Pattoo Documentation

Peter Harrison

Aug 31, 2020

Introduction:

`pattoo` stores timeseries data in a database and makes it available for users via a GraphQL API.

Visit the Pattoo GitHub site to see the code.

CHAPTER 1

Introduction

General information about the project, including the the prerequisite steps to get it operational on your system.

1.1 About Pattoo

`pattoo` allows you to use your web browser to chart your organization's constantly changing data.

It was inspired by the need to collect and visualize data from various DevOps, network, industrial PLC controllers, electro-mechanical and enterprise systems on a single web dashboard.

This data is collected by `pattoo` agents. There are standard agents for:

- Linux
- SNMP
- Modbus TCP
- Bacnet/IP
- OPC UA

With programming skill, you can create your own custom agents if needed.

1.1.1 Operational Overview

`pattoo` has a number of inter-related components. You can see how they all work together on the pattoo web page.

1.1.2 The Palisadoes Foundation

`pattoo` is based on the original `infoset` code created by the Palisadoes Foundation as part of its annual Calico Challenge program. Calico provides paid summer internships for Jamaican university students to work on selected open source projects. They are mentored by software professionals and receive stipends based on the completion of predefined milestones. Calico was started in 2015.

1.2 Basic Installation

This section covers some key steps to get you started.

1.2.1 Prerequisites

There are some software components that need to be installed prior to starting.

1. `pattoo` requires the installation of a MySQL or MariaDB database. Make sure this software is installed beforehand.
2. `pattoo` only runs on Python 3.6 or higher

Let's install the software.

1.2.2 Installation

Follow these steps.

1. Install `git` on your system.
2. Select and create the parent directory in which you want to install `pattoo`.

```
$ mkdir -p /installation/parent/directory
$ cd /installation/parent/directory
```

3. Clone the repository to the parent directory using the `git clone` command. You can also choose to downloading and unzip the file in the parent directory. The repository can be found at: https://github.com/PalisadoesFoundation/pattoo

```
$ cd /installation/parent/directory
$ git clone https://github.com/PalisadoesFoundation/pattoo.git
```

4. Enter the `/installation/parent/directory/pattoo` directory with the `pattoo` files.
5. Install the required packages using the `pip_requirements` document in the `pattoo` root directory

```
$ pip3 install -r pip_requirements.txt
```

6. Create the `MySQL` or `MariaDB` database for `pattoo` with the correct authentication parameters provided in the *Configuration Guide* In this example adjust the name of your database and the password accordingly.

```
$ sudo mysql
```

```
CREATE DATABASE pattoo;
GRANT ALL PRIVILEGES ON pattoo.* TO pattoo@"localhost" IDENTIFIED BY
↪'PATTOO_PASSWORD';
FLUSH PRIVILEGES;
exit;
```

7. Use the *Configuration Guide* to create a working configuration.
8. Run the installation script

```
$ sudo setup/install.py install all
```

9. View additional installation options installation_modes

10. Configure the required `cron` jobs. *Periodic Jobs*

1.3 Configuration Guide

After installation, you will need to create a configuration file in a directory dedicated to `pattoo`.

1.3.1 Setting the Configuration Directory Location

Currently the configuration directory is automatically set when the installationscript is run

1.3.2 Configuration Options

There are two ways to configure `pattoo`. These are the:

1. Quick Method
2. Expert Method

Quick Method

Use the quick method if you are new to `pattoo`.

```
$ sudo setup/pattoo_installation.py install configuration
```

The above command will set the most optimal defaults for your system for pattoo.

Additionally, the pattoo user and group will be created with the home directory for the pattoo user being /home/pattoo All related directories and their subdirectories will be owned by the pattoo user

To guarantee success you will need to know the following beforehand.

1. `db_name`: Database name
2. `db_username`: Database username
3. `db_password`: Database password
4. `db_hostname`: Database hostname

Expert Method

This section goes into configuration parameters in great detail.

Copy the Templates to Your Configuration Directory

Follow the steps in this section if you don't already have a valid configuration files in your `PATTOO_CONFIGDIR` directory.

Copy the template files in the `examples/etc` directory to the `PATTOO_CONFIGDIR` location.

