# geni-lib Documentation

***Release 0.9.7.9***

**Nick Bastin**

**May 07, 2018**

# Contents


**1 Introduction** **2**

**2 Credentials** **3**
- 2.1 Getting Credentials from the NSF GENI Portal . . . 4
- 2.2 Getting Credentials from the CloudLab Portal . . . 5

**3 Installation** **6**
- 3.1 Ubuntu 14.04 . . . 7
- 3.2 Ubuntu 16.04 . . . 8
- 3.3 MacOS X 10.10.x / 10.11.x . . . 9
- 3.4 Vagrant . . . 10
- 3.5 CentOS 7 . . . 12

**4 Tutorials / How-Tos** **13**
- 4.1 Importing a Context from a bundle . . . 13
- 4.2 Creating a Context from Cloudlab Credentials . . . 15
- 4.3 Creating a Custom Context . . . 16
- 4.4 Querying the Federation . . . 18
- 4.5 Creating a Request for a Single VM . . . 22
- 4.6 VTS: Basic Single-Site Topology . . . 24
- 4.7 VTS: Basic WAN Topology . . . 27

**5 API** **31**
- 5.1 geni.aggregate . . . 31
- 5.2 geni.minigcf.config . . . 45
- 5.3 geni.portal . . . 45
- 5.4 geni.rspec . . . 48
- 5.5 geni.types . . . 58
- 5.6 geni.urn . . . 59
- 5.7 geni.util . . . 61

**6 Development** **63**
- 6.1 Supported Use Cases . . . 64
- 6.2 Coding Conventions . . . 65
- 6.3 Pattern Conventions . . . 65
- 6.4 Philosophy Notes . . . 65
- 6.5 Things That Don't Belong . . . 66

7 Indices and tables 67

Python Module Index 68

Contents:

CHAPTER 1

# Introduction

geni-lib is a Python library for interacting with the NSF GENI Federation, or any federation that uses components of the GENI Software Architecture.

Common uses include orchestrating repeatable experiments and writing small tools for inspecting the resources available in a given federation. There are also a number of administrative API handlers available for interacting with software commonly used in experiments - particularly those exposing services to other experimenters.

CHAPTER 2

# Credentials

Using *geni-lib* to interact with cloud / testbed resources requires credentials for your account with a given provider (GENI, CloudLab, Emulab, etc.). The following guides provide documentation for how to acquire the credentials that *geni-lib* needs for each environment.

**Note:** If you only want to use geni-lib to manipulate XML resource files (RSpecs) to use with another tool, you do not need to install any credentials

Guides:

## 2.1 Getting Credentials from the NSF GENI Portal

### 2.1.1 Download The omni.bundle

You need a file called `omni.bundle` which is available from the GENI Portal web interface. Once you log into the GENI Portal you can use the following steps to locate your `omni.bundle` download:

- At the top of the Portal home page click on the tab labeled **Profile**
- In the tabs on the Profile page click on the one labeled **Configure omni**
- Embedded in the text under the **Option 1: Automatic omni configuration** header, there is a button labeled **Download your omni data**. Click this button.

---

**Note:** If you see a warning that no SSH keys have been uploaded you can still use the bundle, but you will need to specify an SSH public key path later if you want to use resources that require SSH login

---

- Click the **Download your omni data** button at the bottom of the next page and it should start downloading immediately in your browser.

## 2.2 Getting Credentials from the CloudLab Portal

CloudLab only provides your x509 certificate from the web interface. You will have to provide your own SSH public key for use with *geni-lib* when you set up your context for using reserved resources.

- Log in to the CloudLab Portal.
- From the **<your-username>** dropdown at the top right of the interface, select the **Download Credentials** option
- Save this text (either via copy/paste or **Save As...** in your brower) to a file called `cloudlab.pem` for use when creating your context.

You will also need to take note of the projects of which you are a member, in order to set up your context. You can view your projects by clicking on the **Membership** tab on the user dashboard interface.

CHAPTER 3

# Installation

## 3.1 Ubuntu 14.04

Release versions of geni-lib are delivered via PyPI, but some system dependencies must be supplied, typically through the use of *apt*.

### 3.1.1 High-Level Dependencies

- Python 2.7.x (http://www.python.org)
- Pip
- OpenSSL
- LibXML

The above packages of course have their own dependencies which will be satisfied along the way.

**Warning:** The version of *pip* supplied via apt packages for Ubuntu 14.04 for Python 2.x is sufficiently broken that it is unlikely to be able to install *geni-lib* (or many other packages). The instructions below install *pip* outside of the package management system, using the most up-to-date installer. If you are already using *virtualenv* or otherwise maintain a sane Python environment you likely do not need to install a new *pip*.

### 3.1.2 Install Dependencies

```
$ sudo apt-get update
$ sudo apt-get install --no-install-recommends python libxml2 libssl1.0.0

$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python2 get-pip.py
```

### 3.1.3 Install

```
$ pip install geni-lib
```

**Note:** You may need to install with *sudo* if you are attempting to install system-wide.

## 3.2 Ubuntu 16.04

Release versions of geni-lib are delivered via PyPI, but some system dependencies must be supplied, typically through the use of *apt*.

### 3.2.1 High-Level Dependencies

- Python 2.7.x (http://www.python.org)
- Pip
- OpenSSL
- LibXML

The above packages of course have their own dependencies which will be satisfied along the way.

**Warning:** The version of *pip* supplied via apt packages for Ubuntu 16.04 for Python 2.x is sufficiently broken that it is unlikely to be able to install *geni-lib* (or many other packages). The instructions below install *pip* outside of the package management system, using the most up-to-date installer. If you are already using *virtualenv* or otherwise maintain a sane Python environment you likely do not need to install a new *pip*.

### 3.2.2 Install Dependencies

```
$ sudo apt-get update
$ sudo apt-get install --no-install-recommends python2.7 libxml2 libssl1.0.0

$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python2.7 get-pip.py
```

### 3.2.3 Install

```
$ pip install geni-lib
```

**Note:** You may need to install with *sudo* if you are attempting to install system-wide.

## 3.3 MacOS X 10.10.x / 10.11.x

These installations require the use of HomeBrew (http://brew.sh). If you use MacPorts or a different manager for installing open source tools on your system you will need to satisfy the dependencies using your tool of choice.

**Note:** These instructions have not been tested on older versions of MacOS X

### 3.3.1 Installation Dependencies

- HomeBrew (http://brew.sh)
- Apple Command Line Tools for XCode (normally downloaded as part of Brew install)

### 3.3.2 Install / Setup

Using HomeBrew (accessible via the `brew` command in your terminal, once you have it installed) we will install the necessary tools and library dependencies for typical `geni-lib` use:

```
$ brew install mercurial
$ brew install python
```

**Note:** You will now have two version of `python` installed on your system - the one that Apple ships with your computer, and the one that we have now installed via `brew`. Only the `brew`-installed `python` will work for running `geni-lib` scripts, which can be launched by running `/usr/local/bin/python` or by changing your `$PATH` variable to have `/usr/local/bin` as the first entry (by editing `~/.profile`, typically).

### 3.3.3 Get geni-lib

You can place the geni-lib repository anywhere on your system that you prefer.

```
$ hg clone http://bitbucket.org/barnstorm/geni-lib
```

### 3.3.4 Install geni-lib

We can now install `geni-lib` into your Python environment:

```
$ cd geni-lib
$ hg update -C 0.9-DEV
$ python setup.py install
```

Congratulations, you are now ready to launch `python` and import geni lib modules!

## 3.4 Vagrant

`geni-lib` can be installed on any platform that supports Vagrant using the instructions below.

Two variants of the Vagrant environment can be created - a *lite* version that only contains `geni-lib` and should only be used by developers that are accustomed to working with Vagrant or similar environments, and a *lab* version that is more fully-featured and should be used by new `geni-lib` users or those using `geni-lib` for a class or conference tutorial.

The documentation here currently only covers the *lab* version.

The Vagrant VM created by this process automatically sets up your geni-lib context and provides a web interface for creating Jupyter notebooks using GENI resources, as well as a web-based interface for accessing the VM shell.

---

**Note:** See the Configurable Options section below for environment variables which can tweak the settings of the VM environment that is created.

---

### 3.4.1 Installation Dependencies

Install these dependencies before creating the Vagrant VM.

- VirtualBox (https://www.virtualbox.org/wiki/Downloads)
- Vagrant (https://www.vagrantup.com/downloads.html)

### 3.4.2 Set up your geni-lib VM directory

- Create a directory on your system named `genivm` to hold your GENI environment
- Copy your `omni.bundle` to this directory
- Download the `geni-lib` Vagrant setup file to this directory from https://bitbucket.org/barnstorm/geni-lib/raw/tip/support/Vagrantfile-lab and rename it to be called `Vagrantfile`
- On systems with `curl` (MacOS X, Linux) you can use the following command:

```
curl https://bitbucket.org/barnstorm/geni-lib/raw/tip/support/Vagrantfile-lab -o␣
↪Vagrantfile
```

- On Windows systems with Powershell you can use the following:

```
PS C:\genivm> $client = new-object System.Net.WebClient
PS C:\genivm> $client.DownloadFile("https://bitbucket.org/barnstorm/geni-lib/raw/
↪tip/support/Vagrantfile-lab", "C:/genivm/Vagrantfile")
```

  **Note:** The full path for the destination *must* be specified in the second argument to *DownloadFile*

- Create your vagrant vm using `vagrant up` in this directory

---

**Note:** This may take a long time (20+ minutes) depending on the speed of your internet connection

---

### 3.4.3 Load the Jupyter web interface

- Open any web browser and load `http://localhost:8900`
- In the upper right-hand corner of the UI, choose `New->(Notebooks) Python 2` from the dropdown menu
- In the new notebook enter `%load_ext genish` in the first cell and enter your key passphrase if necessary (otherwise just hit enter to skip the passphrase entry)

### 3.4.4 Accessing the VM Terminal

You may often want to access the VM command line for accessing your GENI resources, updating `geni-lib`, etc. While you can use *vagrant ssh* on some platforms, this doesn't work very well on Windows, so the VM provides a web-based mechanism for accessing the VM shell directly.

- Open any web browser and load `http://localhost:8900`
- In the upper right-hand corner of the UI, choose `New->Terminal` from the dropdown menu

This will automatically log you into the VM and provide you a shell interface for using the VM OS directly.

### 3.4.5 Configurable Options

The following environment variables can be set to change the parameters under which the VM is created when *vagrant up* is first executed:

| Name | Default | Description |
|---|---|---|
| `glv_port` | 8900 | Local port the Jupyter web interface will be exposed on |
| `glv_ram` | 1024 | Amount of memory available to the VM |
| `apt_cache` | *unset* | URL of proxy used for apt downloads |
| `pypi_test` | *unset* | If set, use the test PyPI repository instead of production |

## 3.5 CentOS 7

geni-lib is currently delivered only as a source repository via mercurial, although dependencies are installed as proper packages using yum.

### 3.5.1 High-Level Dependencies

- Mercurial (http://mercurial.selenic.com)
- Python 2.7.x (http://www.python.org)
- OpenSSL
- LibXML

The above packages of course have their own dependencies which will be satisfied along the way.

### 3.5.2 Install Dependencies

These instructions install dependencies using yum - it is also possible to install the Python packages using pip if you prefer.

The dependencies rely on EPEL (https://fedoraproject.org/wiki/EPEL), so install that first.

```
$ yum install epel-release
```

Now install the dependencies:

```
$ yum install mercurial python-lxml python-requests \
  python-pip python-devel libffi-devel gcc openssl-devel
```

### 3.5.3 Get geni-lib

```
$ hg clone http://bitbucket.org/barnstorm/geni-lib
```

### 3.5.4 Install

```
$ cd geni-lib
$ hg update -C 0.9-DEV
$ pip install .
```

CHAPTER 4

# Tutorials / How-Tos

## 4.1 Importing a Context from a bundle

In order to communicate with any federation resource using `geni-lib` you need to construct a `Context` object that contains information about the framework you are using (for example ProtoGENI, Emulab, GENI Clearinghouse, etc.), as well as your user information (SSH keys, login username, federation urn, etc.). This simple tutorial will walk you through the easiest way to create a `Context` if you have an account at the GENI Portal.

First you need to *acquire your GENI credentials*.

### 4.1.1 Run Context Import Tool

A script called `context-from-bundle` was installed as part of your `geni-lib` installation, which can convert your `omni.bundle` into the data necessary for `geni-lib` to create a `Context` object for you. The instructions for using this tool are below - choose the section appropriate for your OS.

#### MacOS X / Linux

In most installations your path should already include the import tool and it should run cleanly without any additional configuration:

```
$ context-from-bundle --bundle /path/to/omni.bundle
```

If no arguments are supplied the bundle is assumed to be in the current directory. If your bundle does not contain an SSH public key you will be required to supply a path to one using the `--pubkey` argument at the command line.

#### Windows

Unfortunately the default Python installation on Windows does not add the site Scripts directory to your path, so you need to invoke it directly. If you are using Python 2.8 you will need to replace `Python27` with `Python28` below:

```
C:\> python C:\Python27\Scripts\context-from-bundle --bundle path\to\omni.bundle
```

If no arguments are supplied the bundle is assumed to be in the current directory. If your bundle does not contain an SSH public key you will be required to supply a path to one using the `--pubkey` argument at the command line.

### 4.1.2 Test It Out!

Now we can take your newly imported information, instantiate our context, and query an aggregate:

```
$ python
>>> import geni.util
>>> context = geni.util.loadContext()
>>> import geni.aggregate.instageni as IG
>>> import pprint
>>> pprint.pprint(IG.GPO.getversion(context))
{'code': {'am_code': 0,
          'am_type': 'protogeni',
          'geni_code': 0,
          'protogeni_error_log': 'urn:publicid:IDN+instageni.gpolab.bbn.
↪com+log+abedbcc20e6defe716eb83b8586c7e08',
          'protogeni_error_url': 'https://boss.instageni.gpolab.bbn.com/spewlogfile.
↪php3?logfile=abedbcc20e6defe716eb83b8586c7e08'},
...snip...
```

You should get a large structure of formatted output telling you version and configuration information about the GPO InstaGENI aggregate. If you get any errors read them thorougly and review what they may be telling you about any mistakes you may have made. You can also ask your instructor if at an in-person tutorial.

