infoset-ng Documentation

Release 0.0

Peter Harrison

Jun 04, 2018

Contents:

`infoset-ng` is a lightweight Python 3 based REST API that stores and/or retrieves timestamped data. It runs on Linux.

We recommend that you read as much as you can on installation, configuration, operation and the API to get a good understanding of the software.

The introduction will provide useful background information too.

CHAPTER 1

Introduction

There's a lot to know about `infoset-ng` which we'll summarize here.

1.1 Features

`infoset-ng` has the following features:

1. Open source.
2. Written in python, a modern language.
3. Easy configuration.
4. Uses the well known Flask webserver for accepting data and responding to requests.
5. `infoset-ng` has a number of fault tolerant features aimed at making it resilient in unstable computing environemnts.
6. MariaDB / MySQL database backend
7. Database connection pooling to reduce database load.
8. Ingestion of data supports parallel multiprocessing for maximum speed.
9. The infoset-ng-ng API server can tolerate the loss of communication with its database by caching the data locally until the database returns online.
10. The `infoset-ng` configuration is entirely stored in files. This allows it to collect data in the absense of a database, such as during maintenance or an outage.
11. Backups are simple. Just save the entire contents of the `infoset-ng` directory tree including hidden files, and save a copy of the database for your performance data.

We are always looking for more contributors!

1.2 Inspiration / History

The `infoset-ng` project originally took inspiration from the SourceForge based `switchmap` project. `switchmap` was written in PERL and designed to create tabular representations of network topologies. Early versions of `infoset` eventually had expanded features which included the polling of network devices for real time performance data. This data was presented via a web interface. The code became cumbersome and the original `infoset` was split into three componet parts.

1. `infoset-ng`: An API for storing and retrieving real time data.
2. `garnet`: A network / server performance charting web application that uses various types of agents for collecting real time data. `garnet` uses `infoset-ng` to store its data.
3. `switchmap-ng` A python 3 based feature equivalent version of `switchmap`.

Each of these projects resides on the University of the West Indies Computing Society's GitHub account.

1.3 Oversight

`infoset-ng` is a student collaboration between:

1. The University of the West Indies Computing Society. (Kingston, Jamaica)
2. The University of Techology, IEEE Student Branch. (Kingston, Jamaica)
3. The Palisadoes Foundation http://www.palisadoes.org

And many others.

CHAPTER 2

Installation

This section outlines how to install and do basic configuration of `infoset-ng`.

2.1 Dependencies

`infoset-ng` has the following requirements:

- python >= 3.5
- python3-pip
- MySQL >= 5.7 OR MariaDB >= 10.0

It will not work with lower versions.

2.1.1 Ubuntu / Debian / Mint

The commands for installing the dependencies are:

```
$ sudo apt-get -y install python3 python3-pip python3-dev memcached
```

Select either of these commands to install MySQL server or MariaDB server

```
$ sudo apt-get -y install mysql-server
$ sudo apt-get -y install mariadb-server
```

2.1.2 Centos / Fedora

The commands for installing the dependencies are:

```
$ sudo dnf -y install python3 python3-pip python3-dev memcached
```

Select either of these commands to install MySQL server or MariaDB server

```
$ sudo dnf -y install mysql-server
$ sudo dnf -y install mariadb-server
```

2.2 Installation

Installation is simple. There are three basic steps.

1. Clone the Repository
2. Setup the Database
3. Run the Installation Script

This will now be explained in more detail.

2.2.1 Clone the Repository

Now clone the repository and copy the sample configuration file to its final location.

```
$ git clone https://github.com/PalisadoesFoundation/infoset-ng
$ cd infoset-ng
```

2.2.2 Setup the Database

Next create the MySQL or MariaDB database. Make sure the database server is running.

```
$ mysql -u root -p
password:
mysql> create database infoset_ng;
mysql> grant all privileges on infoset_ng.* to infoset_ng@"localhost" identified by
↪'PASSWORD';
mysql> flush privileges;
mysql> exit;
```

Note Remember the value you select for `PASSWORD`. It will be required when you edit the `infoset-ng` configuration file later.

2.2.3 Run Installation Script

Run the installation script. There are two alternatives:

Installing as a regular user

There are some things to keep in mind when installing *switchmap-ng* as a regular user.

1. Use this method if you don't have `root` access to your system.
2. The `switchmap-ng` daemons *will not* automatically restart on reboot using this method.

To make `switchmap-ng` run with your username, then execute this command:

```
$ maintenance/install.py
```

Installing as the "root" user

There are some things to keep in mind when installing *switchmap-ng* as the *root* user.

1. The `switchmap-ng` daemons *will* automatically restart on reboot using this installation method.
2. **Note**: Do not run setup using `sudo`. Use `sudo` to become the `root` user first.

To install `switchmap-ng` as the `root` user execute this command:

```
# maintenance/install.py
```

2.3 Next Steps

It is now time to review the various configuration options.

CHAPTER 3

Configuration

It is important to have a valid configuration file in the `etc/` directory before starting data collection. The installation automatically creates a default version that may need to be edited. This page explains the various configuration parameters.

The `examples/etc` directory includes a sample reference file.

3.1 infoset-ng Configuration Example

In this example we explain each parameter in the configuration file.