NOTE: If a `/path/to/configuration/directory/pattoo_server.yaml` or `/path/to/configuration/directory/pattoo.yaml` file already exists in the directory then skip this step and edit the file according to the steps in following sections.

```
$ cp examples/etc/pattoo_server.yaml.template \
    /path/to/configuration/directory/pattoo_server.yaml

$ cp examples/etc/pattoo.yaml.template \
    /path/to/configuration/directory/pattoo.yaml
```

The next step is to edit the contents of both files.

Edit Your Configuration Files

The `pattoo` server uses two configuration files:

1. `pattoo.yaml`: Provides general configuration information for all `pattoo` related applications. `pattoo.yaml` also defines how `pattoo` agents should connect to the `pattoo` server APIs.
2. `pattoo_server.yaml`: Provides configuration details for all the `pattoo` server's API daemons. These APIs accept data from `pattoo` agents and also provide data to `pattoo` related applications through your browser.

Take some time to read up on `YAML` formatted files if you are not familiar with them. A background knowledge is always helpful.

Server Configuration File

The `pattoo_server.yaml` file created from the template will have sections that you will need to edit with custom values. Don't worry, these sections are easily identifiable as they all start with `PATTOO_`

NOTE: The indentations in the YAML configuration are important. Make sure indentations line up. Dashes '-' indicate one item in a list of items (if applicable).

```
pattoo_api_agentd:

    ip_bind_port: 20201
    ip_listen_address: 0.0.0.0

pattoo_apid:

    ip_bind_port: 20202
    ip_listen_address: 0.0.0.0

pattoo_ingesterd:

    ingester_interval: 3600
    batch_size: 500
    graceful_timeout: 10

pattoo_db:
    db_pool_size: 10
    db_max_overflow: 10
    db_hostname: PATTOO_DB_HOSTNAME
    db_name: PATTOO_DB_NAME
    db_password: PATTOO_DB_PASSWORD
    db_username: PATTOO_DB_USERNAME
```

Server Configuration Explanation

This table outlines the purpose of each configuration parameter.

Section	Configuration Parameters	Description
pattoo_api_agentd		
	ip_listen_address	IP address used by the pattoo_api_agentd daemon for accepting data from remote pattoo agents. Default of '0.0.0.0' which indicates listening on all available network interfaces. You can also use IPv6 nomenclature such as ::. The pattoo APIs don't support IPv6 and IPv4 at the same time.
	ip_bind_port	TCP port of used by the pattoo_api_agentd daemon for accepting data from remote pattoo agents. Default of 20201.
pattoo_apid		
	ip_listen_address	IP address used by the pattoo_apid daemon for providing data to remote clients. Default of '0.0.0.0' which indicates listening on all available network interfaces. You can also use IPv6 nomenclature such as ::. The pattoo APIs don't support IPv6 and IPv4 at the same time.
	ip_bind_port	TCP port of used by the pattoo_apid daemon for providing data to remote clients. Default of 20202.
pattoo_ingesterd		
	ingester_interval	The interval between checking for new agent files in the cache directory. Only valid if using the pattoo_ingesterd daemon.
	batch_size	The number of files to read per processing batch until all files are processed.
	graceful_timeout	The amount of time required for the ingester to finish processing data when the stop or restart command is excuted before it is forcefully stopped or restarted.
pattoo_db		
	db_hostname	Hostname of the database server
	db_username	Username required for database access
	db_password	Password required for database access
	db_name	Name of database
	db_pool_size	This is the largest number of connections that will be keep persistently with the database
	db_max_overflow	Maximum overflow size. When the number of connections reaches the size set in db_pool_size, additional connections will be returned up to this limit. This is the floating number of additional database connections to be made available.

Client Configuration File

The pattoo.yaml file created from the template will have sections that you will need to edit with custom values. Don't worry, these sections are easily identifiable as they all start with PATTOO_

NOTE: The indentations in the YAML configuration are important. Make sure indentations line up. Dashes '-' indicate one item in a list of items (if applicable).

```
pattoo:
    log_level: debug
    log_directory: PATTOO_LOG_DIRECTORY
    cache_directory: PATTOO_CACHE_DIRECTORY
    daemon_directory: PATTOO_DAEMON_DIRECTORY
    system_daemon_directory: PATTOO_SYSTEM_DAEMON_DIRECTORY
```

Client Configuration Explanation

This table outlines the purpose of each configuration parameter.