### 4.1.3 Finished!

Assuming you have experienced no errors, your `geni-lib` installation is now set up and can communicate with all aggregates in the federation. If you have any issues you can send a message to the geni-users google group for help.

## 4.2 Creating a Context from Cloudlab Credentials

You can use the generic *build-context* tool included with *geni-lib* to build a context definition for use with Cloudlab. You will need the following files before you start:

- A Cloudlab x509 credential in .pem format
- The name of a project of which you are a member
- An ssh public key

Given the above, you can run the *build-context* tool directly:

```
build-context --type cloudlab --cert /path/to/cloudlab.pem \
  --pubkey /path/to/ssh_key.pub --project projectname
```

Replacing the paths and project name with values appropriate for your environment. This will create a context definition which will be loaded automatically when using the *genish* or *ipython* interfaces, and can be loaded into your custom scripts using *geni.util.loadContext()*.

## 4.3 Creating a Custom Context

In order to communicate with any federation resource using `geni-lib` you need to construct a `Context` object that contains information about the framework you are using (for example ProtoGENI, Emulab, GENI Clearinghouse, etc.), as well as your user information (SSH keys, login username, federation urn, etc.). You can use the `context-from-bundle` script that comes with `geni-lib` to create a context from an `omni.bundle` provided by the GENI Portal as documented in the "Importing a Context from the GENI Portal" tutorial, or you can create one using a small Python module which allows for more configurability, and we illustrate that method here.

- To start, we will create a new Python file called `mycontext.py` and (inside the directory containing your `geni-lib` clone) import the necessary modules to start building your own context using your favorite editor:

```
from geni.aggregate import FrameworkRegistry
from geni.aggregate.context import Context
from geni.aggregate.user import User
```

- Now we add a function that you will call each time you want to create your context (using the GENI Clearinghouse as the default framework):

```
def buildContext ():
  framework = FrameworkRegistry.get("portal")()
```

- You need to give the framework instance the location of your user certificate and key files:

```
framework.cert = "/home/user/.ssh/portal-user.pem"
framework.key = "/home/user/.ssh/portal-user.key"
```

---

**Note:** You may only have one file which contains both items - you can use the same path for both variables if this is the case.

---

- Now we need to define an account and SSH key(s) that will be used to access reserved compute resources:

```
user = User()
user.name = "myusername"
user.urn = "urn:publicid:IDN+ch.geni.net+user+myusername"
user.addKey("/home/user/.ssh/geni_dsa.pub")
```

  We create a `User()` object, give it a name (no spaces, commonly a username), and the user URN. We then add an SSH public key that will be installed on any compute resources that you reserve in order to authenticate with those resources.

- Next we make the parent `Context` object and add our user and framework to it, with a default project:

```
context = Context()
context.addUser(user, default = True)
context.cf = framework
context.project = "GEC21"
```

  This adds the user we created above, sets the control framework (`cf`), and sets your default project.

- You now want to return this object so that you can use this function every time you need a context:

```
return context
```

Now to see the complete code in one block:

```
from geni.aggregate import FrameworkRegistry
from geni.aggregate.context import Context
from geni.aggregate.user import User

def buildContext ():
  framework = FrameworkRegistry.get("portal")()
  framework.cert = "/home/user/.ssh/portal-user.pem"
  framework.key = "/home/user/.ssh/portal-user.key"

  user = User()
  user.name = "myusername"
  user.urn = "urn:publicid:IDN+ch.geni.net+user+myusername"
  user.addKey("/home/user/.ssh/geni_dsa.pub")

  context = Context()
  context.addUser(user, default = True)
  context.cf = framework
  context.project = "GEC21"

  return context
```

You can dynamically alter this object at any time, but your defaults will serve your purposes for the vast majority of your use cases.

### 4.3.1 Test It Out!

Now we can take your newly written file, instantiate our context, and query an aggregate:

```
$ python
>>> import mycontext
>>> context = mycontext.buildContext()
>>> import geni.aggregate.instageni as IG
>>> import pprint
>>> pprint.pprint(IG.GPO.getversion(context))
{'code': {'am_code': 0,
          'am_type': 'protogeni',
          'geni_code': 0,
          'protogeni_error_log': 'urn:publicid:IDN+instageni.gpolab.bbn.
↪com+log+abedbcc20e6defe716eb83b8586c7e08',
          'protogeni_error_url': 'https://boss.instageni.gpolab.bbn.com/spewlogfile.
↪php3?logfile=abedbcc20e6defe716eb83b8586c7e08'},
...snip...
```

You should get a large structure of formatted output telling you version and configuration information about the GPO InstaGENI aggregate. If you get any errors read them thorougly and review what they may be telling you about any mistakes you may have made. You can also ask your instructor (if at a GEC / Live Tutorial), or send a message to the geni-users google group.

## 4.4 Querying the Federation

Before we can reserve resources, it is useful to know what resources are available across the federation. This tutorial will walk you through using the `Context` object you created in the previous tutorial to communicate with aggregates known to `geni-lib`.

### 4.4.1 Finding Aggregate Locations

`geni-lib` contains a set of package files which have pre-built objects representing known aggregates that are ready for you to use, contained within the following Python modules:

```
geni.aggregate.exogeni
geni.aggregate.instageni
geni.aggregate.instageni_openflow
geni.aggregate.opengeni
geni.aggregate.protogeni
geni.aggregate.vts
```

While these aggregates objects will likely cover your needs, `geni-lib` may of course not be updated as frequently as new aggregates come online. You can find a list of the current set of aggregates on the GENI Wiki.

### 4.4.2 Getting Aggregate Information

Given that we have our previously created `Context` object, and a wealth of aggregate objects available to us, the GENI federation provides the ability to request two blocks of information from each aggregate - the version information (which you may have seen briefly in a previous tutorial), and a list of the advertised resources.

The result from `getversion`, as we saw in the previous tutorial, is reasonably concise and human readable (but also contains information about API versions and supported request formats that you may need to extract in your tools). The list of advertised resources is acquired using the `listresources` call, and returns a large XML document describing the available resources, which is relatively difficult to work with without a tool.

---

**Note:** We will be using GENI AM API version 2 throughout this tutorial. Some API call names will be different if you elect to interact with aggregates using AM API version 3 in the future.

---

- Lets start by getting an advertisement from a single aggregate. If you built a custom context using Python code you will need to replace the code below to load your custom context:

  ```
  $ python
  >>> import geni.util
  >>> context = geni.util.loadContext()
  >>> import geni.aggregate.instageni as IGAM
  >>> ad = IGAM.Illinois.listresources(context)
  ```

  Now of course we have an advertisement (assuming everything went well) stored into a Python object, which is reasonably boring!

---

**Note:** If you get timeouts or failures, you may want to try a different InstaGENI aggregate (this one may be particularly busy). You can get a list of (mostly) aggregate objects by using the `dir()` command on the IGAM module - `dir(IGAM)`.

---

- We can simply print out the advertisement raw text to see what the aggregate sent us:

```
>>> print ad.text
<rspec xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ......
...
```

  As you can see, even with this relatively small rack (5 hosts) the amount of data is significant.

- As `geni-lib` has parsed this advertisement into a more functional object, we have access to data objects instead of just raw xml. For example, we can inspect the routable address space available at a site:

```
>>> ad.routable_addresses.available
167
>>> ad.routable_addresses.capacity
190
```

- You may have noticed that if you just print the `routable_addresses` attribute, you get nothing useful:

```
>>> ad.routable_addresses
<geni.rspec.pgad.RoutableAddresses object at 0x1717f10>
```

  While we are adding online documentation for `geni-lib` objects, there are many objects that are undocumented. However, you can still gain some insight by using the `dir()` built-in to see what attributes are available:

```
>>> dir(ad.routable_addresses)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute_
↪_', '__hash__',
'__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
↪_setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'available', 'capacity
↪', 'configured']
```

  In general attributes starting with underscores are not useful to us, so we can see 3 attributes of value - `available`, `capacity`, and `configured`. In most cases their meanings should be obvious, so just knowing they exist even without documentation is quite helpful.

- There are also 3 iterators that are provided with `Advertisement` objects - `nodes`, `links`, and `shared_vlans`:

```
>>> for svlan in ad.shared_vlans:
...   print svlan
...
mesoscale-openflow
exclusive-openflow-1755
exclusive-openflow-1756
exclusive-openflow-1757
...snip...
```

- While `shared_vlans` just iterates over a set of strings, `node` objects are much more complex and have many more attributes and nested data structures to allow you to fully inspect their state:

```
>>> print dir(ad.nodes[0])
[..., 'available', 'component_id', 'component_manager_id', 'exclusive', 'hardware_
↪types', 'images',
'interfaces', 'location', 'name', 'shared', 'sliver_types']
```

- Particularly useful for the puposes of binding requests to certain nodes at a given site is the component_id:

```
>>> for node in ad.nodes:
...     print node.component_id
...
urn:publicid:IDN+instageni.illinois.edu+node+procurve2
urn:publicid:IDN+instageni.illinois.edu+node+pc3
urn:publicid:IDN+instageni.illinois.edu+node+pc5
urn:publicid:IDN+instageni.illinois.edu+node+interconnect-ion
urn:publicid:IDN+instageni.illinois.edu+node+pc1
urn:publicid:IDN+instageni.illinois.edu+node+interconnect-campus
urn:publicid:IDN+instageni.illinois.edu+node+pc2
urn:publicid:IDN+instageni.illinois.edu+node+interconnect-geni-core
urn:publicid:IDN+instageni.illinois.edu+node+pc4
urn:publicid:IDN+instageni.illinois.edu+node+internet
```

- Spend some time inspecting the other attributes of each node. You can get a specific node by using Python indexing on the `nodes` iterator:

```
>>> node = ad.nodes[1]
>>> node.component_id
'urn:publicid:IDN+instageni.illinois.edu+node+pc3'
```

### 4.4.3 Iterating Over Aggregates

Often you will want to inspect a large number of aggregates (particularly if there are of an idential or similar type) in order to find those that have availability in the resources that you require. The aggregate modules in `geni-lib` provide some convenience methods for assisting in this task:

```
>>> import geni.aggregate.instageni as IGAM
>>> for am in IGAM.aggregates():
...     print am.name
...
ig-cenic
ig-cwru
ig-clemson
ig-cornell
ig-ohmetrodc
ig-gatech
ig-gpo
ig-illinois
...snip...
```

Using this iterator you can act on each aggregate in a given module with the same snippet of code.

- Lets try getting (and saving) the `getversion` output from each InstaGENI site:

```
>>> import json
>>> for am in IGAM.aggregates():
...     print am.name
...     verdata = am.getversion(context)
...     ver_file = open("%s-version.json" % (am.name), "w+")
...     json.dump(verdata, ver_file)
...
ig-cenic
ig-cwru
ig-clemson
...snip...
```

This will write out a file for every aggregate (barring any exceptions) to the current directory.

---

**Note:** `verdata` in the above case is a Python `dict` object, so we need to pick a way to write it (in a human readable form) to a file. In the above example we pick serializing to JSON (which is reasonably readable), but you could also use the `pprint` module to format it nicely to a file as a nice string.

---

### 4.4.4 Exercises

We can now combine all of the above pieces, plus some Python knowledge, into some useful scripts.

1. Move the `getversion` code fragment above into a standalone script, and improve it to continue to the next aggregate if any exceptions are thrown by the current aggregate (unreachable, busy, etc.).
2. Write a script that prints out the number of availble routable IPs for each InstaGENI aggregate.

## 4.5 Creating a Request for a Single VM

This example walks through the basic of creating an RSpec (xml file) requesting a single VM from a compute aggregate. This example does not require that *geni-lib* is configured with user credentials or keys - it will create an XML file that you can feed into another tool such as *Jacks* or *Omni* (other examples cover how to make this request using *geni-lib* itself).

**Note:** You can find the complete source code for this example in a single file in the *geni-lib* distribution in *samples/onevm.py*.

### 4.5.1 Walk-through

- Since we only want to output the XML of the request, we need very few imports:

```
import geni.rspec.pg as PG
import geni.rspec.egext as EGX
import geni.rspec.igext as IGX
```

**Note:** While the first module is named 'pg' (after ProtoGENI), the base rspec format is common across compute aggregates and all will use the same *Request* container, although the resources in that container will differ based on what is available at a given site.

- Now we need to create the basic *Request* container:

```
r = PG.Request()
```

- Unfortunately there is no unified VM object for all compute aggregates, so you will need to know which "flavor" of compute aggregate you intend to use (most commonly either InstaGENI or ExoGENI).

**Note:** In later examples you will see how, if you are using *geni-lib* to make your reservations directly with the aggregates, you can indeed create a single VM request that can be used across aggregate "flavors".

- Now we will allocate a VM object that can be added to our request (examples shown here for both ExoGENI and InstaGENI):

```
# ExoGENI
exovm = EGX.XOSmall("vm1")

# InstaGENI
igvm = IGX.XenVM("vm1")
```

**Note:** The only required configuration for each resource is the *name* argument that is passed to the constructor. These names must be unique within a single site, but can be reused at different sites.

