heavily modified to integrate this new model and generate different things depending on the new attributes.

All the documentation and tests need to be written.

Interface with the DHCP server

The CLI interface has the option –lastuse, it can be used to record the last query seen by the DHCP server for a particular host and address. An interface could easily made by parsing the DHCP log files to track the unused addresses.

Improvements to the CLI interface

The whole CLI interface is quite terrible, with to much options and a lot of implicit behaviors. It should be remade from scratch.

1.4.4 Internal structures

This page describe the internal structures of SLAM.

Models

Models are used to store information persistently in the database.
Pool class

Note: the reason why we use `random.randint` instead of `random.choice` is because `random.choice` uses `len(range)`, see `addrrange` module.

Host class

Alias Class

Note: relation between Alias and Host is n,m because we want, by design, that several hosts have the same alias and also that several aliases point to the same host.

Address class

Property class

Address collections

These class represent a collection of address: it can be either an entire subnet or set of manually defined addresses.

addrrange module

Note: we expect these classes to provide a `.len()` method instead of a `__len__()` because `__len__` expect an integer in python2 which triggers an exception when we call `len` on an `Ip6Range` which often return a long.

Contains classes that represent a collection of addresses.

```python
class slam.addrrange.AddrSet (addr_set=None, dns_record='A')
    Represent a generic set of addresses. The addresses must be added manually.
    __init__ (addr_set=None, dns_record='A')
        Initialize a new address set of DNS record type `dns_record`.
    add (addr)
        Add the address `addr` to the set.
    len ()
        Return the number of elements in the set.
    remove (addr)
        Remove the address `addr` from the set.
    sortable (addr)
        Returns a sortable representation of an address from this range.

exception slam.addrrange.InvalidAddressError
    The format of the given address was not recognized.
    __weakref__
        list of weak references to the object (if defined)
```

class slam.addrrange.Ip4Range (iprange)
    Represents an IPv4 subnet.
__contains__(addr)
    Return true if the given addr belongs to this subnet.

__getitem__(ind)
    Return the formatted addresses at index n from the subnet.

__init__(iprange)
    Initialize the range with the given subnet iprange with format x.x.x.x/x.

__iter__()
    Iterator over the IPv4 subnet.

__len__()
    Return the number of addresses in this subnet.

__str__()
    Return a human-readable representation of the subnet.

len()
    Return the number of addresses in this subnet.

sortable(addr)
    Returns a sortable representation of an address from this range.

class slam.addrrange.IP6Range(iprange)
    Represents an IPv6 subnet.

__contains__(addr)
    Return true if the given addr belongs to this subnet.

__getitem__(ind)
    Return the formatted addresses at index n from the subnet.

__init__(iprange)
    Initialize the range with the given subnet iprange with format 12ab:34cd::89ef/x.

__iter__()
    Iterator over the IPv6 subnet.

__len__()
    Return the number of addresses in this subnet.

__str__()
    Return a human-readable representation of the subnet.

len()
    Return the number of addresses in this subnet.

sortable(addr)
    Returns a sortable representation of an address from this range.

**Configuration Generators**

These class are used to generate configuration files for common network services such as DNS, DHCP or Quattor.

generator module

Interfaces
Generic Interface

slam_cli module

Web Interface

Tests

Note that a few functionality are not very well tested because they are just a straight adaptation of the Python built-in structures such as the AddressSet which is just a wrapper around python’s set.

test_range module

test_pool module

test_cli module
CHAPTER 2

Indices and tables

- genindex
- modindex
- search
S

slam.addrange, 16
Symbols

-alias ALIAS
  slam-cli command line option, 7
-checkfile CHECK
  slam-cli command line option, 7
-comment COMMENT
  slam-cli command line option, 7
-domain DOMAIN
  slam-cli command line option, 7
-duration DURATION
  slam-cli command line option, 7
-footer FTR
  slam-cli command line option, 7
-header HDR
  slam-cli command line option, 7
-inventory INVENTORY
  slam-cli command line option, 7
-lastuse LASTUSE
  slam-cli command line option, 7
-nodns
  slam-cli command line option, 7
-seqal SERIAL
  slam-cli command line option, 7
-timeout TIMEOUT
  slam-cli command line option, 7
-A, -address ADDRESS
  slam-cli command line option, 6
-H, -host HOST
  slam-cli command line option, 6
-a, -action ACTION
  slam-cli command line option, 6
-c, -category CAT
  slam-cli command line option, 6
-g, -generator GEN
  slam-cli command line option, 7
-h, -help
  slam-cli command line option, 6
-m, -mac MAC
  slam-cli command line option, 7
-o, -output OUTFILE
  slam-cli command line option, 7
-p POOL, -pool POOL
  slam-cli command line option, 6
-pn, -pool-name POOL_NAME
  slam-cli command line option, 6
-r, -random
  slam-cli command line option, 7
__contains__() (slam.addrrange.Ip4Range method), 16
__contains__() (slam.addrrange.Ip6Range method), 17
__getitem__() (slam.addrrange.Ip4Range method), 17
__getitem__() (slam.addrrange.Ip6Range method), 17
__init__() (slam.addrrange.AddrSet method), 16
__init__() (slam.addrrange.Ip4Range method), 17
__init__() (slam.addrrange.Ip6Range method), 17
__iter__() (slam.addrrange.Ip4Range method), 17
__len__() (slam.addrrange.Ip4Range method), 17
__len__() (slam.addrrange.Ip6Range method), 17
__str__() (slam.addrrange.Ip4Range method), 17
__str__() (slam.addrrange.Ip6Range method), 17
__weakref__ (slam.addrrange.InvalidAddressError attribute), 16

A
add() (slam.addrrange.AddrSet method), 16
AddrSet (class in slam.addrrange), 16
ARG
  slam-cli command line option, 7

I
InvalidAddressError, 16
Ip4Range (class in slam.addrrange), 16
Ip6Range (class in slam.addrrange), 17
L
len() (slam.addrrange.AddrSet method), 16
len() (slam.addrrange.Ip4Range method), 17
len() (slam.addrrange.Ip6Range method), 17

R
remove() (slam.addrrange.AddrSet method), 16

S
slam-cli command line option
-alias ALIAS, 7
-checkfile CHECK, 7
-comment COMMENT, 7
-domain DOMAIN, 7
-duration DURATION, 7
-footer FTR, 7
-header HDR, 7
-inventory INVENTORY, 7
-lastuse LASTUSE, 7
-nodns, 7
-serial SERIAL, 7
-timeout TIMEOUT, 7
-A, -address ADDRESS, 6
-H, -host HOST, 6
-a, -action ACTION, 6
-c, -category CAT, 6
-g, -generator GEN, 7
-h, -help, 6
-m, -mac MAC, 7
-o, -output OUTFILE, 7
-p POOL, -pool POOL, 6
-pn, -pool-name POOL_NAME, 6
-r, -random, 7
ARG, 7
slam.addrrange (module), 16
sortable() (slam.addrrange.AddrSet method), 16
sortable() (slam.addrrange.Ip4Range method), 17
sortable() (slam.addrrange.Ip6Range method), 17