RIPE Atlas Tools (Magellan) Documentation

Release 2.1

The RIPE Atlas Team

Apr 21, 2017

Contents

1 Why This Exists **3**

2 Contents **5**

2.1 Quickstart 5
2.2 Requirements & Installation 6
2.3 How to Use the RIPE Atlas Toolkit 9
2.4 How to Create Your Own Plugins 17
2.5 How To Contribute 20
2.6 Packaging 21
2.7 Troubleshooting 22
2.8 Release History 23

The official command-line client for RIPE Atlas.

CHAPTER 1

Why This Exists

RIPE Atlas is a powerful Internet measurements platform that until recently was only accessible via the website and the RESTful API. The reality however is that a great many people using RIPE Atlas are most comfortable on the command-line, so this project is an attempt to fill that gap.

CHAPTER 2

Contents

Quickstart

This is a very fast break down of everything you need to start using Ripe Atlas on the command line. Viewing public data is quick & easy, while creation is a little more complicated, since you need to setup your authorisation key.

Viewing Public Data

1. *Install* the toolkit.
2. View help with: `ripe-atlas --help`
3. View a basic report for a public measurement: `ripe-atlas report <measurement_id>`
4. View the live stream for a measurement: `ripe-atlas stream <measurement_id>`
5. Get a list of probes in ASN 3333: `ripe-atlas probe-search --asn 3333`
6. Get a list of measurements with the word "wikipedia" in them: `ripe-atlas measurement-search --search wikipedia`

Creating a Measurement

1. Log into RIPE Atlas. If you don't have an account, you can create one there for free.
2. Visit the API Keys page and create a new key with the permission `Create a new user defined measurement`
3. Install the toolkit as below.
4. Configure the toolkit to use your key with `ripe-atlas configure --set authorisation.create=MY_API_KEY`
5. View the help for measurement creation with `ripe-atlas measure --help`
6. Create a measurement with `ripe-atlas measure ping --target example.com`

Advanced Use

Refer to the *complete usage documentation* for more advanced options.

Requirements & Installation

This is a Linux-based tool, though it should work just fine in a BSD variant. Windows is experimentally supported. In terms of the actual installation, only Python's package manager (`pip`) is currently supported, and the installation process may require some system packages to be installed in order for everything to work.

System Requirements

Some of the dependencies need to be compiled, so you'll need a compiler on your system, as well as the development libraries for Python. In the Linux world, this typically means a few packages need to be installed from your standard package manager, but in true Linux fashion, each distribution does things slightly differently.

The most important thing to know is that you need Python 2.7 or 3. Python 2.6 will never be supported because it's old, ugly, and needs to die.

Distribution Specific Requirements

Note: If you're running OpenBSD, you can skip this whole section. You can even skip the next one too. Just skip down to *Installation:OpenBSD* and follow the instructions. Everything else is taken care of for you.

Debian/Ubuntu

The following has been tested on Debian Jessie.

Debian-based distributions require three system packages to be installed first:

```
sudo apt-get install python-dev libffi-dev libssl-dev
```

You'll also need either `virtualenv` (recommended), or if you're not comfortable with that, at the very least, you'll need `pip`:

```
sudo apt-get install python-virtualenv python-pip
```

CentOS

This following has been tested on CentOS 7.

Since we require Python's `pip`, we first need to install the `epel-release` repository:

```
sudo yum install epel-release
```

You'll also need the following system libraries:

```
sudo yum install gcc libffi-devel openssl-devel
```

Once that's finished, you'll need access to `virtualenv` (recommended), or if you're not comfortable with that, at the very least, you'll need `pip`:

```
sudo yum install python-virtualenv python-pip
```

Gentoo

If you're a Gentoo user, you never have to worry about development libraries, but if you intend to use the bleeding-edge version of this package (and what self-respecting Gentoo user wouldn't?) then you'll probably want to make sure that git is built with curl support:

```
sudo USE="curl" emerge git
```

If you're not going bleeding edge, or if you're just going to use SSH to get the code from GitHub, then Gentoo will have everything ready for you.

Apple OSX

These instructions expect that you've got Python's `pip` installed, so if you have no idea what that is, or simply don't have it yet, you should be able to install pip with one easy command:

```
sudo easy_install pip
```

Outside of that, a few of the Python dependencies require that you have a compiler on your system. For this, you need only get a free copy of Xcode from the app store, and from there you should be good to go.