- For the purposes of this example we will only add the InstaGENI VM to the actual request that we will produce:

```
r.addResource(igvm)
```

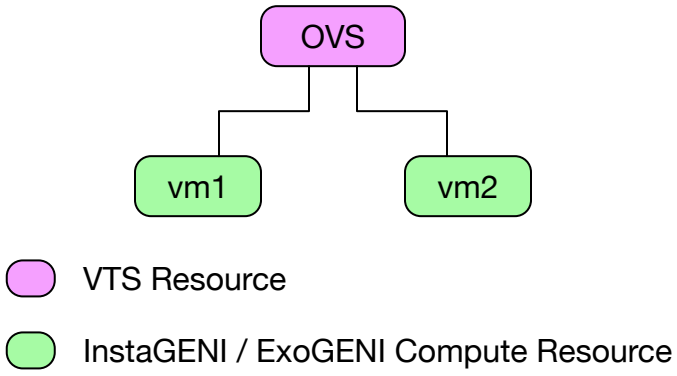
- Now that we have a request that contains a resource, we can write the XML to disk that represents this request:

```
r.writeXML("onevm-request.xml")
```

## 4.6 VTS: Basic Single-Site Topology

This example walks through creating a simple VTS topology with one forwarding element (an Open vSwitch instance) connected to two virtual machines provided by the site compute aggregate. Resources for this request will come from two different aggregate managers at the same site, using information returned from the VTS aggregate to structure the second request to the compute aggregate.

The resultant topology will be as in the diagram below. The resources are color-coded to indicate which aggregate will provision them:




This obviously has no advantage over having the local site compute aggregate connect the VMs directly, but it serves as the simplest example of how to stage the provisioning of VTS resources and forms the basis for more advanced use cases.

**Note:** This example requires that you have set up a valid context object with GENI credentials.

### 4.6.1 Set up VTS Sliver

For this example we'll use InstaGENI compute resources, but this would work for ExoGENI sites that have VTS support as well if you change the InstaGENI imports to the relevant ones for ExoGENI.

- We need to set up basic imports to create requests and send them to the aggregate:

```
import geni.rspec.pg as PG
import geni.rspec.igext as IGX
import geni.rspec.vts as VTS

import geni.aggregate.instageni as IGAM
import geni.aggregate.vts as VTSAM
```

- Here we also set up the slice name you're going to use, as well as the context object that specifies your credential information. If you set up your `geni-lib` using the GENI Portal Import method, the code below will directly work. If you built a custom context using Python code you will need to replace the code below to load your custom context:

```
import geni.util

context = geni.util.loadContext()
SLICENAME = "my-slice-name"  # Change this to be your slice name
```

**Note:** If you do not have a slice available in your project, you may need to go back to the GENI Portal web interface and create a new slice.

- VTS reservations are a two-stage process, where the VTS resources must be reserved first and the results used to create the proper compute request:

```
vtsr = VTS.Request()
```

- First we select the image we want to use for our VTS forwarding elements:

```
image = VTS.OVSL2Image()
```

**Note:** Images support varying functionality that can be configured here, such as sflow and netflow collectors, openflow controllers, etc.

- We then instantiate a single forwarding element with this image, and request two local circuits to connect to our VMs:

```
felement = VTS.Datapath(image, "fe0")
felement.attachPort(VTS.LocalCircuit())
felement.attachPort(VTS.LocalCircuit())
vtsr.addResource(felement)
```

- Now the request object is complete - we need to contact the aggregate and have it build our topology for us:

```
manifest = VTSAM.UtahDDC.createsliver(context, SLICENAME, vtsr)
```

**Note:** If you are at an in-person tutorial at GEC, etc., please replace `VTSAM.UtahDDC` with the aggregate you have been given on your tutorial worksheet.

### 4.6.2 Set up InstaGENI Compute Sliver

The VTS aggregate manager returned to us a manifest containing information about the resources we have provisioned - specifically identifying information about the local circuits we requested. We will now use this information to request compute resources and connect them to the VTS sliver.

- Initialize a basic GENI compute request:

```
igr = PG.Request()
```

- The VTS Manifest object allows us to iterate over the local circuits that were returned, and we'll make a VM with a single interface and link for each one, and give them IPs in the same subnet:

```
IP = "10.50.1.%d"

for idx,circuit in enumerate(manifest.local_circuits):
  vm = IGX.XenVM("vm%d" % (idx))
  intf = vm.addInterface("if0")
  intf.addAddress(PG.IPv4Address(IP % (idx+1), "255.255.255.0"))
  igr.addResource(vm)
```

(continues on next page)

(continued from previous page)

```
lnk = PG.Link()
lnk.addInterface(intf)
lnk.connectSharedVlan(circuit)
igr.addResource(lnk)
```

There is a lot of code above, but the workflow is fairly simple:

- First, we set up a simple string substitution so we can add a small number of IP addresses in the same subnet (otherwise the compute AM will give the interfaces IP addresses in different subnets and you will have to fix them after you log into the nodes).
- Next we iterate over all of the circuits returned from the VTS AM that match a certain type ("local"), while using the Python `enumerate` built-in to maintain a counter.
- For each circuit we create a VM object, add an interface to it, give that interface a unique IP address on our chosen subnet, and add that interface to a `Link` object, along with the circuit ID (which in this case is a shared VLAN).

- Now we just need to make the reservation and wait for our nodes to come up:

```
igm = IGAM.UtahDDC.createsliver(context, SLICENAME, igr)
geni.util.printlogininfo(manifest = igm)
```

---

**Note:** If you are at an in-person tutorial at GEC, etc., please replace `IGAM.UtahDDC` with the aggregate you have been given on your tutorial worksheet.

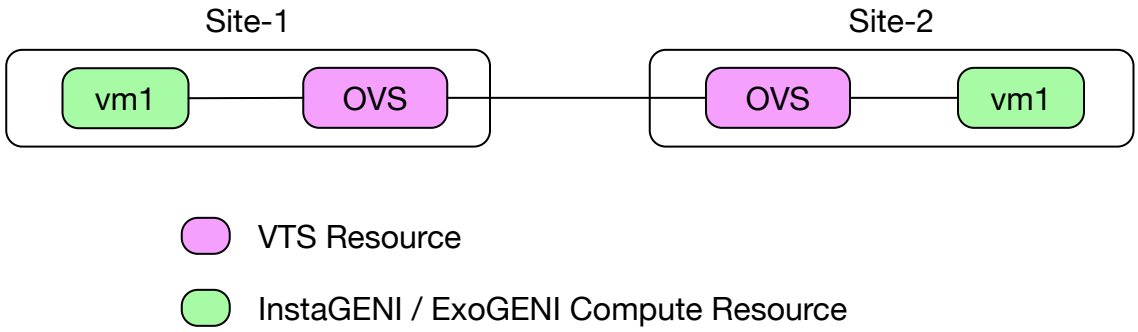
---

- In a few minutes you should be able to log into your VMs with the info printed out by the above step and send test traffic (ping, etc.) between the VMs across your VTS topology.
- Once you are done using your topology and exploring the tutorial, please delete all the resources you have reserved:

```
IGAM.UtahDDC.deletesliver(context, SLICENAME)
VTSAM.UtahDDC.deletesliver(context, SLICENAME)
```

## 4.7 VTS: Basic WAN Topology

This example walks through creating a two-site WAN topology with one forwarding element at each site. Like all VTS reservations that require compute resources, the resources for each site will come from two different aggregate managers. This example also employs further sequencing constraints in order to build the WAN circuit.




In order to build a circuit between two sites those sites need to share a common **circuit plane**. This is simply a named substrate that both sides have a common attachment to. In this tutorial we will use the `geni-al2s` circuit plane, which is currently available at most GENI VTS sites and replaces the `geni-mesoscale` circuit plane that is available at some sites but is being phased out.

**Note:** This example requires that you have set up a valid context object with GENI credentials.

For this example we'll use InstaGENI compute resources, but this would work for ExoGENI sites that have VTS support as well if you change the InstaGENI imports to the relevant ones for ExoGENI.

### 4.7.1 Set up VTS Slivers

We will first set up VTS slivers at both sites, before creating the local compute resources. This is not a strict requirement - you must always set up the VTS sliver at a site before the compute sliver, but you can request the compute sliver at a site before requesting the next site VTS sliver if that better fits your workflow.

In this example we will save the VTS manifests for later use to get compute resources, in case your interactive Python session needs to be restarted.

- We need to set up basic imports to create requests and send them to the aggregate:

```
import geni.rspec.pg as PG
import geni.rspec.igext as IGX
import geni.rspec.vts as VTS

import geni.aggregate.instageni as IGAM
import geni.aggregate.vts as VTSAM
```

- Here we also set up the slice name you're going to use, as well as the context object that specifies your credential information. If you set up your `geni-lib` using the GENI Portal Import method, the code below will directly work. If you built a custom context using your own Python code you will need to replace the code below to load your custom context:

```
import geni.util

```

(continues on next page)

(continued from previous page)

```
context = geni.util.loadContext()
SLICENAME = "my-slice-name"  # Change this to be your slice name
```

**Note:** If you do not have a slice available in your project, you may need to go back to the GENI Portal web interface and create a new slice. Also if you have multiple projects you may need to modify which one is being used by setting the `context.project` attribute.

- VTS reservations are typically a multistage process, where the VTS resources at a site must be reserved before the compute resources, or neighbour site VTS resources, and the results from the earlier reservations will be used to seed data in all subsequent reservations. In the case of WAN reservations we will need advertisement information from the *remote* VTS site we intend to connect our circuits to:

```
remote_ad = VTSAM.NPS.listresources(context)
```

- We need to search this remote advertisement for information that describes the endpoint we want to use for our chosen circuit plane:

```
for cp in remote_ad.circuit_planes:
  if cp.label == "geni-al2s":
    remote_endpoint = cp.endpoint
```

- We now start to build our primary site VTS request rspec:

```
s1r = VTS.Request()
```

- As in previous tutorials we will select a default L2 learning image for our forwarding elements:

```
image = VTS.OVSL2Image()
```

- We the instantiate a single forwarding element with this image, and request a local circuit to connect to our VM, as well as a WAN circuit to connect to the remote site:

```
felement = VTS.Datapath(image, "fe0")
felement.attachPort(VTS.LocalCircuit())
wan_port = felement.attachPort(VTS.GRECircuit("geni-al2s", remote_endpoint))
s1r.addResource(felement)
```

**Note:** We have chosen to use a GRE Circuit here to reach the remote site, although other types might be available. Each site advertises a list of supported encapsulation types for each circuit plane, allowing you to choose the one that best suits your needs based on performance and packet overhead.

- Now our request object is complete for our first site, so we can contact the aggregate manager and make the reservation:

```
ukym = VTSAM.UKYPKS2.createsliver(context, SLICENAME, s1r)
```

**Note:** If you are at an in-person tutorial you may need to replace `VTSAM.UKYPKS2` with the aggregate you have been given on your tutorial worksheet.

- We will write out our returned manifest to disk in case we need to restart our Python session:

```
ukym.writeXML("vts-ukypks2-manifest.xml")
```

- Now we will start building the VTS request at the remote site:

```
s2r = VTS.Request()
```

- The basic parts of the request are the same at each site:

```
felement = VTS.Datapath(image, "fe0")
felement.attachPort(VTS.LocalCircuit())
s2r.addResource(felement)
```

- Now we need to attach one port to our forwarding element that connects to the remote site that we have already configured:

```
felement.attachPort(VTS.GRECircuit("geni-al2s", ukym.findPort(wan_port.clientid).
↪local_endpoint))
```

  This searches our previous manifest for the WAN port we have already defined, and gathers the endpoint information to put in a remote request. The combination of this inforamtion will create a complete WAN circuit.

- Having created our request, we send it to the aggregate manager to reserve our resources, and write the output to a file:

```
npsm = VTSAM.NPS.createsliver(context, SLICENAME, s2r)
npsm.writeXML("vts-nps-manifest.xml")
```

### 4.7.2 Set up InstaGENI Compute Slivers

As we have two sites, we will need to set up our compute slivers at both sites, using the manifests returned from each VTS request. We want to set up IP addresses that we will use on both sides of our WAN topology:

```
IP = "10.50.1.%d"
NETMASK = "255.255.255.0"
```

- Each request is relatively simple, containing only a single VM connected to a single VTS port, pulled from the site VTS manifest:

```
ukyr = PG.Request()

for idx,circuit in enumerate(ukym.local_circuits):
  vm = IGX.XenVM("vm%d" % (idx))
  intf = vm.addInterface("if0")
  intf.addAddress(PG.IPv4Address(IP % (1), NETMASK))
  ukyr.addResource(vm)
  lnk = PG.Link()
  lnk.addInterface(intf)
  lnk.connectSharedVlan(circuit)
  ukyr.addResource(lnk)
```

The code above is the same as in earlier tutorials, which you can refer to for more thorough explanation.