Section	Configuration Parameters	Description
pattoo		
	log_directory	Path to logging directory. Make sure the username running the daemons have RW access to files there.
	log_level	Default level of logging. debug is best for troubleshooting.
	cache_directory	Directory that will temporarily store data data from agents prior to be added to the pattoo database.
	daemon_directory	Directory used to store daemon related data that needs to be maintained between reboots
	system_daemon_directory	Directory used to store daemon related data that should be deleted between reboots. This should only be configured if you are running pattoo daemons as systemd daemons. The systemd daemon installation procedure automatically adjusts this configuration. This parameter defaults to the daemon_directory value if it is not configured.

1.4 Configuring systemd Daemons

You can also setup all the pattoo related daemons located in this GitHub repository as system daemons by executing the setup/systemd/bin/install_systemd.py script.

The script requires you to specify the following parameters. Make sure you have a username and group created for running your pattoo services.

```
usage: install_systemd.py [-h] -f CONFIG_DIR -u USERNAME -g GROUP

optional arguments:
  -h, --help            show this help message and exit
  -f CONFIG_DIR, --config_dir CONFIG_DIR
                        Directory where the pattoo configuration files will be located
  -u USERNAME, --username USERNAME
                        Username that will run the daemon
  -g GROUP, --group GROUP
                        User group to which username belongs
```

Note The daemons are not enabled or started by default. You will have to do this separately using the systemctl command after running the script.

```
$ sudo setup/systemd/bin/install_systemd.py --user pattoo --group pattoo --config_dir␣
↪/etc/pattoo

SUCCESS! You are now able to start/stop and enable/disable the following systemd␣
↪services:

pattoo_api_agentd.service
pattoo_apid.service
pattoo_ingesterd.service

$
```

1.5 Backup and Restoration

Always take precautions. Backup your data as you'll never know when you'll need to restore it.

1.5.1 Backup

It is strongly advised that you backup your agents to protect you in the event of catastrophe.

The following directories need to be saved periodically.

1. The `PATTOO_CONFIGDIR` directory which contains your configuration
2. The `daemon_directory` location defined in your configuration. This area stores important authentication information.
3. The `pattoo` directory which contains your source code.

We'll discuss data restoration next.

1.5.2 Restoration

It's important to follow these steps in this order when restoring `pattoo` after a disaster.

1. FIRST make sure all the `pattoo` agents are stopped.
2. SECOND restore the contents of the `daemon_directory` location defined in your configuration. This area stores important authentication information.
3. Restore the `PATTOO_CONFIGDIR` directory which contains your configuration
4. Restore the `pattoo` directory which contains your source code.

You should now be able to restart your agents without issue.

CHAPTER 2

Daemon and Cron Setup

How to get the daemons running to collect data.

2.1 Periodic Jobs

You will need to configure some jobs to improve `pattoo` performance and troubleshooting.

2.1.1 Logrotate Configuration

The default `pattoo` debug logging mode can quickly create large logging files. The logrotate utility can automatically compress and archive them.

1. Copy the the `examples/logrotate.d/pattoo` file to the `/etc/logrotate.d` directory.
2. Edit the file path accordingly.

Read up on the logrotate utility if you are not familiar with it. The documentation is easy to follow.

2.2 Pattoo Web API Daemon

`pattoo_apid` serves `pattoo` agent data from the database via a web API.

2.2.1 Installation

Follow these steps.

1. Follow the installation steps in the *Basic Installation* file.
2. Configure the main section of the configuration file following the steps in *Configuration Guide* file.

3. Start the desired daemons using the commands below. You may want to make these `systemd` daemons, if so follow the steps in the *Basic Installation* file.

2.2.2 Usage

`pattoo_apid` has a simple command structure.