The `main` section governs the general operation of `infoset-ng`.

```
main:
    log_directory: /opt/infoset/log
    log_level: debug
    ingest_cache_directory: /opt/infoset/cache
    ingest_pool_size: 20
    interval: 300
    listen_address: 0.0.0.0
    bind_port: 6000
    sqlalchemy_pool_size: 10
    sqlalchemy_max_overflow: 10
    memcached_hostname: localhost
    memcached_port: 11211
    db_hostname: localhost
    db_username: infoset_ng
    db_password: PASSWORD
    db_name: infoset_ng
    username: USERNAME
```

An explanation of these fields follows:

Parameter	Description
`main:`	YAML key describing the server configuration.
`log_directory:`	The directory where `infoset-ng` places its log files
`log_level:`	Defines the logging level. `debug` level is the most verbose, followed by `info`, `warning` and `critical`
`ingest_cache_directory:`	Location where the agent data ingester will store its data in the event it cannot communicate with either the database or the server's API
`ingest_pool_size:`	The maximum number of threads used to ingest data into the database
`interval:`	The expected interval in seconds between updates to the database from systems posting to the infoset API. Data retieved from the API will be spaced `interval` seconds apart.
`listen_address:`	IP address the API will be using. The default is `0.0.0.0` or all available IP addresses
`bind_port:`	The TCP port the API will be listening on
`sqlalchemy_pool_size:`	The SQLAlchemy pool size. This is the largest number of connections that `infoset-ng` will be keep persistently with the MySQL database
`sqlalchemy_max_overflow:`	The SQLAlchemy maximum overflow size. When the number of connections reaches the size set in `sqlalchemy_pool_size`, additional connections will be returned up to this limit. This is the floating number of additional database connections to be made available.
`memcached_hostname:` `localhost`	The hostname of our `memcached` cache server
`memcached_port:` `11211`	The port which `memcached` is running on
`db_hostname:`	The devicename or IP address of the database server.
`db_username:`	The database username
`db_password:`	The database password
`db_name:`	The name of the database
`username:`	The username that scripts should run as

3.2 Logrotate Configuration

The `examples/linux/logrotate/infoset-ng` file is a working logrotate configuration to rotate the log files that `infoset-ng` generates. The `infoset-ng` log file data can be extensive and adding the logrotate file to your system is highly recommended.

```
$ sudo cp examples/linux/logrotate/infoset-ng /etc/logrotate.d
```

3.3 Next Steps

It is time to test the operation of `infoset-ng`.

CHAPTER 4

Advanced Operation

The `switchmap-ng` CLI is meant for ease of use. This page shows some advanced features.

4.1 Operating the Ingester as a System Daemon

This is the preferred mode of operation for **production systems**. This mode is automatically configured if you installed `switchmap-ng` using the `root` user.

Note: Sample `systemd` files can be found in the `examples/linux/systemd/` directory.

The `ingester` can be started like this:

```
$ sudo systemctl start switchmap-ng-ingester.service
```

The `ingester` can be stopped like this:

```
$ sudo systemctl stop switchmap-ng-ingester.service
```

You can get the status of the `ingester` like this:

```
$ sudo systemctl status switchmap-ng-ingester.service
```

You can get the `ingester` to automatically restart on boot like this:

```
$ sudo systemctl enable switchmap-ng-ingester.service
```

4.2 Operating the API as a System Daemon

This is the preferred mode of operation for production systems. This mode is automatically configured if you installed `switchmap-ng` using the `root` user.

Note: Sample `systemd` files can be found in the `examples/linux/systemd/` directory.

The API can be started like this:

```
$ sudo systemctl start switchmap-ng-api.service
```

The API can be stopped like this:

```
$ sudo systemctl stop switchmap-ng-api.service
```

You can get the status of the API like this:

```
$ sudo systemctl status switchmap-ng-api.service
```

You can get the API to automatically restart on boot like this:

```
$ sudo systemctl enable switchmap-ng-api.service
```

CHAPTER 5

Command Line Interface (CLI)

This page outlines how to use the `infoset-ng` command line interface (CLI)

5.1 Viewing `infoset-ng` status

There are two important `infoset-ng` daemons.

1. **ingester:** Gets data from devices
2. **API:** Displays device data on web pages

You can get the status of each daemon using the following CLI commands:

5.1.1 Ingester status

You can get the status of the ingester using this command:

```
$ bin/infoset-ng-cli show ingester status
```

5.1.2 API status

You can get the status of the API using this command:

```
$ bin/infoset-ng-cli show api status
```

5.2 Managing the `infoset-ng` Daemons

You can manage the daemons using the CLI. Here's how:

5.2.1 Ingester Management

The ingester can be started, stopped and restarted using the following commands. Use the `--force` option only if the daemon may be hung.

```
$ bin/infoset-ng-cli ingester start

$ bin/infoset-ng-cli ingester stop
$ bin/infoset-ng-cli ingester stop --force

$ bin/infoset-ng-cli ingester restart
$ bin/infoset-ng-cli ingester restart --force
```

Note: You will need to do a restart whenever you modify a configuration parameter.

5.2.2 API Management

The API can be started, stopped and restarted using the following commands. Use the `--force` option only if the daemon may be hung.

```
$ bin/infoset-ng-cli api start

$ bin/infoset-ng-cli api stop
$ bin/infoset-ng-cli api stop --force

$ bin/infoset-ng-cli api restart
$ bin/infoset-ng-cli api restart --force
```

Note: You will need to do a restart whenever you modify a configuration parameter.

5.3 Testing Infoset-NG

You will need to verify system operation. Here's how.

```
$ bin/infoset-ng-cli test ingester
```

5.4 Viewing `infoset-ng` logs

When troubleshooting it is a good practice to view the `infoset-ng` log files.