- Now we make the reservation:

```
ukyigm = IGAM.UKYPKS2.createsliver(context, SLICENAME, ukyr)
geni.util.printlogininfo(manifest=ukyigm)
```

- We execute nearly identical code for the second site (note the IP address change):

```
npsr = PG.Request()

for idx,circuit in enumerate(npsm.local_circuits):
  vm = IGX.XenVM("vm%d" % (idx))
  intf = vm.addInterface("if0")
  intf.addAddress(PG.IPv4Address(IP % (2), NETMASK))
  npsr.addResource(vm)
  lnk = PG.Link()
  lnk.addInterface(intf)
  lnk.connectSharedVlan(circuit)
  npsr.addResource(lnk)
```

- Now we make the second site reservation:

```
npsigm = IGAM.NPS.createsliver(context, SLICENAME, npsr)
geni.util.printlogininfo(manifest=npsigm)
```

- In a few minutes you should be able to log into your VMs with the info printed out by the above step and send test traffic (ping, etc.) between the VMs across your VTS WAN topology.
- Once you are done using your topology and exploring the tutorial, please delete all the resources you have reserved:

```
IGAM.NPS.deletesliver(context, SLICENAME)
IGAM.UKYPKS2.deletesliver(context, SLICENAME)
VTSAM.NPS.deletesliver(context, SLICENAME)
VTSAM.UKYPKS2.deletesliver(context, SLICENAME)
```

# CHAPTER 5

# API

## 5.1 geni.aggregate

### 5.1.1 geni.aggregate.cloudlab

**class CloudLabAM**(*name*, *host*, *cmid=None*, *url=None*)

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**amtype**

**api**

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**geniCancelUpdateUsers**(*context*, *sname*)

**geniRestart**(*context*, *sname*, *urns*)

**geniStart**(*context*, *sname*)

**geniUpdateUsers**(*context*, *sname*, *user_info_list*)

**getConsoleURL**(*context*, *sname*, *urn*)

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

---

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

---

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregates**()

**name_to_aggregate**()

### 5.1.2 geni.aggregate.exogeni

**class EGCompute**(*name*, *host*, *cmid=None*, *url=None*)

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**amtype**

**api**

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregates**()

**name_to_aggregate**()

### 5.1.3 geni.aggregate.instageni

**class IGCompute**(*name*, *host*, *cmid=None*, *url=None*)

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**amtype**

**api**

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context

- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**geniCancelUpdateUsers**(*context*, *sname*)

**geniRestart**(*context*, *sname*, *urns*)

**geniStart**(*context*, *sname*)

**geniUpdateUsers**(*context*, *sname*, *user_info_list*)

**getConsoleURL**(*context*, *sname*, *urn*)

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

---

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

---

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context

- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregates**()

**cmid_to_aggregate**()

**name_to_aggregate**()

### 5.1.4 geni.aggregate.opengeni

**class OGCompute**(*name*, *host*, *cmid=None*, *url=None*)

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**amtype**

**api**

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

---

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

---

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregates**()

**name_to_aggregate**()

### 5.1.5 geni.aggregate.protogeni

**class PGCompute**(*name*, *host*, *cmid=None*, *url=None*)

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**amtype**

**api**

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**geniCancelUpdateUsers**(*context*, *sname*)

**geniRestart**(*context*, *sname*, *urns*)

**geniStart**(*context*, *sname*)

**geniUpdateUsers**(*context*, *sname*, *user_info_list*)

**getConsoleURL**(*context*, *sname*, *urn*)

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

---

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

---

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregates**()

**name_to_aggregate**()

### 5.1.6 geni.aggregate.transit

**class Transit**(*name*, *amtype*, *cmid*, *url*)

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**amtype**

**api**

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

---

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

---

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregates**()

**name_to_aggregate**()

### 5.1.7 geni.aggregate.vts

**class HostPOAs**(*vtsam*)

**execcmd**(*context*, *sname*, *client_ids*, *cmd*)

**getARPTable**(*context*, *sname*, *client_ids*)

**getRouteTable**(*context*, *sname*, *client_ids*)

**svcStatus**(*context*, *sname*, *client_ids*)

**class Policy**(*vtsam*)

**getText**(*context*, *pid=None*)
Get the text contents of the policy requested. If no policy is specified and only one policy exists at the aggregate, that policy text will be returned.

**Parameters**

- **context** – geni-lib context
- **pid** – policy ID (typically from *getversion* output)

**Returns** *str* – Text contents of policy

**giveConsent**(*context*, *pid*)
Give consent to the policy indicated for the user URN in the credential used.

**Parameters**

- **context** – geni-lib context
- **pid** – policy ID

**revokeConsent**(*context*, *pid*)
Revoke consent from this date forward to the policy indicated for the user URN in the credential used.

**Parameters**

- **context** – geni-lib context
- **pid** – policy ID

**class VTS**(*name*, *host*, *url=None*)
Wrapper for all VTS-supported AMAPI functions

**exception InvalidRSpecPathError**(*path*)

**args**

**message**

**exception UnspecifiedComponentManagerError**

**args**

**message**

**addDNSResourceRecord**(*context*, *sname*, *client_id*, *record_name*, *record_type*, *record_value*, *record_ttl=7200*)

**addFlows**(*context*, *sname*, *flows*)

**addSSHKeys**(*context*, *sname*, *client_ids*, *keys*)

**allocate**(*context*, *sname*, *rspec*)

**amtype**

**api**

**changeController**(*context*, *sname*, *url*, *datapaths*, *ofver=None*)

**clearFlows**(*context*, *sname*, *datapaths*)
Clear all installed flows from the requested datapaths.

**Parameters**

- **context** – geni-lib context

- **sname** (*str*) – Slice name
- **datapaths** (*list*) – A list of datapath client_id strings

**clearL2Table**(*context*, *sname*, *client_ids*)

**component_manager_id**

**createsliver**(*context*, *sname*, *rspec*)
GENI AM APIv2 method to reserve resources at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **rspec** (*geni.rspec.RSpec*) – Valid request RSpec

**deleteDNSResourceRecord**(*context*, *sname*, *client_id*, *record_name*, *record_type*)

**deletesliver**(*context*, *sname*)
GENI AM APIv2 method to delete a resource reservation at this aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**dropboxFinalize**(*context*, *authcode*)
Finalize the Dropbox account link for this aggregate.

**Parameters**

- **context** – geni-lib context
- **authcode** (*str*) – Authorization code given by Dropbox

**dropboxLink**(*context*)
Link your user_urn to a Dropbox account at this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *str* – Dropbox authorization URL to paste into web browser

**dropboxUpload**(*context*, *sname*, *cvols*)
Trigger upload to associated Dropbox account from requested container volumes.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **cvols** (*list*) – List of *(container client-id, volume-id)* tuples

**dumpFlows**(*context*, *sname*, *datapaths*, *show_hidden=False*)
Get the current flows and flow stats from the requested datapaths.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **datapaths** (*list*) – A list of datapath client_id strings
- **show_hidden** (*bool*) – Show hidden flows (if any)

**Returns** *dict* – Key/Value dictionary of format *{ client_id : [(flow_field, ...), ... ] }*

**getAllDNSResourceRecords**(*context*, *sname*, *client_ids*)

**getL2Table**(*context*, *sname*, *client_ids*)

**getLastDNSDHCPops**(*context*, *sname*, *client_ids*, *number_of_operations*, *dns_OR_dhcp*)

**getLeaseInfo**(*context*, *sname*, *client_ids*)

**getPortInfo**(*context*, *sname*, *datapaths*)

**getSTPInfo**(*context*, *sname*, *datapaths*)

**getversion**(*context*)
GENI AM API method to get the version information for this aggregate.

**Parameters** **context** – geni-lib context

**Returns** *dict* – Dictionary of key/value pairs with version information from this aggregate.

**hgPull**(*context*, *sname*, *cvols*)
Update an HgMount volume with the latest data from the source repository.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **cvols** (*list*) – List of *(container client-id, volume-id)* tuples

**listresources**(*context*, *sname=None*, *available=False*)
GENI AM APIv2 method to get available resources from an aggregate, or resources allocated to a specific sliver.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name (optional)
- **available** (*bool*) – Only list available resources

**Returns** *geni.rspec.RSpec* – If *sname* is provided, *listresources* will return a manifest rspec for the given slice name. Otherwise, *listresources* will return the advertisement rspec for the given aggregate.

**portDown**(*context*, *sname*, *client_id*)

**portUp**(*context*, *sname*, *client_id*)

**provision**(*context*, *sname*)

**renewsliver**(*context*, *sname*, *date*)
GENI AM APIv2 method to renew a sliver until the given datetime.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **date** (*str*) – RFC 3339-compliant date string for new expiration date

**Note:** Aggregates may have maximum expiration limits, restricting how far in the future you can set your expiration. This call may result in an error in such cases, or success with a sooner future date.

**setDHCPSubnet**(*context*, *sname*, *subnet_tuples*)

**setDeleteLock**(*context*, *sname*)
Prevent the given sliver from being deleted by another user with the credential.

**Note:** Locks are cumulative, and removed by calling *deletesliver*. When the last locking user calls *deletesliver*, the sliver will be deleted. It is not possible to remove your lock without risking deletion.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**setPortBehaviour**(*context*, *sname*, *port_list*)

**setPortTrunk**(*context*, *sname*, *port_list*)

**setPortVLAN**(*context*, *sname*, *port_tuples*)

**sliverstatus**(*context*, *sname*)
GENI AM APIv2 method to get the status of a current sliver at the given aggregate.

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name

**Returns** *dict* – Mapping of key/value pairs for status information the aggregate supports.

**aggregateFromHost**(*host*)

**aggregates**()

**name_to_aggregate**()

**class v4RouterPOAs**(*vtsam*)

**addOSPFNetworks**(*context*, *sname*, *client_ids*, *nets*)
Add OSPF Networks to areas on the given routers

**Parameters**

- **context** – geni-lib context
- **sname** (*str*) – Slice name
- **client_ids** (*list*) – A list of client-id strings
- **nets** (*list*) – A list of (network, area) tuples

**getOSPFNeighbors**(*context*, *sname*, *client_ids*)

**getRouteTable**(*context*, *sname*, *client_ids*)

## 5.2 geni.minigcf.config

**class HTTP**
Global configuration options for MiniGCF HTTP(S) calls.

**ALLOW_REDIRECTS = False**
Allow MiniGCF to follow HTTP redirects (301).

**LOG_RAW_REQUESTS = False**
If set to a valid *(log_handle, log_level)* tuple, will write all raw requests (before any parsing) to AM API and CH API calls to that log at the given level.

**LOG_RAW_RESPONSES = False**
If set to a valid *(log_handle, log_level)* tuple, will write all raw responses (before any parsing) from AM API and CH API calls to that log at the given level.

**LOG_URLS = False**
If set to a valid *(log_handle, log_level)* tuple, will log all URLs as they are used.

**TIMEOUT = 60**
Initial response timeout. Note that this is not the time limit on the entire download, just the initial server response.

## 5.3 geni.portal

Library for dealing with scripts that are run in the context of a portal.

**class Context**
Handle context for scripts being run inside a portal.

This class handles context for the portal, including where to put output RSpecs and handling parameterized scripts.

Scripts using this class can also be run "standalone" (ie. not by the portal), in which case they take parameters on the command line and put RSpecs on the standard output.

This class is a singleton. Most programs should access it through the portal.context variable; any additional "instances" of the object will be references to this.

**bindParameters**(*altParamSrc=None*)
Returns values for the parameters defined by defineParameter().

Returns a Namespace (like argparse), so if you call foo = bindParameters(), a parameter defined with name "bar" is accessed as foo.bar . Since defaults are required, all parameters are guaranteed to have values in the Namespace

If run standaline (not in the portal), parameters are pulled from the command line (try running with –help); if run in the portal, they are pulled from the portal itself. Or, if you provide the altParamSrc argument, you can specify your own parameters. If altParamSrc is a dict, we will bind the params as a dict, using the keys as parameter names, and the values as parameter values. If altParamSrc is a geni.rspec.pgmanifest.Manifest, we will extract the parameters and their values from the Manifest. Finally, if altParamSrc is a string, we'll try to parse it as a PG manifest xml document. No other forms of altParamSrc are currently specified.

**bindRequestRSpec**(*rspec*)
Bind the given request RSpec to the context, so that it can be automatically used with methods like printRequestRSpec.

At the present time, only one request can be bound to a context

**defineParameter**(*name*, *description*, *typ*, *defaultValue*, *legalValues=None*, *longDescription=None*, *advanced=False*, *groupId=None*, *hide=False*, *prefix='emulab.net.parameter.'*)

Define a new paramter to the script.

The given name will be used when parameters are bound. The description is brief help text that will be shown to the user when making his/her selection. The type should be one of the types defined by ParameterType. defaultValue is required, but legalValues (a list) is optional; the defaultValue must be one of the legalValues. Entries in the legalValues list may be either simple strings (eg. "m400"), in which case they will be show directly to the user, or 2-element tuples (eg. ("m400", "ARM64"),), in which the second entry is what is shown to the user. defaultValue may be a tuple, so that one can pass, say, 'legalvalues[0]' for the option. The longDescription is an optional, detailed description of this parameter and how it relates to other parameters; it will be shown to the user if they ask to see the help, or as a pop-up/tooltip. advanced, group, and groupName all provide parameter group abstractions. Parameter groups are hidden by default from the user, and the user can expand them to view and modify them if desired. By setting advanced to True, you create a parameter group named "Advanced Parameters"; this group will not exist or be shown if none of your parameters set the 'advanced' argument to True.

After defining parameters, bindParameters() must be called exactly once.

**defineParameterGroup**(*groupId*, *groupName*)

Define a parameter group. Parameters may be added to this group, which has an identifying token composed of alphanumeric characters (groupId), and a human-readable name (groupName). Groups are intended to be used for advanced parameters; in the portal UI, they hidden in an expandable panel with the groupName — and the user can choose to see and modify them, or not. You do not need to specify any groups; you can simply stuff all your parameters into the "Advanced Parameters" group by setting the 'advanced' argument of defineParameter to True. If you need multiple groups, define your own groups this way.

**makeParameterWarningsFatal**()

Enable this option if you want to return an error to the user for incorrect parameter values, even if they can be autocorrected. This can be useful to show the user that

**makeRequestRSpec**()

Make a new request RSpec, bind it to this context, and return it

**printRequestRSpec**(*rspec=None*)

Print the given request RSpec, or the one bound to this context if none is given.

If run standalone (not in the portal), the request will be printed to the standard output; if run in the portal, it will be placed someplace the portal can pick it up.

If the given rspec does not have a Tour object, this will attempt to build one from the file's docstring

**reportError**(*parameterError*, *immediate=False*)

Report a parameter error to the portal. @parameterError is an exception object of type ParameterError. If @immediate is True, your script will exit immediately at this point with a dump of the errors (and fatal warnings, if enabled via Context.makeParameterWarningsFatal) in JSON format. If @immediate is False, the errors will accumulate until Context.verifyParameters is called (and the errors will then be printed).

**reportWarning**(*parameterError*)

Record a parameter warning. Warnings will be printed if there are other errors or if warnings have been set to be fatal, when Context.verifyParameters() is called, or when there is another subsequent immediate error.

**suppressAutoPrint**()

Suppress the automatic printing of the bound RSpec that normally happens when the program exits.

**verifyParameters**()

If there have been calls to Context.parameterError, and/or to Context.parameterWarning (and Con-

text.makeParameterWarningsFatal has been called, making warnings fatal), this function will spit out some nice JSON-formatted exception info on stderr

**exception IllegalParameterDefaultError**(*val*)

**exception MultipleRSpecError**(*val*)

**exception NoRSpecError**(*val*)

**exception ParameterBindError**(*val*)

**exception ParameterError**(*message*, *paramList*)
A simple class to describe a parameter error. If you need to report an error with a user-specified parameter value to the Portal UI, please create (don't throw) one of these error objects, and tell the Portal about it by calling Context.reportError.