Python Requirements

Importantly, Magellan requires Python 2.7 or higher. For most desktop users, this shouldn't be a problem, but for some older servers like CentOS 6 and lower, this may cause some pain. Thankfully, for most such systems, there are usually work-arounds that allow you to install a more modern version of Python in parallel.

Magellan depends on two other RIPE Atlas libraries, Cousteau and Sagan, which in turn depend on a reasonable number of Python libraries. Thankfully, Python's package manager, `pip` should handle all of these for you:

- ripe.atlas.cousteau
- ripe.atlas.sagan
- tzlocal
- pyyaml

Installation

OpenBSD

OpenBSD was the first platform to have a port for Magellan, so installation is easy:

```
sudo pkg_add py-ripe.atlas.tools
```

FreeBSD

FreeBSD has a port ready for you:

cd /usr/ports/net/py-ripe.atlas.tools make install

Gentoo

There's an ebuild for Magellan in Portage, so installation is as any other package:

```
sudo emerge ripe-atlas-tools
```

From PyPi

Python's `pip` program can be used to install packages globally (not a good idea since it conflicts with your system package manager) or on a per-user basis. Typically, this is done with virtualenv, but if you don't want to use that, you can always pass `--user` to the `pip` program and it'll install a user-based copy in `${HOME}/.local/`.

```
# From within a virtualenv
pip install ripe.atlas.tools

# In your user's local environment
pip install --user ripe.atlas.tools
```

Or if you want to live on the edge and perhaps try submitting a pull request of your own:

One day, we want this process to be as easy as installing any other command-line program, that is, with `apt`, `dfn`, or `emerge`, but until that day, Python's standard package manager, `pip` does the job nicely.

From GitHub

If you're feeling a little more daring and want to go bleeding-edge and use our `master` branch on GitHub, you can have pip install right from there:

```
pip install git+https://github.com/RIPE-NCC/ripe-atlas-tools.git
```

If you think you'd like to contribute back to the project, we recommend the use of pip's `-e` flag, which will place the Magellan code in a directory where you can edit it, and see the results without having to go through a new install procedure every time. Simply clone the repo on GitHub and install it like so:

```
pip install -e git+https://github.com/your-username/ripe-atlas-tools.git
```

From a Tarball

If for some reason you want to just download the source and install it manually, you can always do that too. Simply un-tar the file and run the following in the same directory as `setup.py`:

```
python setup.py install
```

How to Use the RIPE Atlas Toolkit

Configuration

For most features, Magellan will work out-of-the-box, but if you'd like to customise the experience, or if you want to use this tool to create a measurement of your own, then you'll need to configure it.

Thankfully, configuration is easy by way of the `configure` command::

```
$ ripe-atlas configure --help
```

Options

Option	Arguments	Explanation
`--editor`		Invoke ${EDITOR} to edit the configuration directly
`--set`	path=value	Permanently set a configuration value so it can be used in the future.
`--init`		Create a configuration file and save it into your home directory at: `${HOME}/.config/ripe-atlas-tools/rc`

Examples

Create a standard configuration file. Note that this typically isn't necessary:

```
$ ripe-atlas configure --init
```

Invoke your editor of choice to manually fiddle with the configuration file:

```
$ ripe-atlas configure --editor
```

Set an arbitrary value within the configuration file. You can use dot-separated notation to dictation the value you wish to change:

```
$ ripe-atlas configure --set authorisation.create=YOUR_API_KEY
```

Quick Measurement Information

For the impatient, and for those looking to see how they might write their own plugins, we have a simple `go` command::

```
$ ripe-atlas go <measurement-id>
```

This will open a web browser and take you to the detail page for the measurement id provided.

Measurement Querying

A querying tool for finding existing measurements in the RIPE Atlas database. You can request a table-formatted list of measurements based on search-string lookups, type, start time, etc.