5.4.1 Ingester logs

You can view the ingester logs using this command:

```
$ bin/infoset-ng-cli show ingester logs
```

5.4.2 API logs

You can view the API logs using this command:

```
$ bin/infoset-ng-cli show api logs
```

5.5 Viewing the `infoset-ng` Configuration

You can view the configuration using this command:

```
$ bin/infoset-ng-cli show configuration
```

CHAPTER 6

Using the API

This section outlines how to use the API

6.1 Why Infoset-ng Expects UTC Timestamps

There is a good reason for this. According to the python datetime documentation page, *The rules for time adjustment across the world are more political than rational, change frequently, and there is no standard suitable for every application aside from UTC.*

We cannot guarantee the python timezone libraries will be always up to date, so we default to UTC as recommended.

6.2 Posting Data to the API

Posting data to the API is. Add the prefix `http://SERVER_IP:6000` to all the examples below to update data in your instance of `infoset-ng`

6.2.1 Route /infoset/api/v1/receive/<id_agent>

JSON data needs to be posted to the `http://SERVER_IP:6000/infoset/api/v1/receive/<id_agent>` URL where `id_agent` is a unique identifier of the software script that is posting the data. This `id_agent` must be unique and consistent for each **script** or **application** posting data to `infoset-ng`. For example, if you have three data collection scripts running across two devices, then each script must report a unique `id_agent`, three unique IDs in total. We suggest using a hash of a random string to generate your `id_agent`. There is a 512 character limit on the size of the `agent_id`.

The example below explains the expected JSON format:

```
{'agent': 'agent_name',
'timeseries': {'label_1': {'base_type': 1,
                           'data': [[1, 224.0, 'source_1']],
```

(continues on next page)

(continued from previous page)

```
                                'description': 'description_1},
             'label_2': {'base_type': 1,
                                 'data': [[1, 1383.2, 'source_2']],
                                 'description': 'description_2'}},
'devicename': '192.168.3.100',
'timestamp': 1474823400,
'id_agent': '8a6887228e33e3b433bd0da985c203904a48e2e90804ae217334dde2b905c57e'}
```

Where feasible, we will use Linux and networking related examples to make explanation easier.

Field	Descripton
`agent`	Agent or application name. If your agent script is designed to collect server performance data, you could name it 'server_performance'. Each server performance agent would therefore report the same agent value.
`timeseries`	TimeSeries data follows
`timeseries[1][label]`	A short label defining what the data is about.
`timeseries[1][base_type]`	Define the type of data. The values are basesd on the SNMP standard. Values include: *0* for relatively unchanging alphanumeric data, which could include things like the version of an operating system; *1* for non-incremental, point-in-time numeric data such as temperature, speed, process count; *32* for numeric data that increments using a 32 bit counter such as bytes through a network interface since the device booted; *64* for 64 bit counter numeric data.
`timeseries[1][description]`	Description of the data, such as 'temperature data'
`timeseries[1][data]`	Data related to the labels. It is a list of lists. Each list has three fields *[index, value, source]*. The *index* value is a unique, unchangeable identifier for the source of the data, this is preferrably numeric such as an interface index number, but could also be string information such as an interface name or disk partition mount point. The *value* is the value of the data recorded. The *source* is a description of the source of the data to make it more recognizable when the data is eventually presented to your users. This could be *interface eth0* versus a plain *eth0*
`devicename`	Devicename of the **device** sending the data. For phone apps, this could be set to a phone number of SIM ID.
`timestamp`	Epoch **UTC** time when data was generated. This must be an integer.
`agent_id`	A unique, unchanging identifier for the **application** sending the data.

6.2.2 How to Create Agents Using Python

When you installed `infoset-ng` you probably ran the `bin/tools/test_installation.py` test script. This script emulates an agent and uses the classess in the `infoset/reference/reference.py` file to create the required JSON. Refer to this code when creating your agents.

6.3 Retrieving Data from the API

This section covers how to retrieve data from the API. First we cover some of the basics.

6.3.1 Overview

Retrieving data from infoset is easy. Add the prefix `http://SERVER_IP:6000` to all the examples below to get data from your instance of `infoset-ng`

You can test each route using the command:

```
$ curl http://SERVER_IP:6000/route
```

The json fields in the results received from the API are explained in the `infoset/db/db_orm.py` file.

6.3.1.1 Database Table Names

It is important to understand the purpose of each database table as they are used in the routes. The structure of each table can be seen by reviewing the `db_orm.py` file in the `infoset.db` module.

Table	Descripton
`iset_agent`	Data on the agents that have posted information to the API
`iset_deviceagent`	The same agent could be installed on multiple devices. This table tracks which unique device and agent combination have posted information to the API
`iset_device`	Tracks all the devices that have posted information to the API
`iset_datapoint`	Stores metadata on the various datapoints that agents report on. A datapoint ID is unique throughout the system
`iset_data`	Stores the actual data for each datapoint
`iset_billcode`	Stores data on the billing code for datapoints. Useful for financial accounting.
`iset_department`	Stores data on the departments to which the billing code should be applied. Useful for financial accounting.

6.3.2 Routes

Data is retrieved by making HTTP requests to well known URIs or `routes`. These are covered next.

6.3.2.1 Route /infoset/api/v1/agents

This route will retreive data on all agents that have ever posted data to the API. It is returned in the form of a list of lists.