**class ParameterType**
Parameter types understood by Context.defineParameter().

**AGGREGATE = 'aggregate'**
URN specifying an Aggregate Manger

**BANDWIDTH = 'bandwidth'**
Floating-point number to be used for bandwidth

**BOOLEAN = 'boolean'**
True/False

**IMAGE = 'image'**
URN specifying a particular image

**INTEGER = 'integer'**
Simple integer

**LATENCY = 'latency'**
Floating-point number to be used for latency

**LOSSRATE = 'lossrate'**
Floating-point number 0.0 <= N < 1.0

**NODETYPE = 'nodetype'**
String specifying a type of node

**PUBKEY = 'pubkey'**
An RSA public key.

**SIZE = 'size'**
Integer for size (eg. MB, GB, etc.)

**STRING = 'string'**
Any string

**exception ParameterWarning**(*message*, *paramList*, *fixedValues=None*)
A simple class to describe a parameter warning. If you need to report an warning with a user-specified parameter value to the Portal UI, please create (don't throw) one of these error objects, and tell the Portal about it by calling Context.reportWarning . The first time the Portal UI runs your geni-lib script with a user's parameter values, it turns on the "warnings are fatal" mode (and then warnings are reported as errors). This gives you a chance to warn the user that they might be about to do something stupid, and/or suggest a set of modified values that will improve the situation. .

**exception PortalError**(*message*)

**class PortalJSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

**default**(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

**context = <geni.portal.Context object>**

Module-global Context object - most users of this module should simply use this rather than trying to create a new Context object

## 5.4 geni.rspec

### 5.4.1 geni.rspec.emulab

Convenience library to load all extensions supported by Emulab-based aggregates. In most cases, you will not need to load these extension libraries individually, just load this one.

### 5.4.2 geni.rspec.emulab.emuext

Common set of RSpec extensions supported by many Emulab-based aggregates

**class BridgedLink**(*name=None*)

A bridged link is syntactic sugar used to create two links separated by an Emulab delay (bridge) node. The BridgedLink class will create the following topology:

left-link right-link

node1 ========== bridge ========== node2

The bridge is a special node type (sliver_type="delay") that tells the CM to insert an Emulab delay node instead of a plain (router) node. A delay node is a transparent Ethernet bridge between the left and right segments above, but on which the traffic can be shaped wrt. bandwidth, latency, and loss. For example:

# Create the bridged link between the two nodes. link = request.BridgedLink("link") # Add two interfaces link.addInterface(iface1) link.addInterface(iface2) # Give the link (bridge) some shaping parameters. link.bandwidth = 10000 link.latency = 15 link.plr = 0.01

**addInterface**(*interface*)

**bandwidth**

**latency**

**plr**

**class InstantiateOn**(*parent*)
Added to a node to specify that it a Xen VM should be bound to (instantiated on) another node in the topology. Argument is the node instance or the client id of another node in the topology.

**exception InvalidParent**(*parent*)

**class ProgramAgent**(*name*, *command*, *directory=None*, *onexpstart=False*)
Add an Emulab Program Agent, which can be controlled via the Emulab event system. Optional argument 'directory' specifies where to invoke the command from. Optional argument 'onexpstart' says to invoke the command when the experiment starts (time=0 in event speak). This is different than the Execute service, which runs every time the node boots.

**class ShapedLink**(*name=None*)
A ShapedLink is a synonym for BridgedLink

**class setCollocateFactor**(*mfactor*)
Added to a top-level Request object, this extension limits the number of VMs from one experiment that Emulab will collocate on each physical host.

**class setDelayImage**(*urn*)
Added to a top-level Request object, this extension sets the disk image that will be used for all delay nodes configured for the experiment.

**class setFailureAction**(*action*)
Added to a node this extension will tell Emulab based aggregates to ignore errors booting this node when starting an experiment. This allows the experiment to proceed so that the user has time to debug.

**class setForceShaping**
Added to a Link or LAN object, this extension forces Emulab link shaping to be enabled, even if it is not strictly necessary. This allows the link properties to be changed dynamically via the Emulab event system.

**class setNoBandwidthShaping**
Added to a Link or LAN object, this extension forces Emulab link shaping to be disabled for bandwidth, even if it is necessary. This is ignored if the link must be shaped for other reason (delay, loss).

**class setNoInterSwitchLinks**
Added to a Link or LAN object, this extension forces the Emulab mapper to disallow mapping a link in the request topology to an inter-switch link. This allows users to require that specific nodes in their topology be attached to the same switch(es).

**class setPackingStrategy**(*strategy*)
Added to a top-level Request object, this extension controls the strategy used for distributing VMs across physical hosts

**class setRoutingStyle**(*style*)
Added to a top-level Request object, this extension controls the routing that is automatically configured on the experiment (data-plane) side of the network.

**class setUseTypeDefaultImage**
Added to a node that does not specify a disk image, this extension forces Emulab to use the hardware type default image instead of the standard geni default image. Useful with special hardware that should run a special image.

### 5.4.3 geni.rspec.igext

**class AddressPool**(*name*, *count=1*, *type='any'*)
A pool of public dynamic IP addresses belonging to a slice.

**name**

**class Blockstore**(*name*, *mount=None*)

**size**

**class Bridge**(*name*, *if0name='if0'*, *if1name='if1'*)

**class Pipe**

**getPipe**(*interface*)

**class Desire**(*name*, *weight*)

**class Firewall**(*style*)

**class Direction**

**INCOMING = 'incoming'**

**OUTGOING = 'outgoing'**

**class Style**

**BASIC = 'basic'**

**CLOSED = 'closed'**

**OPEN = 'open'**

**addException**(*port*, *direction*, *ip=None*)

**class OFController**(*host*, *port=6633*)

OpenFlow controller specification to be used on a PG VLAN.

Add to link objects using the Link.addChild() method.

---

**Note:** This will have no effect if a trivial link is created by the aggregate. You need to make sure that a VLAN will be provisioned (typically by making sure that at least two interfaces on the link are on different physical hosts).

---

**class ParameterData**(*parameters*)

**class Password**(*name=None*)

A declaration for a randomly generated password.

The portal will generate the password, encrypt it, and pass on the encrypted value to the AM(s) and therefore the node(s).

**class RemoteBlockstore**(*name*, *mount=None*, *ifacename='if0'*)

**dataset**

**interface**

**mountpoint**

**placement**

**readonly**

**rwclone**

**size**

**class Site**(*id*)

**class Tour**

**Description**(*type*, *desc*)

**Instructions**(*type*, *inst*)

**MARKDOWN = 'markdown'**

**SPLIT_REGEX = <_sre.SRE_Pattern object>**

**class Step**(*target*, *description*, *steptype=None*, *description_type='markdown'*)

**MARKDOWN = 'markdown'**

**TEXT = 'text'**

**TEXT = 'text'**

**addStep**(*step*)

**useDocstring**(*module=None*)

**class XenVM**(*client_id*, *component_id=None*, *exclusive=False*)
Xen-based Virtual Machine resource

**Parameters**

- **client_id** (*str*) – Your name for this VM. This must be unique within a single *Request* object.
- **component_id** (*Optional[str]*) – The *component_id* of the site node you want to bind this VM to
- **exclusive** (*Optional[bool]*) – Request this VM on an isolated host used only by your sliver.

**cores**
*int* – Number of CPU cores

**ram**
*int* – Amount of memory in megabytes

**disk**
*int* – Amount of disk space in gigabytes

**xen_ptype**
*str* – Physical node type on which to instantiate the VM. Types are AM-specific.

### 5.4.4 geni.rspec.pg

**class Address**(*atype*)

**class Command**(*cmd*, *data*)

**resolve**()

**exception DuplicateExtensionError**(*klass*)

**class Execute**(*shell*, *command*)

**class IPv4Address**(*address*, *netmask*)

**class Install**(*url*, *path*)

**class Interface**(*name*, *node*, *address=None*)

**exception InvalidAddressTypeError**(*addr*)

**addAddress**(*address*)

**name**

**class L2GRE**(*name=None*)

**class L3GRE**(*name=None*)

**class LAN**(*name=None*)

**class Link**(*name=None*, *ltype=''*, *members=None*)

**DEFAULT_BW = -1**

**DEFAULT_LAT = 0**

**DEFAULT_PLR = 0.0**

**EXTENSIONS = [('Site', <class 'geni.rspec.igext.Site'>), ('setForceShaping', <class 'g**

**LNKID = 0**

**addChild**(*obj*)

**addComponentManager**(*component_manager*)

**addInterface**(*intf*)

**addNode**(*node*)

**addRawElement**(*elem*)

**best_effort**

**connectSharedVlan**(*name*)

**disableMACLearning**()

**enableVlanTagging**()

**link_multiplexing**

**classmethod newLinkID**()

**trivial_ok**

**vlan_tagging**

**class Namespaces**

**CLIENT = http://www.protogeni.net/resources/rspec/ext/client/1**

**DATA = http://www.protogeni.net/resources/rspec/ext/user-data/1**

**DELAY = http://www.protogeni.net/resources/rspec/ext/delay/1**

**EMULAB = http://www.protogeni.net/resources/rspec/ext/emulab/1**

**INFO = http://www.protogeni.net/resources/rspec/ext/site-info/1**

**JACKS = http://www.protogeni.net/resources/rspec/ext/jacks/1**

**PARAMS = http://www.protogeni.net/resources/rspec/ext/profile-parameters/1**

**RS = http://www.protogeni.net/resources/rspec/ext/emulab/1**

**TOUR = http://www.protogeni.net/resources/rspec/ext/apt-tour/1**

**VTOP = http://www.protogeni.net/resources/rspec/ext/emulab/1**

**class Node**(*name*, *ntype*, *component_id=None*, *exclusive=None*)

**exception DuplicateInterfaceName**

**EXTENSIONS = [('Blockstore', <class 'geni.rspec.igext.Blockstore'>), ('Firewall', <cla**

**addInterface**(*name=None*, *address=None*)

**addRawElement**(*elem*)

**addService**(*svc*)

**name**

**class NodeType**

**RAW = 'raw'**

**VM = 'emulab-xen'**

**XEN = 'emulab-xen'**

**class RawPC**(*name*, *component_id=None*)

**class Request**

**EXTENSIONS = [('Link', <class 'geni.rspec.pg.Link'>), ('LAN', <class 'geni.rspec.pg.LA**

**addRawElement**(*elem*)

**addResource**(*rsrc*)

**addTour**(*tour*)

**hasTour**()

**resources**

**toXMLString**(*pretty_print=False*)

Return the current request contents as an XML string that represents an rspec in the GENIv3 format.

**writeXML**(*path*)

Write the current request contents as an XML file that represents an rspec in the GENIv3 format.

**class Resource**

**addNamespace**(*ns*)

**class Service**

**class StitchedLink**(*name=None*)

**exception TooManyInterfacesError**

**exception UnknownComponentManagerError**(*cid*)

**VM**
alias of *geni.rspec.pg.XenVM*

**class VZContainer**(*name*, *exclusive=False*)

**class XenVM**(*name*, *component_id=None*, *exclusive=False*)
Deprecated since version 0.4: Use *geni.rspec.igext.XenVM* instead.

### 5.4.5 geni.rspec.pgad

**class AdInterface**(*name*)
Wrapper object for a Node Interface in a GENIv3 Advertisement.

**component_id**
*str* – Component ID URN

**role**
*str* – The resource role of this interface (typically "control" or "experimental"). *None* if unset.

**name**
*str* – Friendly name for this interface, *None* if unset.

**class AdLink**

**text**

**class AdNode**
Wrapper object for a Node in a GENIv3 advertisement.

---

**Note:** In general this object is created on-demand through *Advertisement* objects, but you can load this object from a Node XML element by using the *_fromdom* classmethod.

**Attributes:** component_id (str): Component ID URN component_manager_id (str): Component Manager ID URN name (str): Friendly name provided by aggregate for this resource. exclusive (bool): True if a node can be reserved as a raw PC available (bool): Whether this node is currently available for reservations hardware_types (dict): Mapping of *{ type_name : type_slots, ... }* sliver_types (set): Supported sliver type images (dict): Mapping of *{ sliver_type : [supported_image_name, ... ], ... }* shared (bool): *True* if currently being used as a shared resource interfaces (list): List of *AdInterface* objects for this Node location (AdLocation): *None* if not available ram (int): Currently available system RAM in megabytes. *None* if not available. cpu (int): Maximum Per-core CPU speed in Mhz. *None* if not available.

---

**text**

**class AdSharedVLAN**

**class Advertisement**(*path=None*, *xml=None*)
Wrapper object for a GENIv3 XML advertisement.

Only one argument can be supplied (if both are provided *path* will be used)

**Parameters**

- **path** (*str, unicode*) – Path to XML file on disk containing an advertisement

- **xml** (*str, unicode*) – In-memory XML byte stream containing an advertisement

**images**
An iterable of the unique images found in this advertisement.

**links**
An indexable iterator over the AdLink objects in this advertisement.

**nodes**
An indexable iterator over the AdNode objects in this advertisement.

**routable_addresses**
A RoutableAddresses object containing the number of configured and available publicly routable IP addresses at this site.

**shared_vlans**
An indexable iterator of the shared vlan names found in this advertisement.

**stitchinfo**
Reference to the stitching info in the manifest, if present.

**text**
Advertisement XML contents as a string, formatted with whitespace for easier reading.

**writeXML**(*path*)
Write the current advertisement as an XML file that contains an rspec in the format returned by the aggregate.