Options

Option	Arguments	Explanation
`--search`	A free-form string	This could match the target or description.
`--status`	One of: scheduled, stopped, ongoing	The measurement status.
`--af`	One of: 4, 6	The address family.
`--type`	One of: ping, traceroute, dns, sslcert, ntp, http	The measurement type.
`--field`	One of: status, target, url, type, id, description	The field(s) to display. Invoke multiple times for multiple fields. The default is id, type, description, and status.
`--ids-only`		Display a list of measurement ids matching your filter criteria.
`--limit`	An integer	The number of measurements to return. The number must be between 1 and 1000
`--started-before`	An ISO timestamp	Filter for measurements that started before a specific date. The format required is YYYY-MM-DDTHH:MM:SS
`--started-after`	An ISO timestamp	Filter for measurements that started after a specific date. The format required is YYYY-MM-DDTHH:MM:SS
`--stopped-before`	An ISO timestamp	Filter for measurements that stopped before a specific date. The format required is YYYY-MM-DDTHH:MM:SS
`--stopped-after`	An ISO timestamp	Filter for measurements that stopped after a specific date. The format required is YYYY-MM-DDTHH:MM:SS

Examples

Get a list of measurements:

```
$ ripe-atlas measurement-search
```

Filter that list by `status=ongoing`:

```
$ ripe-atlas measurement-search --status ongoing
```

Further filter it by getting measurements that conform to IPv6:

```
$ ripe-atlas measurement-search --status ongoing --af 6
```

Get that same list, but strip out everything but the measurement ids:

```
$ ripe-atlas measurement-search --status ongoing --af 6 --ids-only
```

Limit that list to 200 entries:

```
$ ripe-atlas measurement-search --status ongoing --af 6 --limit 200
```

Get that list, but show only the id, url and target fields:

$ ripe-atlas measurement-search –status ongoing –af 6 –field id –field url –field target

Filter for measurements of type `dns` that started after January 1, 2015:

```
$ ripe-atlas measurement-search --type dns --started-after 2015-01-01
```

Probe Querying

Just like the `measurement-search` command, but for probes, and a lot more powerful. You can use this command to find probes within an ASN, prefix, or geographical region, and then aggregate by country, ASN, and/or prefix.

Options

Option	Arguments	Explanation
`--limit`	An integer	Return limited number of probes.
`--field`	One of: status, description, address_v6, address_v4, asn_v4, is_public, asn_v6, id, prefix_v4, prefix_v6, is_anchor, country, coordinates	The field(s) to display. Invoke multiple times for multiple fields. The default is id, asn_v4, asn_v6, country, and status.
`--aggregate-by`	country, asn_v4, asn_v6, prefix_v4, prefix_v6	Aggregate list of probes based on all specified aggregations. Multiple aggregations supported.
`--all`		Fetch *ALL* probes. That will give you a loooong list.
`--max-per-aggregation`	An integer	Maximum number of probes per aggregated bucket.
`--ids-only`		Print only IDs of probes. Useful to pipe it to another command.
`--asn`	An integer	Filter the list by an ASN
`--asnv4`	An integer	Filter the list by an ASN
`--asnv6`	An integer	Filter the list by an ASN
`--prefix`	A prefix string	Filter the list by a prefix
`--prefixv4`	A prefix string	Filter the list by a prefix
`--prefixv6`	A prefix string	Filter the list by a prefix
`--location`	A free-form string	The location of probes as a string i.e. 'Amsterdam'
`--center`	A pair of geographic coordinates	Location as <lat>,<lon>-string, i.e. "48.45,9.16"
`--radius`	An integer	Radius in km from specified center/point.
`--country`	A two-letter ISO country code	The country in which the probes are located.

Examples

Get a list of probes within ASN 3333:

```
$ ripe-atlas probe-search --asn 3333
```

Further filter that list to show only probes in ASN 3333 from the Netherlands:

```
$ ripe-atlas probe-search --asn 3333 --country nl
```

Change the limit from the default of 25 to 200:

```
$ ripe-atlas probe-search --asn 3333 --limit 200
```

Aggregate the probes by country, and then by ASN:

```
$ ripe-atlas probe-search --asn 3333 --aggregate-by country --aggregate-by asn
```

Show the id, url, target, description, and whether the probe is public or not:

```
$ ripe-atlas probe-search --asn 3333 --field id --field url --field description \
  --field is_public
```

Result Reporting

A means to generate a simple text-based report based on the results from a measurement. Typically, this is used to get the latest results of a measurement in a human-readable format, but with the `--start-time` and `--stop-time` options, you can get results from any time range you like. It's possible to generate the report by automatically fetching the results from the API, by reading a local file, or by reading standard input.