Field	Description
`exists`	True if the agent exists, False if not
`enabled`	True if enabled, False if disabled
`id_agent`	The Agent ID
`idx_agent`	The unique index value of the agent in the database
`name`	The agent name
`last_timestamp`	The **UTC** timestamp of the the most recent data posted by the agent to the API

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/agents

[
  {
    "enabled": true,
    "exists": true,
    "id_agent": "ece739a93cca2c8e5444507990158b05b7d890d5798dc273578382d171bf6500",
    "idx_agent": 2,
    "last_timestamp": 1480570200,
    "name": "linux_in"
```

(continues on next page)

(continued from previous page)

```
  },
  {
    "enabled": true,
    "exists": true,
    "id_agent": "1b3c081ba928d8a1ebb16084f23e55b972b0cda1737b0449853b591f4c84ad42",
    "idx_agent": 3,
    "last_timestamp": 1480570200,
    "name": "_garnet"
  },
]
```

6.3.2.2 Route /infoset/api/v1/agents/<idx_agent>

This route retrieves information for a specific agent index value.

Field	Description
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
id_agent	The unique Agent ID
idx_agent	The unique index of the agent in the database
devicename	Unique devicename in the *infoset-ng* database
name	The agent name
last_timestamp	The **UTC** timestamp of the the most recent data posted by the agent to the API

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/agents/3

{
  "enabled": true,
  "exists": true,
  "id_agent": "70f2d9061f3ccc96915e19c13817c8207e2005d05f23959ac4c225b6a5bfe557",
  "idx_agent": 3,
  "last_timestamp": 1480611300,
  "name": "linux_in"
}
$
```

6.3.2.3 Route /infoset/api/v1/agents?id_agent=<id_agent>

This route retrieves information for a specific id_agent value as a list.

Field	Description
agent_label	Label that the agent assigned to the datapoint
agent_source	The source of the data
base_type	Base type of the data
billable	True if billable, False if not.
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
id_datapoint	The unique datapoint ID
idx_datapoint	The unique datapoint index
idx_agent	The unique index of the agent that reported on this datapoint
idx_billcode	The index of the billing code to be applied to the datapoint
idx_department	The index value of the department to which the billing code should be applied
idx_device	The unique index of the device in the database
last_timestamp	The **UTC** timestamp of the the most recent data posted by the agent to the API

Example:

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/agents?id_
↪agent=70f2d9061f3ccc96915e19c13817c8207e2005d05f23959ac4c225b6a5bfe557"

[{
  "enabled": true,
  "exists": true,
  "id_agent": "70f2d9061f3ccc96915e19c13817c8207e2005d05f23959ac4c225b6a5bfe557",
  "idx_agent": 3,
  "last_timestamp": 1480611600,
  "name": "linux_in"
}]
$
```

6.3.2.4 Route /infoset/api/v1/deviceagents

The same agent could be installed on multiple devices. This route returns data that tracks each unique device and agent combination have posted information to the API. It is returned as a list of dicts.

Field	Description
idx_agent	The index value of the agent
idx_device	The index value of the device

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/deviceagents

[
  {
    "idx_agent": 1,
    "idx_device": 1
  },
  {
    "idx_agent": 2,
    "idx_device": 2
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "idx_agent": 3,
    "idx_device": 2
  },
  {
    "idx_agent": 4,
    "idx_device": 2
  }
]
$
```

6.3.2.5 Route /infoset/api/v1/deviceagents/idx_deviceagent

The same agent could be installed on multiple devices. This route returns data that tracks each unique device and agent combination have posted information to the API, filtered by `idx_deviceagent`. It is returned as a list of dicts.

Field	Description
idx_agent	The index value of the agent
idx_device	The index value of the device

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/deviceagents

[
  {
    "idx_agent": 1,
    "idx_device": 1
  },
  {
    "idx_agent": 2,
    "idx_device": 2
  },
  {
    "idx_agent": 3,
    "idx_device": 2
  },
  {
    "idx_agent": 4,
    "idx_device": 2
  }
]
$
```

6.3.2.6 Route /infoset/api/v1/devices/<idx_device>

This route retrieves information for a specific device index value.

Field	Description
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
devicename	Unique devicename in the‘‘infoset-ng‘‘ database
idx_device	The unique index of the device in the database

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/devices/2

{
  "description": null,
  "enabled": true,
  "exists": true,
  "devicename": "afimidis",
  "idx_device": 2x
}
$
```

6.3.2.7 Route /infoset/api/v1/datapoints/<idx_datapoint>

This route retrieves information for a specific datapoint index value value.

Please read section on the API's `/infoset/api/v1/receive` route for further clarification of the field description in the table below.

Field	Description
agent_label	Label that the agent assigned to the datapoint
agent_source	The source of the data
base_type	Base type of the data
billable	True if billable, false if not.
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
id_datapoint	The unique datapoint ID
idx_datapoint	The unique datapoint index
idx_agent	The unique index of the agent that reported on this datapoint
idx_billcode	The index of the billing code to be applied to the datapoint
idx_department	The index value of the department to which the billing code should be applied
idx_device	The unique index of the device in the database
last_timestamp	The **UTC** timestamp of the the most recent data posted by the agent to the API
timefixed_value	Some datapoints may track unchanging numbers such as the version of an operating system. This value is placed here if the base_type is *0''*

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/datapoints/2

{
  "agent_label": "cpu_count",
  "agent_source": null,
  "base_type": 1,
  "billable": false,
  "enabled": true,
  "exists": true,
  "id_datapoint": "fef5fb0c60f6ecdd010c99f14d120598d322151b9d942962e6877945f1f14b5f",
  "idx_agent": 2,
  "idx_billcode": 1,
  "idx_datapoint": 2,
  "idx_department": 1,
  "idx_device": 2,
```

(continues on next page)

(continued from previous page)