**class Image**

**class Location**

**class RoutableAddresses**

**capacity**

### 5.4.6 geni.rspec.vts

**exception BadImageTypeError**(*rtype*)

**class Container**(*image*, *name*)

**EXTENSIONS = [('Mount', <class 'geni.rspec.vts.Mount'>), ('HgMount', <class 'geni.rspe**

**addIPRoute**(*network*, *gateway*)

**attachPort**(*port*)

**connectCrossSliver**(*other_dp*)

**class ContainerPort**(*target*, *vlan=None*, *delay_info=None*, *loss_info=None*)

**addIPv4Address**(*value*)

**class Datapath**(*image*, *client_id*)

**attachPort**(*port*)

**connectCrossSliver**(*other_dp*)

**name**

**class DatapathImage**(*name*)

**class DelayInfo**(*time=None*, *jitter=None*, *correlation=None*, *distribution=None*)

**class DropboxMount**(*name*, *mount_path*)

**class GRECircuit**(*circuit_plane*, *endpoint*)

**class HgMount**(*name*, *source*, *mount_path*, *branch='default'*)
Clone a public mercurial repo on a host

**Parameters**

- **name** (*str*) – a reference name given on the mounting AM, must be unique within a sliver
- **source** (*str*) – the URL to the source of repository
- **mount_path** (*str*) – the path where the repository would be mounted in the host filesystem
- **branch** (*str*) – the branch of the repository to be cloned on host (if any)

**exception IllegalModeForParamError**(*param*)

**class Image**(*name*)

**setImageAttribute**(*name*, *val*)

**class InternalCircuit**(*target*, *vlan=None*, *delay_info=None*, *loss_info=None*)

**class L2SSLVPNClient**(*client_id*)

**LocalCircuit**
alias of *geni.rspec.vts.PGCircuit*

**class LossInfo**(*percent*)

**class MirrorPort**(*port*)

**class Mount**(*type*, *name*, *mount_path*)

**class Namespaces**

**SDN = http://geni.bssoftworks.com/rspec/ext/sdn/request/1**

**VTS = http://geni.bssoftworks.com/rspec/ext/vts/request/1**

**class NetFlow**(*collector_ip*)

**class OVSImage**(*name*)

**netflow**

**setMirror**(*port*)

**sflow**

**class OVSL2Image**

**class OVSL2STP**

**RSTP = 2**

**STP = 1**

**address**

**ageing_time**

**forward_delay**

**hello_time**

**max_age**

**mode**

**priority**

**system_id**

**type**

**xmit_hold_count**

**class OVSOpenFlowImage**(*controller*, *ofver='1.0'*, *dpid=None*)

**class PGCircuit**(*name=None*, *delay_info=None*)

**class Port**(*name=None*)

**class ReorderInfo**(*percent*, *correlation*, *gap=None*)

**class Request**

**EXTENSIONS = [('SSLVPNFunction', <class 'geni.rspec.vts.SSLVPNFunction'>), ('L2SSLVPNS**

**addResource**(*rsrc*)

**resources**

**toXMLString**(*pretty_print=False*)

**writeXML**(*path*)

**class SFlow**(*collector_ip*)

**class SSLVPNFunction**(*client_id*)

**class SecureHgMount**(*getversion_output*, *name*, *source*, *mount_path*, *branch='default'*)

**rebind**(*getversion_output*)

**class SimpleDHCPImage**(*subnet=None*)

**exception UnknownSTPModeError**(*val*)

**class VFCircuit**(*target*)

**connectInternalCircuit**(*dp1*, *dp2*, *delay_info=None*, *loss_info=None*)

### 5.4.7 geni.rspec.vtsmanifest

**class Manifest**(*path=None*, *xml=None*)

Wrapper object for GENI XML manifest rspec, providing a pythonic API to the contained data

**containers**

Iterator over all allocated containers as `ManifestContainer` objects.

**datapaths**

Iterator over all allocated datapaths as `ManifestDatapath` objects.

**findPort**(*client_id*)
Get the datapath port object representing the given *client_id*.

**Parameters** **client_id** (*str*) – client_id of the port you want to find

**Returns** GenericPort or *None*

**findTarget**(*client_id*)
Get the container or datapath representing the given *client_id* in the manifest.

**Parameters** **client_id** (*str*) – Requested client ID of the object you want to find

**Returns** ManifestDatapath, ManifestContainer, or *None*

**functions**
Iterator over all allocated functions as ManifestFunction objects.

**local_circuits**
Iterator for allocated circuit names on the local PG circuit plane (as strings).

**pg_circuits**
Iterator for allocated circuit names on the local PG circuit plane (as strings).

**ports**
Iterator for all datapath and container ports as subclasses of GenericPort objects.

**text**
String representation of original XML content, with added whitespace for easier reading

**write**(*path*)
Deprecated since version 0.4: Use *geni.rspec.vtsmanifest.Manifest.writeXML()* instead.

**writeXML**(*path*)
Write the XML representation of this manifest to the supplied path.

**Parameters** **path** (*str*) – Path to output file

**exception UnhandledPortTypeError**(*typ*)

## 5.5 geni.types

**class DPID**(*val*)
Utility class representing OpenFlow Datapath IDs

This class tries to handle all likely inputs and desired outputs, while providing a single internal type to work with in the code.

String representations passed in must be represented in hex, but may contain common separators (colon, dash, and period) in any configuration.

**Parameters** **val** (*int, long, unicode, str*) –

**Raises**

- *DPID.OutOfRangeError* – If the DPID represented by *val* is larger than the spec allows or less than zero.
- *DPID.InputTypeError* – If *val* is not a supported data type

**exception InputTypeError**(*val*)

**MAX = 18446744073709551615L**

**exception OutOfRangeError**(*val*)

**hexstr**()
Unformatted hex representation of DPID

**Returns** *str* – Hex formatted DPID, without colons

**class EthernetMAC**(*val*)
Utility class representing 48-bit Ethernet MAC Addresses

This class tries to handle all likely inputs and desired outputs, while providing a single internal type to work with in the code.

String representations passed in must be represented in hex, but may contain common separators (colon, dash, and period) in any configuration.

**Parameters** **val**(*int, long, unicode, str*) –

**Raises**

- *EthernetMAC.OutOfRangeError* – If the MAC represented by *val* is larger than than 48-bits or less than zero.
- *EthernetMAC.InputTypeError* – If *val* is not a supported data type

**exception InputTypeError**(*val*)

**MAX = 281474976710656**

**exception OutOfRangeError**(*val*)

**hexstr**()
Unformatted hex representation of MAC

**Returns** *str* – Hex formatted MAC, without separators

## 5.6 geni.urn

Simple library for manipulating URNs, particularly those used for GENI objects

**Authority**(*authorities*, *name*)
Create a new GENI URN with type 'authority'.

**class Base**(**args*)
Base class representing any URN (RFC 2141).

**__init__**(**args*)
Create a new generic URN

URNs can be initialized in one of two ways:

1. Passing a single string in URN format ('urn:NID:NSS')
2. Passing two strings (the NID and the NSS) separately

**__repr__**()
x.__str__() <==> str(x)

**__str__**() <==> *str(x)*

**static isValidNID**(*s*)
Returns True if the string is a valid NID, False if not.

**static isValidNSS**(*s*)
Returns True if the string is a valid NSS, False if not.

**static isValidURN**(*s*)
Returns True if the string is a valid URN, False if not.

**class GENI**(**args*)
Class representing the URNs used by GENI, which use the publicid NID and IDN (domain name) scheme, then impose some additional strucutre.

**static GENIURNType**(*s*)
Returns the type of the object if the URN is a valid GENI URN, or None otherwise.

**TYPE_AUTHORITY = 'authority'**
Aggregate Managers, Slice Authorities, etc.

**TYPE_IMAGE = 'image'**
Disk images

**TYPE_INTERFACE = 'interface'**
Network interfaces

**TYPE_LINK = 'link'**
Point-to-point and multipoint links

**TYPE_NODE = 'node'**
Physical and virtual machines

**TYPE_SLICE = 'slice'**
Container for allocated resources

**TYPE_SLIVER = 'sliver'**
Slice of a specific resource

**TYPE_USER = 'user'**
Principal

**__init__**(**args*)
Create a URN in the format used for GENI objects

There are four forms of this constructor:

1. Pass a single string in GENI URN format ('urn:publicid:IDN+auth+type+name')
2. Pass three arguments: the authority (a single string), the type (see the **TYPE_** variables in this class), and the object name
3. Pass three arguments: as #2, but the authorit(ies) are passed as a list, with the top-level authority coming first, followed by any subauthorities
4. Pass three arguments: as #2, but the authority is a geni.aggregate.core.AM object, and the authority is taken from that object

**authorities**
Returns a list containing at least one authority string (the top level authority) and possibly additional subauthorities.

**authority**
Return a single string capturing the entire authority/subauthority chain

**static isValidGENIURN**(*s*)
Returns True if the given string is a valid URN in GENI format, False otherwise.

**name**
Returns the 'name' part of a GENI URN.

**type**
Returns the 'type' part of a GENI URN.

**Image** (*authorities*, *name*, *version=None*)
Create a new GENI URN with type 'image'.

**Interface** (*authorities*, *name*)
Create a new GENI URN with type 'interface'.

**Link** (*authorities*, *name*)
Create a new GENI URN with type 'link'.

**Make** (*s*)
Returns the 'most specific' URN object that it can for the given string.

Specifically, returns a GENI URN if the string is in GENI format, or a Base URN if it is not. May throw a MalformedURNError exception if the string is not a valid URN at all.

**exception MalformedURNError** (*val*)
Exception indicating that a string is not a proper URN.

**__init__** (*val*)
x.__init__(...) initializes x; see help(type(x)) for signature

**__str__** () <==> *str(x)*

**Node** (*authorities*, *name*)
Create a new GENI URN with type 'node'.

**Slice** (*authorities*, *name*)
Create a new GENI URN with type 'slice'.

**Sliver** (*authorities*, *name*)
Create a new GENI URN with type 'sliver'.

**User** (*authorities*, *name*)
Create a new GENI URN with type 'user'.

## 5.7 geni.util

**class APIEncoder** (*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

**default** (*obj*)
Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

**exception MissingPublicKeyError**

**exception PathNotFoundError**(*path*)

**buildContextFromBundle**(*bundle_path*, *pubkey_path=None*, *cert_pkey_path=None*)

**builddot**(*manifests*)
Constructs a dotfile of the topology described in the passed in manifest list and returns it as a string.

**checkavailrawpc**(*context*, *am*)
Returns a list of node objects representing available raw PCs at the given aggregate.

**deleteSliverExists**(*am*, *context*, *slice*)
Attempts to delete all slivers for the given slice at the given AM, suppressing all returned errors.

**getAdvertisements**(*context*, *ams*)
Returns a dictionary of the form:

```
{ site_object : advertisement_object, ...}
```

Containing the advertisements for all the requested aggregates. Requests are made in parallel and the function blocks until the slowest site returns (or times out).

**Warning:** Particularly large advertisements may break the shared memory queue used by this function.

**getManifests**(*context*, *ams*, *slices*)
Returns a two-level dictionary of the form:

```
{slice_name : { site_object : manifest_object, ... }, ...}
```

Containing the manifests for all provided slices at all the provided sites. Requests are made in parallel and the function blocks until the slowest site returns (or times out).

**hasDataContext**()

**loadAggregates**(*path=None*)

**loadContext**(*path=None*, *key_passphrase=None*)

**printlogininfo**(*context=None*, *am=None*, *slice=None*, *manifest=None*)
Prints out host login info in the format:

```
[client_id][username] hostname:port
```

If a manifest object is provided the information will be mined from this data, otherwise you must supply a context, slice, and am and a manifest will be requested from the given aggregate.

**saveAggregates**(*ammap*, *path=None*)

**updateAggregates**(*context*, *ammap*)

CHAPTER 6

# Development

This section of the documentation is for people wishing to contribute to the `geni-lib` project, or at least hoping to gain further insight into the existing code internals. This documentation is even less complete than the other sections, but hopefully will be useful.

## 6.1 Supported Use Cases

This document makes a weak stab at articulating the use cases which `geni-lib` is expected to support. This is in an attempt to provide guidance on architecture decisions for new features to make sure we don't break existing use cases.

### 6.1.1 Exact Request Rspec Creation

The most basic use of `geni-lib` is to write a small script whose sole purpose is to create a single rspec, from a simple evaluation of end-to-end instructions. Such a use may involve basic parameters, but does not take input from active querying of the federation. This provides code that is easy to edit to change behaviour, but otherwise is indistinguishable from editing an XML file.

This use case should not be complicated.

### 6.1.2 Modular / Multi-rspec Creation

It is often useful to provide subtrees of resource definitions that are not complete Request objects (a specific VM disk image, memory/disk configuration and execution scripts, etc). These trees can then be composed into Request objects for novel topologies. In this vein, it is also useful to create more than one Request at a time, composing them together for issuing to separate aggregates.

### 6.1.3 Federation Querying

It is valuable and must be possible for users to hold in memory multiple advertisement and manifest rspec wrappers at the same time. At the very minimum the following resource tuples must be able to exist in memory at the same time:

- (site, advertisement)
- (site, slice, manifest)

At the moment any number of instances of these combinations are supported - any change to restrict these instances to being unique (e.g. only one advertisement per site at a time, etc.) will have to be well justified.

### 6.1.4 Aggregate / Clearinghouse Actions

Users should be able to operate on many aggregates and clearinghouses in the same script. There is a current (and likely ongoing) requirement that the user only be configured to use one control framework (and credentials) at a time. If more than one CH supports the same credentials and API, they should be able to be used concurrently, as AMs also must. Users are responsible for completing all previous credentialed federation actions before changing the CF or credentials their context refers to.

## 6.2 Coding Conventions

There is a `pylint.rc` file that is tweaked for most of the project style. It is not perfect, but is a good check to run immediately after cloning, and then right before a pull request (as `pylint` will report the difference in compliance once you have run it once). The script `lint.sh` in the root directory will run `pylint` with this file over the proper directory tree.