Options

Option	Arguments	Explanation
`--auth`	RIPE Atlas key alias	One of the RIPE Atlas key alias configured for results fetching.
`--probes`	A comma-separated list of probe ids	Limit the report to only results obtained from specific probes.
`--probe-asns`	A comma-separated list of ASNs	Limit the report to only results obtained from probes belonging to specific ASNs.
`--renderer`	One of: dns, http, ntp, ping, raw, ssl_consistency, sslcert, traceroute, traceroute_aspath, aggregate_ping	The renderer you want to use. If this isn't defined, an appropriate renderer will be selected.
`--from-file`	A file path	The source of the data to be rendered. Conflicts with specifying a measurement_id to fetch from the API.
`--aggregate-by`	One of: status, prefix_v4, prefix_v6, country, rtt-median, asn_v4, asn_v6	Tell the rendering engine to aggregate the results by the selected option. Note that if you opt for aggregation, no output will be generated until all results are received.
`--start-time`	An ISO timestamp	The start time of the report. The format should conform to YYYY-MM-DDTHH:MM:SS
`--stop-time`	An ISO timestamp	The stop time of the report. The format should conform to YYYY-MM-DDTHH:MM:SS

Examples

Get the latest results of measurement 1001:

```
$ ripe-atlas report 1001
```

The same, but specifically request the ping renderer:

```
$ ripe-atlas report 1001 --renderer ping
```

Aggregate those results by country:

```
$ ripe-atlas report 1001 --aggregate-by country
```

Get results from the same measurement, but show all results from the first week of 2015:

```
$ ripe-atlas report 1001 --start-time 2015-01-01 --stop-time 2015-01-07
```

Get results from the first day of 2015 until right now:

```
$ ripe-atlas report 1001 --start-time 2015-01-01
```

Pipe the contents of an arbitrary file into the renderer. The rendering engine will be guessed from the first line of input:

```
$ cat /path/to/file/full/of/results | ripe-atlas report
```

The same, but point Magellan to a file deliberately rather than using a pipe:

```
$ ripe-atlas report --from-file /path/to/file/full/of/results
```

Result Streaming

Connect to the streaming API and render the results in real-time as they come in.

Options

Option	Arguments	Explanation
`--auth`	RIPE Atlas key alias	One of the RIPE Atlas key alias configured for results fetching.
`--limit`	A number < 1000	The maximum number of results you want to stream. The default is to stream forever until you hit `Ctrl+C`.
`--renderer`	One of: dns, http, ntp, ping, raw, ssl_consistency, sslcert, traceroute, traceroute_aspath, aggregate_ping	The renderer you want to use. If this isn't defined, an appropriate renderer will be selected.

Examples

Stream the results from measurement #1001:

```
$ ripe-atlas stream 1001
```

Limit those results to 500:

```
$ ripe-atlas stream 1001 --limit 500
```

Specify a renderer:

```
$ ripe-atlas stream 1001 --renderer ping
```

Combine for fun and profit:

```
$ ripe-atlas stream 1001 --renderer ping --limit 500
```

Measurement Creation

The most complicated command we have, this will create a measurement (given a plethora of options) and begin streaming the results back to you in a standardised rendered form.

It's invoked by using a special positional argument that dictates the type of measurement you want to create. This also unlocks special options, specific to that type. See the *examples* for more information.

Options

All measurements share a base set of options.

Option	Arguments	Explanation
`--renderer`	One of: dns, http, ntp, ping, raw, ssl_consistency, sslcert, traceroute, traceroute_aspath, aggregate_ping	The renderer you want to use. If this isn't defined, an appropriate renderer will be selected.
`--dry-run`		Do not create the measurement, only show its definition.
`--auth`	An API key	The API key you want to use to create the measurement.
`--af`	One of: 4, 6	The address family, either 4 or 6. The default is a guess based on the target, favouring 6.
`--description`	A free-form string	The description/name of your new measurement.
`--target`	A domain or IP	The target, either a domain name or IP address. If creating a DNS measurement, the absence of this option will imply that you wish to use the probe's resolver.
`--no-report`		Don't wait for a response from the measurement, just return the URL at which you can later get information about the measurement.
`--interval`	An integer	Rather than run this measurement as a one-off (the default), create this measurement as a recurring one, with an interval of n seconds between attempted measurements. This option implies `--no-report`.
`--from-area`	One of: WW, West, North-Central, South-Central, North-East, South-East	The area from which you'd like to select your probes.
`--from-country`	A two-letter ISO country code	The country from which you'd like to select your probes.
`--from-prefix`	A prefix string	The prefix from which you'd like to select your probes.
`--from-asn`	An ASN number	The ASN from which you'd like to select your probes.
`--from-probes`	A comma-separated list of probe ids	Probes you want to use in your measurement.
`--from-measurement`	A measurement id	A measurement id which you want to use as the basis for probe selection in your new measurement. This is a handy way to re-create a measurement under conditions similar to another measurement.
`--probes`	An integer	The number of probes you want to use.
`--include-tag`	A tag name	Include only probes that are marked with this tag. Note that this option may be repeated.
`--exclude-tag`	A tag name	Exclude probes that are marked with this tag. Note that this option may be repeated.