```
  "last_timestamp": 1480611600,
  "timefixed_value": null
}
$
```

6.3.2.8 Route /infoset/api/v1/datapoints?id_datapoint=<id_datapoint>

This route retrieves information for a specific id_datapoint value value.

Please read section on the API's `/infoset/api/v1/receive` route for further clarification of the field description in the table below.

Field	Description
agent_label	Label that the agent assigned to the datapoint
agent_source	The source of the data
base_type	Base type of the data
billable	True if billable, false if not.
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
id_datapoint	The unique datapoint ID
idx_datapoint	The unique datapoint index
idx_agent	The unique index of the agent that reported on this datapoint
idx_billcode	The index of the billing code to be applied to the datapoint
idx_department	The index value of the department to which the billing code should be applied
idx_device	The unique index of the device in the database
last_timestamp	The **UTC** timestamp of the the most recent data posted by the agent to the API
timefixed_value	Some datapoints may track unchanging numbers such as the version of an operating system. This value is placed here if the base_type is *0''*

Example:

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/datapoints?id_
↪datapoint=fef5fb0c60f6ecdd010c99f14d120598d322151b9d942962e6877945f1f14b5f"

{
  "agent_label": "cpu_count",
  "agent_source": null,
  "base_type": 1,
  "billable": false,
  "enabled": true,
  "exists": true,
  "id_datapoint": "fef5fb0c60f6ecdd010c99f14d120598d322151b9d942962e6877945f1f14b5f",
  "idx_agent": 2,
  "idx_billcode": 1,
  "idx_datapoint": 2,
  "idx_department": 1,
  "idx_device": 2,
  "last_timestamp": 1480611600,
  "timefixed_value": null
}
$
```

6.3.2.9 Route /infoset/api/v1/datapoints?idx_deviceagent=<idx_deviceagent>

This route retrieves information for a specific idx_deviceagent value value.

Please read section on the API's `/infoset/api/v1/receive` route for further clarification of the field description in the table below.

Field	Description
agent_label	Label that the agent assigned to the datapoint
agent_source	The source of the data
base_type	Base type of the data
billable	True if billable, false if not.
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
id_datapoint	The unique datapoint ID
idx_datapoint	The unique datapoint index
idx_deviceagent	The unique index of the device agent that reported on this datapoint
idx_billcode	The index of the billing code to be applied to the datapoint
idx_department	The index value of the department to which the billing code should be applied
idx_device	The unique index of the device in the database
last_timestamp	The **UTC** timestamp of the the most recent data posted by the agent to the API
timefixed_value	Some datapoints may track unchanging numbers such as the version of an operating system. This value is placed here if the base_type is *0''*

Example:

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/datapoints?idx_deviceagent=2"

{
  "agent_label": "cpu_count",
  "agent_source": null,
  "base_type": 1,
  "billable": false,
  "enabled": true,
  "exists": true,
  "id_datapoint": "fef5fb0c60f6ecdd010c99f14d120598d322151b9d942962e6877945f1f14b5f",
  "idx_deviceagent": 2,
  "idx_billcode": 1,
  "idx_datapoint": 2,
  "idx_department": 1,
  "idx_device": 2,
  "last_timestamp": 1480611600,
  "timefixed_value": null
}
$
```

6.3.2.10 Route /infoset/api/v1/datapoints/all/summary

This route retrieves dummary information about all datapoints.

Please read section on the API's `/infoset/api/v1/receive` route for further clarification of the field description in the table below.

Field	Description
agent_label	Label that the agent assigned to the datapoint
agent_source	The source of the data
devicename	Unique devicename in the *infoset-ng* database
id_agent	The unique Agent ID
id_datapoint	The unique datapoint ID
idx_datapoint	The unique datapoint index
idx_deviceagent	The unique index of the deviceagent in the database
name	The agent name

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/datapoints/all/summary
[
  {
    "agent_label": "system",
    "agent_source": null,
    "devicename": "palisadoes",
    "id_agent": "f32eda632703ac9d94d80b43d5dd54d0198cd0dabf541dae97b94e5b75b851d5",
    "idx_datapoint": 417,
    "idx_deviceagent": 4,
    "name": "remote_linux_passive"
  },
  {
    "agent_label": "version",
    "agent_source": null,
    "devicename": "palisadoes",
    "id_agent": "f32eda632703ac9d94d80b43d5dd54d0198cd0dabf541dae97b94e5b75b851d5",
    "idx_datapoint": 418,
    "idx_deviceagent": 4,
    "name": "remote_linux_passive"
  }
]
$
```

6.3.2.11 Route /infoset/api/v1/datapoints&id_datapoint="<id_datapoint>"

This route retrieves information for a specific datapoint ID value value.

Please read section on the API's `/infoset/api/v1/receive` route for further clarification of the field description in the table below.