- All indentation uses spaces
- Indents are 2 spaces
- Exceptions that exist only for specializing names should be written on one line:

```
class MyCommonException(Exception):
  def __init__ (self, some_data):
    ...
  def __str__ (self):
    ...

class SpecificNameOne(MyCommonException): pass
class SpecificNameTwo(MyCommonException): pass
```

  There is an example of this use case in `geni.aggregate.pgutil`

- Maximum line length is 132 characters
- Class-level and global variables are **highly** discouraged
- Never use bare `except:` clauses
- Do not use `print`

## 6.3 Pattern Conventions

- Exception hierarchies are highly encouraged, as they allow users to dispatch their own scripting on the the type of exception, rather than having a single exception with varying messages or `errno`-like mechanics.
- Anything that imports `cryptography` should never be imported at module level. Always import crypto and network functionality inside functions, as this minimizes the dependencies for users who are only generating or parsing XML.
- All public attribute storage types should be internally consistent for reads. If you want to support setting an underlying float storage type using a string or integer, use a property. Non-public attributes should also follow this rule unless there is a really good reason not to.
- If you can foresee a problem in the future, but don't have time to fix it now, at least leave a `# TODO` note.
- At the moment many `geni-lib` instances are re-entrant. However, while we should support this where possible, it is not required nor will it be guaranteed to users.

## 6.4 Philosophy Notes

`geni-lib` is a surprisingly useful tool for end-users. However, that does not mean it is designed as a user tool. As it is a library, constraints and "convenience" can always be wrapped around it, but they can't be removed if the library is too opinionated at a base level. Convenience is generally acceptable if the underlying functionality can be directly accessed by the user should they want to avoid whatever level of "help" is being offered.

- Do not provide Magic(tm) to users in the base API. `geni-lib` should not go out of its' way to protect the user from building a request that we believe the be impossible to satisfy, or from passing "bad" data to AM API calls.
- AM wrappers provide some basic level of sanity checking of input values (specifically for POAs in the VTS support). This is acceptable as there is a lower-level API a user may use if they want to ignore those checks (`geni.minigcf.amapi3`). That being said, this level of convenience should be limited.
- While we support IronPython and Jython, those `geni-lib` users should still prefer `multiprocessing` for parallel execution, instead of `threading`.
- Providing convenience shouldn't extend to essentially providing new tools - it's hard to say where this line is, but providing too much tool-like functionality in `geni-lib` puts pressure on both the release schedule and the versioning and API stability.

## 6.5 Things That Don't Belong

For legacy reasons, `geni-lib` includes code that either doesn't belong entirely, or is in the wrong place. Anything enumerated here should not be referred to as an example of good practice moving forward:

- `geni.aggregate.*` - This package is a mess. Frameworks need to move out somewhere else, and the AM locations should not be in the code (they should be loaded from data files which can be updated independently of the `geni-lib` code). `context` also doesn't belong here, once framework support moves.
- `geni.rspec.pg*` - The "pg" rspec is now the "GENI" rspec, and we should rename accordingly. Things that are *actually* part of Emulab and not the base functionality should live in extensions.

CHAPTER 7

# Indices and tables

- modindex
- search

# Python Module Index


## a

geni.aggregate.cloudlab, 31
geni.aggregate.exogeni, 33
geni.aggregate.instageni, 34
geni.aggregate.opengeni, 36
geni.aggregate.protogeni, 37
geni.aggregate.transit, 39
geni.aggregate.vts, 40

## m

geni.minigcf.config, 45

## p

geni.portal, 45

## r

geni.rspec.emulab, 48
geni.rspec.emulab.emuext, 48
geni.rspec.igext, 49
geni.rspec.pg, 51
geni.rspec.pgad, 54
geni.rspec.vts, 55
geni.rspec.vtsmanifest, 57

## t

geni.types, 58

## u

geni.urn, 59
geni.util, 61

# Index

## Symbols


__init__() (Base method), 59
__init__() (GENI method), 60
__init__() (MalformedURNError method), 61
__repr__() (Base method), 59
__str__() (Base method), 59
__str__() (MalformedURNError method), 61


## A


addAddress() (Interface method), 52
addChild() (Link method), 52
addComponentManager() (Link method), 52
addDNSResourceRecord() (VTS method), 41
addException() (Firewall method), 50
addFlows() (VTS method), 41
addInterface() (BridgedLink method), 48
addInterface() (Link method), 52
addInterface() (Node method), 53
addIPRoute() (Container method), 55
addIPv4Address() (ContainerPort method), 55
addNamespace() (Resource method), 53
addNode() (Link method), 52
addOSPFNetworks() (v4RouterPOAs method), 44
addRawElement() (Link method), 52
addRawElement() (Node method), 53
addRawElement() (Request method), 53
addResource() (Request method), 53, 57
Address (class in geni.rspec.pg), 51
address (OVSL2STP attribute), 56
AddressPool (class in geni.rspec.igext), 49
addService() (Node method), 53
addSSHKeys() (VTS method), 41
addStep() (Tour method), 51
addTour() (Request method), 53
AdInterface (class in geni.rspec.pgad), 54
AdLink (class in geni.rspec.pgad), 54
AdNode (class in geni.rspec.pgad), 54
AdSharedVLAN (class in geni.rspec.pgad), 54
Advertisement (class in geni.rspec.pgad), 54
ageing_time (OVSL2STP attribute), 57
AGGREGATE (ParameterType attribute), 47
aggregateFromHost() (in module geni.aggregate.vts), 44
aggregates() (in module geni.aggregate.cloudlab), 32
aggregates() (in module geni.aggregate.exogeni), 34
aggregates() (in module geni.aggregate.instageni), 36
aggregates() (in module geni.aggregate.opengeni), 37
aggregates() (in module geni.aggregate.protogeni), 39
aggregates() (in module geni.aggregate.transit), 40
aggregates() (in module geni.aggregate.vts), 44
allocate() (VTS method), 41
ALLOW_REDIRECTS (HTTP attribute), 45
amtype (CloudLabAM attribute), 31
amtype (EGCompute attribute), 33
amtype (IGCompute attribute), 34
amtype (OGCompute attribute), 36
amtype (PGCompute attribute), 37
amtype (Transit attribute), 39
amtype (VTS attribute), 41
api (CloudLabAM attribute), 31
api (EGCompute attribute), 33
api (IGCompute attribute), 34
api (OGCompute attribute), 36
api (PGCompute attribute), 37
api (Transit attribute), 39
api (VTS attribute), 41
APIEncoder (class in geni.util), 61
args (CloudLabAM.InvalidRSpecPathError attribute), 31
args (CloudLabAM.UnspecifiedComponentManagerError attribute), 31
args (EGCompute.InvalidRSpecPathError attribute), 33
args (EGCompute.UnspecifiedComponentManagerError attribute), 33
args (IGCompute.InvalidRSpecPathError attribute), 34
args (IGCompute.UnspecifiedComponentManagerError attribute), 34
args (OGCompute.InvalidRSpecPathError attribute), 36
args (OGCompute.UnspecifiedComponentManagerError attribute), 36
args (PGCompute.InvalidRSpecPathError attribute), 37

args (PGCompute.UnspecifiedComponentManagerError attribute), 37
args (Transit.InvalidRSpecPathError attribute), 39
args (Transit.UnspecifiedComponentManagerError attribute), 39
args (VTS.InvalidRSpecPathError attribute), 41
args (VTS.UnspecifiedComponentManagerError attribute), 41
attachPort() (Container method), 55
attachPort() (Datapath method), 55
authorities (GENI attribute), 60
authority (GENI attribute), 60
Authority() (in module geni.urn), 59

## B

BadImageTypeError, 55
bandwidth (BridgedLink attribute), 48
BANDWIDTH (ParameterType attribute), 47
Base (class in geni.urn), 59
BASIC (Firewall.Style attribute), 50
best_effort (Link attribute), 52
bindParameters() (Context method), 45
bindRequestRSpec() (Context method), 45
Blockstore (class in geni.rspec.igext), 50
BOOLEAN (ParameterType attribute), 47
Bridge (class in geni.rspec.igext), 50
Bridge.Pipe (class in geni.rspec.igext), 50
BridgedLink (class in geni.rspec.emulab.emuext), 48
buildContextFromBundle() (in module geni.util), 62
builddot() (in module geni.util), 62

## C

capacity (RoutableAddresses attribute), 55
changeController() (VTS method), 41
checkavailrawpc() (in module geni.util), 62
clearFlows() (VTS method), 41
clearL2Table() (VTS method), 42
CLIENT (Namespaces attribute), 52
CLOSED (Firewall.Style attribute), 50
CloudLabAM (class in geni.aggregate.cloudlab), 31
CloudLabAM.InvalidRSpecPathError, 31
CloudLabAM.UnspecifiedComponentManagerError, 31
cmid_to_aggregate() (in module geni.aggregate.instageni), 36
Command (class in geni.rspec.pg), 51
component_id (AdInterface attribute), 54
component_manager_id (CloudLabAM attribute), 31
component_manager_id (EGCompute attribute), 33
component_manager_id (IGCompute attribute), 34
component_manager_id (OGCompute attribute), 36
component_manager_id (PGCompute attribute), 37
component_manager_id (Transit attribute), 39
component_manager_id (VTS attribute), 42
connectCrossSliver() (Container method), 55
connectCrossSliver() (Datapath method), 55
connectInternalCircuit() (in module geni.rspec.vts), 57
connectSharedVlan() (Link method), 52
Container (class in geni.rspec.vts), 55
ContainerPort (class in geni.rspec.vts), 55
containers (Manifest attribute), 57
Context (class in geni.portal), 45
context (in module geni.portal), 48
cores (XenVM attribute), 51
createsliver() (CloudLabAM method), 31
createsliver() (EGCompute method), 33
createsliver() (IGCompute method), 34
createsliver() (OGCompute method), 36
createsliver() (PGCompute method), 37
createsliver() (Transit method), 39
createsliver() (VTS method), 42

## D

DATA (Namespaces attribute), 52
Datapath (class in geni.rspec.vts), 55
DatapathImage (class in geni.rspec.vts), 56
datapaths (Manifest attribute), 57
dataset (RemoteBlockstore attribute), 50
default() (APIEncoder method), 61
default() (PortalJSONEncoder method), 48
DEFAULT_BW (Link attribute), 52
DEFAULT_LAT (Link attribute), 52
DEFAULT_PLR (Link attribute), 52
defineParameter() (Context method), 46
defineParameterGroup() (Context method), 46
DELAY (Namespaces attribute), 52
DelayInfo (class in geni.rspec.vts), 56
deleteDNSResourceRecord() (VTS method), 42
deletesliver() (CloudLabAM method), 31
deletesliver() (EGCompute method), 33
deletesliver() (IGCompute method), 35
deletesliver() (OGCompute method), 36
deletesliver() (PGCompute method), 38
deletesliver() (Transit method), 39
deletesliver() (VTS method), 42
deleteSliverExists() (in module geni.util), 62
Description() (Tour method), 51
Desire (class in geni.rspec.igext), 50
disableMACLearning() (Link method), 52
disk (XenVM attribute), 51
DPID (class in geni.types), 58
DPID.InputTypeError, 58
DPID.OutOfRangeError, 58
dropboxFinalize() (VTS method), 42
dropboxLink() (VTS method), 42
DropboxMount (class in geni.rspec.vts), 56
dropboxUpload() (VTS method), 42
dumpFlows() (VTS method), 42
DuplicateExtensionError, 52

## E

EGCompute (class in geni.aggregate.exogeni), 33
EGCompute.InvalidRSpecPathError, 33
EGCompute.UnspecifiedComponentManagerError, 33
EMULAB (Namespaces attribute), 52
enableVlanTagging() (Link method), 52
EthernetMAC (class in geni.types), 59
EthernetMAC.InputTypeError, 59
EthernetMAC.OutOfRangeError, 59
execcmd() (HostPOAs method), 40
Execute (class in geni.rspec.pg), 52
EXTENSIONS (Container attribute), 55
EXTENSIONS (Link attribute), 52
EXTENSIONS (Node attribute), 53
EXTENSIONS (Request attribute), 53, 57

## F

findPort() (Manifest method), 58
findTarget() (Manifest method), 58
Firewall (class in geni.rspec.igext), 50
Firewall.Direction (class in geni.rspec.igext), 50
Firewall.Style (class in geni.rspec.igext), 50
forward_delay (OVSL2STP attribute), 57
functions (Manifest attribute), 58

## G

GENI (class in geni.urn), 60
geni.aggregate.cloudlab (module), 31
geni.aggregate.exogeni (module), 33
geni.aggregate.instageni (module), 34
geni.aggregate.opengeni (module), 36
geni.aggregate.protogeni (module), 37
geni.aggregate.transit (module), 39
geni.aggregate.vts (module), 40
geni.minigcf.config (module), 45
geni.portal (module), 45
geni.rspec.emulab (module), 48
geni.rspec.emulab.emuext (module), 48
geni.rspec.igext (module), 49
geni.rspec.pg (module), 51
geni.rspec.pgad (module), 54
geni.rspec.vts (module), 55
geni.rspec.vtsmanifest (module), 57
geni.types (module), 58
geni.urn (module), 59
geni.util (module), 61
geniCancelUpdateUsers() (CloudLabAM method), 32
geniCancelUpdateUsers() (IGCompute method), 35
geniCancelUpdateUsers() (PGCompute method), 38
geniRestart() (CloudLabAM method), 32
geniRestart() (IGCompute method), 35
geniRestart() (PGCompute method), 38
geniStart() (CloudLabAM method), 32
geniStart() (IGCompute method), 35
geniStart() (PGCompute method), 38
geniUpdateUsers() (CloudLabAM method), 32
geniUpdateUsers() (IGCompute method), 35
geniUpdateUsers() (PGCompute method), 38
GENIURNType() (GENI static method), 60
getAdvertisements() (in module geni.util), 62
getAllDNSResourceRecords() (VTS method), 43
getARPTable() (HostPOAs method), 40
getConsoleURL() (CloudLabAM method), 32
getConsoleURL() (IGCompute method), 35
getConsoleURL() (PGCompute method), 38
getL2Table() (VTS method), 43
getLastDNSDHCPops() (VTS method), 43
getLeaseInfo() (VTS method), 43
getManifests() (in module geni.util), 62
getOSPFNeighbors() (v4RouterPOAs method), 44
getPipe() (Bridge method), 50
getPortInfo() (VTS method), 43
getRouteTable() (HostPOAs method), 40
getRouteTable() (v4RouterPOAs method), 44
getSTPInfo() (VTS method), 43
getText() (Policy method), 40
getversion() (CloudLabAM method), 32
getversion() (EGCompute method), 33
getversion() (IGCompute method), 35
getversion() (OGCompute method), 36
getversion() (PGCompute method), 38
getversion() (Transit method), 39
getversion() (VTS method), 43
giveConsent() (Policy method), 41
GRECircuit (class in geni.rspec.vts), 56