Ping-Specific Options

Option	Arguments	Explanation
`--packets`	An integer	The number of packets sent
`--size`	An integer	The size of packets sent
`--packet-interval`	An integer	

Traceroute-Specific Options

Option	Arguments	Explanation
`--packets`	An integer	The number of packets sent
`--size`	An integer	The size of packets sent
`--protocol`	One of: ICMP, UDP, TCP	The protocol used. For DNS measurements, this is limited to UDP and TCP, but traceroutes may use ICMP as well.
`--timeout`	An integer	The timeout per-packet
`--dont-fragment`		Don't Fragment the packet
`--paris`	An integer	Use Paris. Value must be between 0 and 64.If 0, a standard traceroute will be performed.
`--first-hop`	An integer	Value must be between 1 and 255.
`--max-hops`	An integer	Value must be between 1 and 255.
`--port`	An integer	Destination port, valid for TCP only.
`--destination-option-size`	An integer	IPv6 destination option header.
`--hop-by-hop-option-size`	An integer	IPv6 hop by hop option header.

DNS-Specific Options

Option	Arguments	Explanation
`--query-class`	One of: IN, CHAOS	The query class. The default is "IN"
`--query-type`	One of: A, SOA, TXT, SRV, SSHFP, TLSA, NSEC, DS, AAAA, CNAME, DNSKEY, NSEC3, PTR, HINFO, NSEC3PARAM, NS, MX, RRSIG, ANY	The query type. The default is "A"
`--query-argument`	A string	The DNS label to query.
`--set-cd-bit`		Set the DNSSEC Checking Disabled flag (RFC4035)
`--set-do-bit`		Set the DNSSEC OK flag (RFC3225)
`--set-nsid-bit`		Include an EDNS name server. ID request with the query.
`--udp-payload-size`	An integer	May be any integer between 512 and 4096 inclusive.
`--set-rd-bit`		Set the Recursion Desired flag.
`--retry`	An integer	Number of times to retry.

SSL Certificate-Specific Options

Option	Arguments	Explanation
`--port`	An integer	The port to query

HTTP-Specific Options

Option	Arguments	Explanation
`--header-bytes`	An integer	The maximum number of bytes to retrieve from the header
`--version`	A string	The HTTP version to use
`--method`	A string	The HTTP method to use
`--path`	A string	The path on the webserver
`--query-string`	A string	An arbitrary query string
`--user-agent`	A string	An arbitrary user agent
`--body-bytes`	An integer	The maximum number of bytes to retrieve from the body
`--timing-verbosity`	One of: 0, 1, 2	The amount of timing information you want returned. 1 returns the time to read, to connect, and to first byte, 2 returns timing information per read system call. 0 (default) returns no additional timing information.

NTP-Specific Options

Option	Arguments	Explanation
`--packets`	An integer	The number of packets sent
`--timeout`	An integer	The timeout per-packet

Examples

The simplest of measurements. Create a ping with 50 probes to example.com:

```
$ ripe-atlas measure ping --target example.com
```

The same, but don't actually create it, just show what would be done:

```
$ ripe-atlas measure ping --target example.com --dry-run
```

Be more specific about which address family you want to target:

```
$ ripe-atlas measure ping --target example.com --af 6
```

Ask for 20 probes from Canada:

```
$ ripe-atlas measure ping --target example.com --probes 20 --from-country ca
```

Or ask for 20 Canadian probes that definitely support IPv6:

```
$ ripe-atlas measure ping --target example.com --probes 20 \
  --from-country ca --include-tag system-ipv6-works
```

Rather than creating a one-off create a recurring measurement:

```
$ ripe-atlas measure ping --target example.com --interval 3600
```

Moving onto DNS measurements, do a lookup for example.com. Since we're not specifying `--target` here, this implies that we want to use the probe's resolver:

```
$ ripe-atlas measure dns --query-argument example.com
```

Getting a little more complicated, let's set a few special bits and make a more complex query:

```
$ ripe-atlas measure dns --query-type AAAA --query-argument example.com \
  --set-nsid-bit --set-rd-bit --set-do-bit --set-cd-bit
```