Field	Description
agent_label	Label that the agent assigned to the datapoint
agent_source	The source of the data
base_type	Base type of the data
billable	True if billable, false if not.
enabled	True if enabled, False if not
exists	True if the requested index value exists in the database
id_datapoint	The unique datapoint ID
idx_datapoint	The unique datapoint index
idx_agent	The unique index of the agent that reported on this datapoint
idx_billcode	The index of the billing code to be applied to the datapoint
idx_department	The index value of the department to which the billing code should be applied
idx_device	The unique index of the device in the database
last_timestamp	The **UTC** timestamp of the the most recent data posted by the agent to the API
timefixed_value	Some datapoints may track unchanging numbers such as the version of an operating system. This value is placed here if the base_type is *0‘‘*

Example:

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/datapoints?id_
↪datapoint=fef5fb0c60f6ecdd010c99f14d120598d322151b9d942962e6877945f1f14b5f"

{
  "agent_label": "cpu_count",
  "agent_source": null,
  "base_type": 1,
  "billable": false,
  "enabled": true,
  "exists": true,
  "id_datapoint": "fef5fb0c60f6ecdd010c99f14d120598d322151b9d942962e6877945f1f14b5f",
  "idx_agent": 2,
  "idx_billcode": 1,
  "idx_datapoint": 2,
  "idx_department": 1,
  "idx_device": 2,
  "last_timestamp": 1480612500,
  "timefixed_value": null
}
$
```

6.3.2.12 Route /infoset/api/v1/devices/<idx_device>/agents

This route will retreive data on all the agents that have reported data from a specific device. The agent data returned are their index values, and the query is done based on the index of the device.

Example:

```
$ curl http://SERVER_IP:6000/infoset/api/v1/devices/2/agents

[
  2,
  3,
  4
```

(continues on next page)

(continued from previous page)

```
]
$
```

6.3.2.13 Route /infoset/api/v1/lastcontacts

This route will retreive **all** the most recently posted data values.

Data is queried starting from 10X the interval value in your configuration file seconds ago until the present.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl http://SERVER_IP:6000/infoset/api/v1/lastcontacts

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 60370900.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 60370900.0
  },

...
...
...
...
...
...

  {
    "idx_datapoint": 417,
    "timestamp": 1483629900,
    "value": 60370900.0
  },
  {
    "idx_datapoint": 418,
    "timestamp": 1483629900,
    "value": 60370900.0
  }
]
```

6.3.2.14 Route /infoset/api/v1/lastcontacts?secondsago=<seconds>

This route will retreive **all** the most recently posted data values.

The query starts looking for contacts as of `secondsago` seconds ago.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts`, which is the preferred method of getting this data.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts?secondsago=3600"

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 60370900.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 60370900.0
  },

...
...
...
...
...
...

  {
    "idx_datapoint": 417,
    "timestamp": 1483629900,
    "value": 60370900.0
  },
  {
    "idx_datapoint": 418,
    "timestamp": 1483629900,
    "value": 60370900.0
  }
]
```

6.3.2.15 Route /infoset/api/v1/lastcontacts?ts_start="timestamp"

This route will retreive **all** the most recently posted data values.

A starting **UTC** timestamp needs to be provided. Searches for contacts are made from starting at this time until the present.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts`, which is the preferred method of getting this data.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts?ts_start=0"

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 60370900.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 60370900.0
  },

...
...
...
...
...
...

  {
    "idx_datapoint": 417,
    "timestamp": 1483629900,
    "value": 60370900.0
  },
  {
    "idx_datapoint": 418,
    "timestamp": 1483629900,
    "value": 60370900.0
  }
]
```

6.3.2.16 Route /infoset/api/v1/lastcontacts/id_agents

This route will retreive **all** the most recently posted data values.

Data is queried starting from 10X the interval value in your configuration file seconds ago until the present.

Field	Description
devicename	The name of the device generating the data
name	The name of the agent generating the data
id_agent	The ID of the agent generating the data
timestamp	**UTC** timestamp of the most recent contact
value	Value of the datapoint reading at the timestamp's point in time

```
$ curl http://SERVER_IP:6000/infoset/api/v1/lastcontacts/id_agents

[
  {
    "devicename": "name_1",
    "id_agent": "bec9ba91e14804001e037fa4f52c94fb1ef027d04e1b86f6a74ab36e3b073609",
    "name": "Agent_Name",
    "timeseries": {
```

(continues on next page)

(continued from previous page)

```
      "agent_label_1": {
        "timestamp": 1487377500,
        "value": 1.8191
      },
      "agent_label_2": {
        "timestamp": 1487377500,
        "value": 1.8694
      }
    }
  },
  {
    "devicename": "name_2",
    "id_agent": "e33ce6311cf95c6264c6777323e9c717220b19ccad7b6da1877384e7fb3364e7",
    "name": "Agent_Name",
    "timeseries": {
      "agent_label_1": {
        "timestamp": 1487377500,
        "value": 1.61078
      },
      "agent_label_2": {
        "timestamp": 1487377500,
        "value": 1.54421
      }
    }
  }
]
```

6.3.2.17 Route /infoset/api/v1/lastcontacts/id_agents?secondsago=<seconds>

This route will retreive **all** the most recently posted data values.

The query starts looking for contacts as of `secondsago` seconds ago.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts/id_agents`, which is the preferred method of getting this data.