## H

hasDataContext() (in module geni.util), 62
hasTour() (Request method), 53
hello_time (OVSL2STP attribute), 57
hexstr() (DPID method), 58
hexstr() (EthernetMAC method), 59
HgMount (class in geni.rspec.vts), 56
hgPull() (VTS method), 43
HostPOAs (class in geni.aggregate.vts), 40
HTTP (class in geni.minigcf.config), 45

## I

IGCompute (class in geni.aggregate.instageni), 34
IGCompute.InvalidRSpecPathError, 34
IGCompute.UnspecifiedComponentManagerError, 34
IllegalModeForParamError, 56
IllegalParameterDefaultError, 47
Image (class in geni.rspec.pgad), 55
Image (class in geni.rspec.vts), 56
IMAGE (ParameterType attribute), 47
Image() (in module geni.urn), 61

images (Advertisement attribute), 55
INCOMING (Firewall.Direction attribute), 50
INFO (Namespaces attribute), 53
Install (class in geni.rspec.pg), 52
InstantiateOn (class in geni.rspec.emulab.emuext), 49
InstantiateOn.InvalidParent, 49
Instructions() (Tour method), 51
INTEGER (ParameterType attribute), 47
Interface (class in geni.rspec.pg), 52
interface (RemoteBlockstore attribute), 50
Interface() (in module geni.urn), 61
Interface.InvalidAddressTypeError, 52
InternalCircuit (class in geni.rspec.vts), 56
IPv4Address (class in geni.rspec.pg), 52
isValidGENIURN() (GENI static method), 60
isValidNID() (Base static method), 59
isValidNSS() (Base static method), 59
isValidURN() (Base static method), 59

## J

JACKS (Namespaces attribute), 53

## L

L2GRE (class in geni.rspec.pg), 52
L2SSLVPNClient (class in geni.rspec.vts), 56
L3GRE (class in geni.rspec.pg), 52
LAN (class in geni.rspec.pg), 52
latency (BridgedLink attribute), 48
LATENCY (ParameterType attribute), 47
Link (class in geni.rspec.pg), 52
Link() (in module geni.urn), 61
link_multiplexing (Link attribute), 52
links (Advertisement attribute), 55
listresources() (CloudLabAM method), 32
listresources() (EGCompute method), 33
listresources() (IGCompute method), 35
listresources() (OGCompute method), 36
listresources() (PGCompute method), 38
listresources() (Transit method), 39
listresources() (VTS method), 43
LNKID (Link attribute), 52
loadAggregates() (in module geni.util), 62
loadContext() (in module geni.util), 62
local_circuits (Manifest attribute), 58
LocalCircuit (in module geni.rspec.vts), 56
Location (class in geni.rspec.pgad), 55
LOG_RAW_REQUESTS (HTTP attribute), 45
LOG_RAW_RESPONSES (HTTP attribute), 45
LOG_URLS (HTTP attribute), 45
LossInfo (class in geni.rspec.vts), 56
LOSSRATE (ParameterType attribute), 47

## M

Make() (in module geni.urn), 61
makeParameterWarningsFatal() (Context method), 46
makeRequestRSpec() (Context method), 46
MalformedURNError, 61
Manifest (class in geni.rspec.vtsmanifest), 57
MARKDOWN (Tour attribute), 51
MARKDOWN (Tour.Step attribute), 51
MAX (DPID attribute), 58
MAX (EthernetMAC attribute), 59
max_age (OVSL2STP attribute), 57
message (CloudLabAM.InvalidRSpecPathError attribute), 31
message (CloudLabAM.UnspecifiedComponentManagerError attribute), 31
message (EGCompute.InvalidRSpecPathError attribute), 33
message (EGCompute.UnspecifiedComponentManagerError attribute), 33
message (IGCompute.InvalidRSpecPathError attribute), 34
message (IGCompute.UnspecifiedComponentManagerError attribute), 34
message (OGCompute.InvalidRSpecPathError attribute), 36
message (OGCompute.UnspecifiedComponentManagerError attribute), 36
message (PGCompute.InvalidRSpecPathError attribute), 37
message (PGCompute.UnspecifiedComponentManagerError attribute), 37
message (Transit.InvalidRSpecPathError attribute), 39
message (Transit.UnspecifiedComponentManagerError attribute), 39
message (VTS.InvalidRSpecPathError attribute), 41
message (VTS.UnspecifiedComponentManagerError attribute), 41
MirrorPort (class in geni.rspec.vts), 56
MissingPublicKeyError, 61
mode (OVSL2STP attribute), 57
Mount (class in geni.rspec.vts), 56
mountpoint (RemoteBlockstore attribute), 50
MultipleRSpecError, 47

## N

name (AddressPool attribute), 49
name (AdInterface attribute), 54
name (Datapath attribute), 55
name (GENI attribute), 60
name (Interface attribute), 52
name (Node attribute), 53
name_to_aggregate() (in module geni.aggregate.cloudlab), 33
name_to_aggregate() (in module geni.aggregate.exogeni), 34

name_to_aggregate() (in module geni.aggregate.instageni), 36
name_to_aggregate() (in module geni.aggregate.opengeni), 37
name_to_aggregate() (in module geni.aggregate.protogeni), 39
name_to_aggregate() (in module geni.aggregate.transit), 40
name_to_aggregate() (in module geni.aggregate.vts), 44
Namespaces (class in geni.rspec.pg), 52
Namespaces (class in geni.rspec.vts), 56
NetFlow (class in geni.rspec.vts), 56
netflow (OVSImage attribute), 56
newLinkID() (geni.rspec.pg.Link class method), 52
Node (class in geni.rspec.pg), 53
Node() (in module geni.urn), 61
Node.DuplicateInterfaceName, 53
nodes (Advertisement attribute), 55
NodeType (class in geni.rspec.pg), 53
NODETYPE (ParameterType attribute), 47
NoRSpecError, 47

## O

OFController (class in geni.rspec.igext), 50
OGCompute (class in geni.aggregate.opengeni), 36
OGCompute.InvalidRSpecPathError, 36
OGCompute.UnspecifiedComponentManagerError, 36
OPEN (Firewall.Style attribute), 50
OUTGOING (Firewall.Direction attribute), 50
OVSImage (class in geni.rspec.vts), 56
OVSL2Image (class in geni.rspec.vts), 56
OVSL2STP (class in geni.rspec.vts), 56
OVSOpenFlowImage (class in geni.rspec.vts), 57

## P

ParameterBindError, 47
ParameterData (class in geni.rspec.igext), 50
ParameterError, 47
ParameterType (class in geni.portal), 47
ParameterWarning, 47
PARAMS (Namespaces attribute), 53
Password (class in geni.rspec.igext), 50
PathNotFoundError, 61
pg_circuits (Manifest attribute), 58
PGCircuit (class in geni.rspec.vts), 57
PGCompute (class in geni.aggregate.protogeni), 37
PGCompute.InvalidRSpecPathError, 37
PGCompute.UnspecifiedComponentManagerError, 37
placement (RemoteBlockstore attribute), 50
plr (BridgedLink attribute), 48
Policy (class in geni.aggregate.vts), 40
Port (class in geni.rspec.vts), 57
PortalError, 47
PortalJSONEncoder (class in geni.portal), 47
portDown() (VTS method), 43
ports (Manifest attribute), 58
portUp() (VTS method), 43
printlogininfo() (in module geni.util), 62
printRequestRSpec() (Context method), 46
priority (OVSL2STP attribute), 57
ProgramAgent (class in geni.rspec.emulab.emuext), 49
provision() (VTS method), 43
PUBKEY (ParameterType attribute), 47

## R

ram (XenVM attribute), 51
RAW (NodeType attribute), 53
RawPC (class in geni.rspec.pg), 53
readonly (RemoteBlockstore attribute), 50
rebind() (SecureHgMount method), 57
RemoteBlockstore (class in geni.rspec.igext), 50
renewsliver() (CloudLabAM method), 32
renewsliver() (EGCompute method), 34
renewsliver() (IGCompute method), 35
renewsliver() (OGCompute method), 37
renewsliver() (PGCompute method), 38
renewsliver() (Transit method), 40
renewsliver() (VTS method), 43
ReorderInfo (class in geni.rspec.vts), 57
reportError() (Context method), 46
reportWarning() (Context method), 46
Request (class in geni.rspec.pg), 53
Request (class in geni.rspec.vts), 57
resolve() (Command method), 51
Resource (class in geni.rspec.pg), 53
resources (Request attribute), 53, 57
revokeConsent() (Policy method), 41
role (AdInterface attribute), 54
routable_addresses (Advertisement attribute), 55
RoutableAddresses (class in geni.rspec.pgad), 55
RS (Namespaces attribute), 53
RSTP (OVSL2STP attribute), 56
rwclone (RemoteBlockstore attribute), 51

## S

saveAggregates() (in module geni.util), 62
SDN (Namespaces attribute), 56
SecureHgMount (class in geni.rspec.vts), 57
Service (class in geni.rspec.pg), 53
setCollocateFactor (class in geni.rspec.emulab.emuext), 49
setDelayImage (class in geni.rspec.emulab.emuext), 49
setDeleteLock() (VTS method), 44
setDHCPSubnet() (VTS method), 44
setFailureAction (class in geni.rspec.emulab.emuext), 49
setForceShaping (class in geni.rspec.emulab.emuext), 49
setImageAttribute() (Image method), 56
setMirror() (OVSImage method), 56

setNoBandwidthShaping (class in geni.rspec.emulab.emuext), 49
setNoInterSwitchLinks (class in geni.rspec.emulab.emuext), 49
setPackingStrategy (class in geni.rspec.emulab.emuext), 49
setPortBehaviour() (VTS method), 44
setPortTrunk() (VTS method), 44
setPortVLAN() (VTS method), 44
setRoutingStyle (class in geni.rspec.emulab.emuext), 49
setUseTypeDefaultImage (class in geni.rspec.emulab.emuext), 49
SFlow (class in geni.rspec.vts), 57
sflow (OVSImage attribute), 56
ShapedLink (class in geni.rspec.emulab.emuext), 49
shared_vlans (Advertisement attribute), 55
SimpleDHCPImage (class in geni.rspec.vts), 57
Site (class in geni.rspec.igext), 51
size (Blockstore attribute), 50
SIZE (ParameterType attribute), 47
size (RemoteBlockstore attribute), 51
Slice() (in module geni.urn), 61
Sliver() (in module geni.urn), 61
sliverstatus() (CloudLabAM method), 32
sliverstatus() (EGCompute method), 34
sliverstatus() (IGCompute method), 35
sliverstatus() (OGCompute method), 37
sliverstatus() (PGCompute method), 38
sliverstatus() (Transit method), 40
sliverstatus() (VTS method), 44
SPLIT_REGEX (Tour attribute), 51
SSLVPNFunction (class in geni.rspec.vts), 57
StitchedLink (class in geni.rspec.pg), 53
StitchedLink.TooManyInterfacesError, 54
StitchedLink.UnknownComponentManagerError, 54
stitchinfo (Advertisement attribute), 55
STP (OVSL2STP attribute), 56
STRING (ParameterType attribute), 47
suppressAutoPrint() (Context method), 46
svcStatus() (HostPOAs method), 40
system_id (OVSL2STP attribute), 57

## T

text (AdLink attribute), 54
text (AdNode attribute), 54
text (Advertisement attribute), 55
text (Manifest attribute), 58
TEXT (Tour attribute), 51
TEXT (Tour.Step attribute), 51
TIMEOUT (HTTP attribute), 45
Tour (class in geni.rspec.igext), 51
TOUR (Namespaces attribute), 53
Tour.Step (class in geni.rspec.igext), 51
toXMLString() (Request method), 53, 57
Transit (class in geni.aggregate.transit), 39
Transit.InvalidRSpecPathError, 39
Transit.UnspecifiedComponentManagerError, 39
trivial_ok (Link attribute), 52
type (GENI attribute), 60
type (OVSL2STP attribute), 57
TYPE_AUTHORITY (GENI attribute), 60
TYPE_IMAGE (GENI attribute), 60
TYPE_INTERFACE (GENI attribute), 60
TYPE_LINK (GENI attribute), 60
TYPE_NODE (GENI attribute), 60
TYPE_SLICE (GENI attribute), 60
TYPE_SLIVER (GENI attribute), 60
TYPE_USER (GENI attribute), 60

## U

UnhandledPortTypeError, 58
UnknownSTPModeError, 57
updateAggregates() (in module geni.util), 62
useDocstring() (Tour method), 51
User() (in module geni.urn), 61

## V

v4RouterPOAs (class in geni.aggregate.vts), 44
verifyParameters() (Context method), 46
VFCircuit (class in geni.rspec.vts), 57
vlan_tagging (Link attribute), 52
VM (in module geni.rspec.pg), 54
VM (NodeType attribute), 53
VTOP (Namespaces attribute), 53
VTS (class in geni.aggregate.vts), 41
VTS (Namespaces attribute), 56
VTS.InvalidRSpecPathError, 41
VTS.UnspecifiedComponentManagerError, 41
VZContainer (class in geni.rspec.pg), 54

## W

write() (Manifest method), 58
writeXML() (Advertisement method), 55
writeXML() (Manifest method), 58
writeXML() (Request method), 53, 57

## X

XEN (NodeType attribute), 53
xen_ptype (XenVM attribute), 51
XenVM (class in geni.rspec.igext), 51
XenVM (class in geni.rspec.pg), 54
xmit_hold_count (OVSL2STP attribute), 57