Shortcuts

If you're creating a lot of measurements in a short time, typing out `ripe-atlas measure traceroute` a whole bunch of times can be tiresome, so we've added a few shortcut scripts for you:

Where you'd typically write	You could use this instead
`ripe-atlas measure ping`	`aping`
`ripe-atlas measure traceroute`	`atraceroute`
`ripe-atlas measure dns`	`adig`
`ripe-atlas measure sslcert`	`asslcert`
`ripe-atlas measure http`	`ahttp`
`ripe-atlas measure ntp`	`antp`

So for example, these two commands are the same:

```
$ ripe-atlas measure ping --target example.com
$ aping --target example.com
```

If you want to streamline your typing process even more than this, we recommend the use of your shell's `alias` feature, which is both powerful and customisable for your needs.

How to Create Your Own Plugins

We built this toolkit for the community, and we knew going in that we couldn't possibly build every feature that every user could want, so we built this thing to be pluggable. You can write your own renderer(s) and use them seamlessly within your own environment, and if you think that others might benefit from your work, you can share your renderer as easy as posting a file online.

Ready?

So you have an idea now. You want to create a renderer called "awesomerenderer" and you want it to do some fancy things with traceroute measurement results. What do you have to do?

Create Your Renderer File

As we've already covered, Magellan will look for renderers in very specific places, so you need to put your file(s) there. Additionally however, you have to make sure that you conform to Python norms, or stuff just won't work. Here's the basic commands to get you started:

```
$ mkdir -p ${HOME}/.config/ripe-atlas-tools/renderers
$ touch ${HOME}/.config/ripe-atlas-tools/renderers/__init__.py
$ touch ${HOME}/.config/ripe-atlas-tools/renderers/my_renderer.py
```

The `mkdir` step there will create the renderers directory (if it doesn't exist already), and the `touch` commands will create the mandatory init file (for Python) and your renderer. Note that you can use whatever name you like for your renderer, so long as it consists only of letters, numbers, and the underscore and that it starts with a letter. Also, to be compliant with the rest of the project, it should be entirely lowercase. For our purposes though, `my_renderer.py` will suffice.

(Try to) Run It!

If you run this right now:

```
$ ripe-atlas report --help
```

You should see `my_renderer` int he list of options for `--renderer`. Pretty cool eh? However, if you try to run that, this'll happen:

```
$ ripe-atlas report 1000192 --renderer my_renderer
The renderer you selected, "my_renderer" could not be found.
```

Which kind of makes sense really. You've created a file called `my_renderer`, but it's totally empty. Magellan found the file alright, but when it tried to import `Renderer` from it, everything exploded.

Actually Write a Renderer

So now you know that we can see your renderer file, but you need to know what kind of code to put in there. Don't worry, we've got you covered:

Anatomy of a Renderer

A "renderer" is simply a file located in a special place that contains some Python code defining a class called `Renderer` that subclasses `ripe.atlas.tools.renderers.base.BaseRenderer`.

Your class need only define one method: `on_result()`, which is called every time a new result comes down the pipe. Let's look at a really simple example:

```
from ripe.atlas.tools.renderers.base import Renderer as BaseRenderer


class Renderer(BaseRenderer):

    # This renderer is capable of handling ping results only.
    RENDERS = [BaseRenderer.TYPE_PING]

    def on_result(self, result):
        """
        on_result() only gets one argument, a result object, which is
        actually an instance of a RIPE Atlas result parsed with Sagan:
          https://ripe-atlas-sagan.readthedocs.org/
        """

        return "Packets received: {}".format(result.packets_received)
```

As you can see, this renderer isn't very useful, but we're providing it here to give you a rough idea of what you get to play with when defining your own renderer.

In the case of our PingPacketRenderer, we're doing the simplest of tasks: we're returning the number of packets in each result. The job of `on_result()` is to take a Sagan result object as input and return a string. **It should not print anything to standard out**, rather it should simply return a string that will get printed to standard out by the surrounding framework.