Field	Description
`devicename`	The name of the device generating the data
`name`	The name of the agent generating the data
`id_agent`	The ID of the agent generating the data
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts/id_agents?secondsago=3600"

[
  {
    "devicename": "name_1",
    "id_agent": "bec9ba91e14804001e037fa4f52c94fb1ef027d04e1b86f6a74ab36e3b073609",
    "name": "Agent_Name",
    "timeseries": {
      "agent_label_1": {
        "timestamp": 1487377500,
        "value": 1.8191
```

(continues on next page)

(continued from previous page)

```
      },
      "agent_label_2": {
        "timestamp": 1487377500,
        "value": 1.8694
      }
    }
  },
  {
    "devicename": "name_2",
    "id_agent": "e33ce6311cf95c6264c6777323e9c717220b19ccad7b6da1877384e7fb3364e7",
    "name": "Agent_Name",
    "timeseries": {
      "agent_label_1": {
        "timestamp": 1487377500,
        "value": 1.61078
      },
      "agent_label_2": {
        "timestamp": 1487377500,
        "value": 1.54421
      }
    }
  }
]
```

6.3.2.18 Route /infoset/api/v1/lastcontacts/id_agents?ts_start="timestamp"

This route will retreive **all** the most recently posted data values.

A starting **UTC** timestamp needs to be provided. Searches for contacts are made from starting at this time until the present.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts/id_agents`, which is the preferred method of getting this data.

Field	Description
`devicename`	The name of the device generating the data
`name`	The name of the agent generating the data
`id_agent`	The ID of the agent generating the data
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts/id_agents?ts_start=0"

[
  {
    "devicename": "name_1",
    "id_agent": "bec9ba91e14804001e037fa4f52c94fb1ef027d04e1b86f6a74ab36e3b073609",
    "name": "Agent_Name",
    "timeseries": {
      "agent_label_1": {
        "timestamp": 1487377500,
        "value": 1.8191
      },
      "agent_label_2": {
```

(continues on next page)

(continued from previous page)

```
        "timestamp": 1487377500,
        "value": 1.8694
      }
    }
  },
  {
    "devicename": "name_2",
    "id_agent": "e33ce6311cf95c6264c6777323e9c717220b19ccad7b6da1877384e7fb3364e7",
    "name": "Agent_Name",
    "timeseries": {
      "agent_label_1": {
        "timestamp": 1487377500,
        "value": 1.61078
      },
      "agent_label_2": {
        "timestamp": 1487377500,
        "value": 1.54421
      }
    }
  }
]
```

6.3.2.19 Route /infoset/api/v1/lastcontacts/<idx_deviceagent>

Searches for contacts are made starting from an hour ago to the present. from a specific Device Agent combination. The query is done based on the device's deviceagent index.

Data is queried starting from 10X the interval value in your configuration file seconds ago until the present.

Field	Description
idx_datapoint	The datapoint index value
timestamp	**UTC** timestamp of the most recent contact
value	Value of the datapoint reading at the timestamp's point in time

```
$ curl http://SERVER_IP:6000/infoset/api/v1/lastcontacts/2

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 4,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 5,
```

(continues on next page)

(continued from previous page)

```
    "timestamp": 1483629900,
    "value": 9.0
  }
]
```

6.3.2.20 Route /infoset/api/v1/lastcontacts/<idx_deviceagent>?secondsago="seconds"

This route will retreive the most recently posted data values from a specific Device Agent combination. The query is done based on the device's deviceagent index.

Data is queried starting from 10X the interval value in your configuration file seconds ago until the present.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts`, which is the preferred method of getting this data.

Field	Description
idx_datapoint	The datapoint index value
timestamp	**UTC** timestamp of the most recent contact
value	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts/2?secondsago=3600"

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 4,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 5,
    "timestamp": 1483629900,
    "value": 9.0
  }
]
$
```

6.3.2.21 Route /infoset/api/v1/lastcontacts/<idx_deviceagent>?ts_start="timestamp"

This route will retreive the most recently posted data values from a specific Device Agent combination. The query is done based on the device's deviceagent index.

A starting **UTC** timestamp needs to be provided. Searches for contacts are made from starting at this time until the present.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts`, which is the preferred method of getting this data.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts/2?ts_start=0"

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 4,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 5,
    "timestamp": 1483629900,
    "value": 9.0
  }
]
$
```

6.3.2.22 Route /infoset/api/v1/lastcontacts/devicenames/<devicename>/id_agents/<id_agent>

Searches for contacts are made starting from an hour ago to the present. from a specific `devicename` and `id_agent` combination.

Data is queried starting from 10X the interval value in your configuration file seconds ago until the present.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl http://SERVER_IP:6000/infoset/api/v1/devicenames/_INFOSET_TEST_/id_agents/
↪558bb0055d7b4299c2ebe6abcc53de64a9ec4847b3f82238b3682cad575c7749

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
```

(continues on next page)

(continued from previous page)

```
    "value": 9.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 4,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 5,
    "timestamp": 1483629900,
    "value": 9.0
  }
]
```

Route /infoset/api/v1/lastcontacts/devicenames/<devicename>/id_agents/<id_agent> ?ts_start=‘‘timestamp‘

This route will retreive the most recently posted data values from a specific `devicename` and `id_agent` combination.

A starting **UTC** timestamp needs to be provided. Searches for contacts are made from starting at this time until the present.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts/devicenames/<devicename>/id_agents/<id_agent>`, which is the preferred method of getting this data.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts/devicenames/_INFOSET_TEST_/
↪id_agent/558bb0055d7b4299c2ebe6abcc53de64a9ec4847b3f82238b3682cad575c7749/?ts_
↪start=0"
[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 4,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "idx_datapoint": 5,
    "timestamp": 1483629900,
    "value": 9.0
  }
]
$
```

Route /infoset/api/v1/lastcontacts/devicenames/`<devicename>`/id_agents/`<id_agent>` ?secondsago=‘‘seconds‘‘
^^

This route will retreive the most recently posted data values from a specific `devicename` and `id_agent` combination.

A starting **UTC** timestamp needs to be provided. Searches for contacts are made from starting at this time until the present.