Additional Options

It's likely that you will only ever need to work with `on_result()`, but in the event that you'd like to get more complicated, there are options: `header()`, `additional()`, and `footer()`. Note however that these other methods are currently only available to the `report` command. Streaming only makes use of `on_result()`.

header()

The value returned from this method is printed to standard out before any results are captured. By default it returns an empty string.

additional()

Typically used for summary logic, this is executed after the last result is rendered. A common pattern is to override `__init__()` to set some collector properties, update them via `on_result()`, and then print out said properties in a summary via this method. For an example, let's update our `Renderer` class:

```
from ripe.atlas.tools.renderers.base import Renderer as BaseRenderer


class Renderer(BaseRenderer):

    RENDERS = [BaseRenderer.TYPE_PING]

    def __init__(self, *args, **kwargs):
        self.packet_total = 0
        BaseRenderer.__init__(self, *args, **kwargs)

    def on_result(self, result):
        self.packet_total += result.packets_received
        return "Packets received: {}\n".format(result.packets_received)

    def additional(self, results):
        return "\nTotal packets received: {}\n".format(self.packet_total)
```

Note that the passed-in value of `results` is the list of Sagan Result objects that were previously looped over for on_result(). You can do some interesting things with that.

footer()

Much the same as `header()`, this should return a string, but unlike `header()`, the output of this method is rendered after everything else.

Run It!

Now that you've written your renderer and the file is stored where it's supposed to be, it should be ready to go:

```
$ ripe-atlas report --help
```

You should see `my_renderer` in the list of options for `--renderer` just as before, but now when you actually try to execute it...

```
$ ripe-atlas report 1000192 --renderer my_renderer
Packets received: 3
Packets received: 3
Packets received: 3
Packets received: 3
Packets received: 3
Packets received: 3

Total packets received: 18
```

It's not very interesting, but it's a start!

Contributing

We love it when people write stuff that talks to our stuff. If you think your stuff is useful, it'd be awesome if you could do any of these:

- Post to the [ripe-atlas mailing list](https://www.ripe.net/mailman/listinfo/ripe-atlas) about it. You can also solicit feedback from the RIPE Atlas developers or the wider community on this list.
- Write a blog post about your plugin, what makes it useful, etc.
- Tweet about it. Feel free to mention [@RIPE_Atlas](https://twitter.com/ripe_atlas) and we might even retweet it.
- Create a [pull request](https://github.com/RIPE-NCC/ripe-atlas-tools/pulls) for this project to get your plugin added to core.

How To Contribute

We would love to have contributions from everyone and no contribution is too small. Please submit as many fixes for typos and grammar bloopers as you can!

To make participation in this project as pleasant as possible for everyone, we adhere to the Code of Conduct by the Python Software Foundation.

The following steps will help you get started:

Fork, then clone the repo:

```
$ git clone git@github.com:your-username/ripe-atlas-tools.git
```

Make sure the tests pass beforehand:

```
$ tox
```

or

```
$ nosetests tests/
```

Make your changes. Include tests for your change. Make the tests pass:

```
$ tox
```

or

```
$ nosetests tests/
```

Push to your fork and submit a pull request.

Here are a few guidelines that will increase the chances of a quick merge of your pull request:

- *Always* try to add tests and docs for your code. If a feature is tested and documented, it's easier for us to merge it.
- Follow PEP 8.
- Write good commit messages.
- If you change something that is noteworthy, don't forget to add an entry to the changes.

Note:

- If you think you have a great contribution but aren't sure whether it adheres – or even can adhere – to the rules: **please submit a pull request anyway**! In the best case, we can transform it into something usable, in the worst case the pull request gets politely closed. There's absolutely nothing to fear.
- If you have a great idea but you don't know how or don't have the time to implement it, please consider opening an issue and someone will pick it up as soon as possible.

Thank you for considering a contribution to this project! If you have any questions or concerns, feel free to reach out the RIPE Atlas team via the mailing list, GitHub Issue Queue, or messenger pigeon – if you must.

Packaging

For those interested in packaging RIPE Atlas Tools for their favourite distro, this section is for you.

Currently Supported

- OpenBSD
- FreeBSD
- Gentoo
- Debian
- Ubuntu

In Progress

- Fedora: Jan Včelák is currently building the binary packages in COPR (which will take some time as there is a lot of other packages in the queue)

Additional Distributions

Is your distribution not listed? If you'd like to build a package for another distro or even if you're just someone who knows someone who can help us package and distribute this, please get in touch.

Further Information

User Agent

When packaging, it's good practise to manually set the user agent used within the toolkit so that we can get a rough idea of which distros are using this software. This is easily done by writing an arbitrary string to `<root>/ripe/atlas/tools/user-agent`. Something like this is recommended::

```
RIPE Atlas Tools (Magellan) [FreeBSD 10.2] 1.2
```

The only limitations to this file are that it should:

- Only have one line in it (all other will be ignored)
- That line should have a maximum of 128 characters in it

Troubleshooting

Sometimes things don't go as planned. In these cases, this page is here to help.

InsecurePlatformWarning

On older systems (running Python versions <2.7.10), you may be presented with a warning message that looks like this:

```
/path/to/lib/python2.7/site-packages/requests/packages/urllib3/util/ssl_.py:100:
InsecurePlatformWarning: A true SSLContext object is not available. This
prevents urllib3 from configuring SSL appropriately and may cause certain
SSL connections to fail. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#insecureplatformwarning.
  InsecurePlatformWarning
```

This is due to the insecure way older versions of Python handle secure connections and a visit to the above URL will tell you that the fix is one of three options:

- Upgrade to a modern version of Python
- Install three Python packages: `pyopenssl`, `ndg-httpsclient`, and `pyasn1`
- Suppress the warnings. Don't do that though.

Sagan, OpenSSL, and OSX

If you're using Mac OSX, the installation of Sagan, (one of Magellan's dependencies) may give you trouble, especially in how Apple handles PyOpenSSL on their machines. Workarounds and proper fixes for this issue can be found in the Sagan installation documentation.

Complaints from libyaml

During the installation, you may see something like this scroll by:

> **Running setup.py install for pyyaml** checking if libyaml is compilable x86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fPIC -I/usr/include/python2.7 -c build/temp.linux-x86_64-2.7/check_libyaml.c -o build/temp.linux-x86_64-2.7/check_libyaml.o build/temp.linux-x86_64-2.7/check_libyaml.c:2:18: fatal error: yaml.h: No such file or directory
>
> > **#include <yaml.h>** ^
> >
> > compilation terminated.
> >
> > libyaml is not found or a compiler error: forcing –without-libyaml (if libyaml is installed correctly, you may need to
> >
> > > specify the option –include-dirs or uncomment and modify the parameter include_dirs in setup.cfg)

Don't worry. This is just the installation script noticing that you don't have libyaml installed and it's complaining because it's good to have around for performance reasons. However, since we're only using YAML for configuration, performance isn't an issue, and the fallback option will be sufficient.

If however, you don't like these sorts of errors, make sure that libyaml is installed for your distribution before attempting to install this toolkit.

Release History

2.1 (released 2016-04-21)

New Features

- Add a simple NTP renderer

Changes

- Use new cousteau (1.4) & sagan(1.2) versions.

Bug Fixes

- Fix for some unicode problems when using colors
- Fix issue #177, with *gdbm* problem.

2.0.2 (released 2016-10-21)

New Features

- Add aliases to measurements IDs
- Add –traceroute-show-asns to traceroute renderer

Bug Fixes

- Stream command was not passing the correct API key. After API became stricter this command started failing.
- Handle missing geometry for probes.
- Fix issues for AS-paths with only 1 probe
- Various fixes for tests

2.0.1 (released 2016-04-20)

Changes

- Corrected references in the docs to obsolete command names.
- Fixed broken 2.0.0 egg.

2.0.0 (released 2016-04-20)

Changes

- Renamed and merged some commands for clarity, preserving the old names as deprecated aliases.
- Improved help text and usage output.
- Support for bash auto-completion.

1.2.3 (released 2016-03-08)

Changes

- Usage of newest Cousteau/Sagan library.
- Support of API keys for fetching results on report command.
- Default radius for probes filtering is changed to 15.
- Several changes for supporting Windows.

1.2.2 (released 2016-01-13)

New Features

- Cleaner and more consistent implementation of the renderer plugable architecture.
- Usage of newest Cousteau library.

1.2.1 (released 2015-12-15)

Bug Fixes

- Restored some required template files.

1.2.0 (released 2015-12-15)

Output Changes

- #119: Support HTTP results.
- #122: Allow packagers to set the user agent.

1.1.1 (released 2015-11-25)

Output Changes

- #103: Removed header from the `report` command.

Bug Fixes

- #105: Measurement report and stream broken on Python3.4.

1.1.0 (released 2015-11-12)

New features

- Support for the creation of NTP, SSLCert, and HTTP measurements.
- Additional argument in report command to filter results by probe ASN.
- Additional renderer that shows the different destination ASNs and some additional stats about them.

Bug Fixes

- Various fixes.

Changes

- Better testing.
- Additional documentation.

1.0.0 (released 2015-11-02)

- Initial release.