This route does not use the cache as efficiently as `/infoset/api/v1/lastcontacts/devicenames/<devicename>/id_agents/<id_agent>`, which is the preferred method of getting this data.

Field	Description
`idx_datapoint`	The datapoint index value
`timestamp`	**UTC** timestamp of the most recent contact
`value`	Value of the datapoint reading at the timestamp's point in time

```
$ curl "http://SERVER_IP:6000/infoset/api/v1/lastcontacts/devicenames/_INFOSET_TEST_/
↪id_agent/558bb0055d7b4299c2ebe6abcc53de64a9ec4847b3f82238b3682cad575c7749/?
↪secondsago=3600"

[
  {
    "idx_datapoint": 2,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 3,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 4,
    "timestamp": 1483629900,
    "value": 9.0
  },
  {
    "idx_datapoint": 5,
    "timestamp": 1483629900,
    "value": 9.0
  }
]
$
```

CHAPTER 7

Troubleshooting

There are a number of ways you can troubleshoot the `ingester` and `API`. The most accessible ways are through the log files and the API test script.

7.1 Ingester Troubleshooting

This section covers the various ways you can troubleshoot `ingester` operation.

7.1.1 Ingester Logging

It is always good to verify the operation of the `ingester` by observing changes in its log file. It is a good source of troubleshooting information.

You can see these changes as they occur by using the `tail -f` command as seen below:

```
$ tail -f /opt/infoset-ng/log/infoset-ng.log
```

The location of the log file is governed by the `log_directory` parameter in the configuration.

7.1.2 Testing Ingester Operation

You can test the operation of the API by using the `curl` command which is often used to test basic website functionality. The example below shows how. Replace `SERVER_IP` with the IP address or fully qualified DNS name.

```
$ curl http://SERVER_IP:6000/infoset/api/v1/status
infoset-ng API Operational.
$
```

The `curl` response should be `infoset-ng API Operational` if successful.

7.1.3 Invalid Agents

There is the possibility that agents may be posting incorrectly formatted JSON data to the `API`. You can view the contents of these invalidated files in the `failures/` sub-directory of the `API` cache directory. The cache directory is defined in the `ingest_cache_directory:` option of the configuration file.

7.2 API Troubleshooting

There are a number of ways you can troubleshoot the `API`. The most accessible ways are through the log file.

7.2.1 API Logging

It is always good to verify the operation of the `API` by observing changes in its log file. It is a good source of troubleshooting information.

You can see these changes as they occur by using the `tail -f` command as seen below:

```
$ tail -f /opt/infoset-ng/log/api-web.log
```

The location of the log file is governed by the `log_directory` parameter in the configuration.

7.2.2 Poor or Blocked Network Connectivity

It is possible that there could be firewalls or intermittent connectivity causing issues to your `API` you should familarize yourself with the `tcpdump` command to determine whether connections are coming through.

In this example we are testing to see whether we are receiving traffic from IP address 192.168.1.100 on TCP port 6000 which the `API` uses

```
$ sudo tcpdump -ni tcp port 6000 and host 192.168.1.100
```

You can also use the basic `telnet` command to determine whether the remote device or network can communicate with the `API`. In this example we are testing to see whether we can communicate with the `API` running on a server with IP address 192.168.1.200 on the default TCP port 6000.

```
$ telnet 192.168.1.200 6000
Trying 192.168.1.200...
Connected to 192.168.1.200.
Escape character is '^]'.
^]
telnet> quit
Connection closed.
```

If you get no response, then you need. Try this approach on both the local and remote ends of the connection. In other words, use the same command on both the remote client and `API` server. If there is response on the server, but none on the client, then there is probably a connectivity issue.

You can also determine whether the `API` server is running at all. Use the `netstat` command on the `API` server itself to determine whether it is listening on port 6000. If there is no response, then the API isn't running.

```
$ netstat -ant |grep 6000
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
$
```

You should also try to use the `curl` examples in the `API` guide to assist further.

CHAPTER 8

Best Practices

There are a number of best practices to consider when implementing `infoset-ng`.

8.1 Use a Web Proxy Server

`infoset-ng` uses `Gunicorn` as lightweight webserver. The `Gunicorn` development team strongly recommends operating `Gunicorn` behind a proxy server.

8.1.1 Nginx Configuration

Although there are many HTTP proxies available, the `Gunicorn` team strongly advises that you use `Nginx`.

According to their website: *If you choose another proxy server you need to make sure that it buffers slow clients when you use default Gunicorn workers. Without this buffering Gunicorn will be easily susceptible to denial-of-service.*

A sample configuration can be found in the `examples/linux/nginx` directory

We also advise that you harden your `nginx` installation to reduce security risks.

8.1.2 Apache Configuration

This is the less preferred option. Use `Nginx` whenever possible.

A sample configuration can be found in the `examples/linux/apache` directory

We also advise that you harden your `nginx` installation to reduce security risks.

CHAPTER 9

Community

Use these channels to communicate about the project.

9.1 Mailing list

The **user** mailing list is general discussion and support list for Gunicorn users.

- To **subscribe**, send an email to *TBD*
- To **unsubscribe**, send an email to *TBD*
- Finally, to post a message to the list use the address to *TBD*

The archive for this list can also be browsed online.

9.2 Irc

The Gunicorn channel is on the Freenode IRC network. You can chat with other on *#TBD*.

9.3 Issue Tracking

Bug reports, enhancement requests and tasks generally go in the Github issue tracker.

9.4 Security Issues

The security mailing list is a place to report security issues. Only developers are subscribed to it. To post a message to the list use the address to *TBD*.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search