The daemon will require a configuration file in the `etc/` directory. See the configuration section for details.

```
$ bin/pattoo_apid.py --help
usage: pattoo_apid.py [-h] [--start] [--stop] [--status] [--restart]
                      [--force]

optional arguments:
  -h, --help  show this help message and exit
  --start     Start the agent daemon.
  --stop      Stop the agent daemon.
  --status    Get daemon daemon status.
  --restart   Restart the agent daemon.
  --force     Stops or restarts the agent daemon ungracefully when used with --stop or
              --restart.
$
```

2.2.3 Configuration

No additional configuration steps beyond that in the *Configuration Guide* file are required.

2.2.4 Testing

There are a number of steps you can take to make sure everything is OK.

1. If you have setup the daemon for `systemd` then you can use the `systemctl` command to get the status of the daemon.
2. The daemon should be running on the port configured with the `ip_bind_port` parameter. Use the `netstat` command to verify this.
3. Visit the URL `http://localhost:20202/pattoo/api/v1/web/status` to get the status page.
4. Use the *Performance and Troubleshooting* for further steps to take

2.2.5 Making `pattoo_apid` Start Automatically After Reboot

The easiest way to do this is to consider *Configuring systemd Daemons*. Otherwise you will need to manually restart the daemon after a reboot.

2.3 Pattoo Agent API Daemon

The `pattoo_api_agentd` API daemon accepts data from remote `pattoo` agents for storage in a database.

2.3.1 Installation

Follow these steps.

1. Follow the installation steps in the *Basic Installation* file.
2. Configure the main section of the configuration file following the steps in *Configuration Guide* file.
3. Start the desired daemons using the commands below. You may want to make these `systemd` daemons, if so follow the steps in the *Basic Installation* file.

2.3.2 Usage

`pattoo_api_agentd` has a simple command structure.

The daemon will require a configuration file in the `etc/`directory. See the configuration section for details.

```
$ bin/pattoo_api_agentd.py --help
usage: pattoo_api_agentd.py [-h] [--start] [--stop] [--status] [--restart]
                            [--force]

optional arguments:
  -h, --help  show this help message and exit
  --start     Start the agent daemon.
  --stop      Stop the agent daemon.
  --status    Get daemon daemon status.
  --restart   Restart the agent daemon.
  --force     Stops or restarts the agent daemon ungracefully when used with --stop or
              --restart.
$
```

2.3.3 Configuration

No additional configuration steps beyond that in the *Configuration Guide* file are required.

2.3.4 Testing

There are a number of steps you can take to make sure everything is OK.

1. If you have setup the daemon for `systemd` then you can use the `systemctl` command to get the status of the daemon.
2. The daemon should be running on the port configured with the `ip_bind_port` parameter. Use the `netstat` command to verify this.
3. The `pattoo_api_agentd` temporarily stores all the data it receives from `pattoo` agents in the `cache/` directory. Check there for recent `.json` files.
4. Visit the URL `http://localhost:20201/pattoo/api/v1/agent/status` to get the status page.
5. Use the *Performance and Troubleshooting* for further steps to take

2.3.5 Making `pattoo_api_agentd` Start Automatically After Reboot

The easiest way to do this is to consider *Configuring systemd Daemons*. Otherwise you will need to manually restart the daemon after a reboot.

2.4 Pattoo Ingester Daemon

`pattoo_ingesterd` batch processes agent data received by the `pattoo` agent API daemon.

2.4.1 Installation

Follow these steps.

1. Follow the installation steps in the *Basic Installation* file.
2. Configure the main section of the configuration file following the steps in *Configuration Guide* file.
3. Start the desired daemons using the commands below. You may want to make these `systemd` daemons, if so follow the steps in the *Basic Installation* file.

2.4.2 Usage

`pattoo_ingesterd` has a simple command structure.

The daemon will require a configuration file in the `etc/` directory. See the configuration section for details.

```
$ bin/pattoo_ingesterd.py --help
usage: pattoo_ingesterd.py [-h] [--start] [--stop] [--status] [--restart]
                           [--force]

optional arguments:
  -h, --help  show this help message and exit
  --start     Start the agent daemon.
  --stop      Stop the agent daemon.
  --status    Get daemon daemon status.
  --restart   Restart the agent daemon.
  --force     Stops or restarts the agent daemon ungracefully when used with --stop or
              --restart.
$
```

2.4.3 Configuration

No additional configuration steps beyond that in the *Configuration Guide* file are required.

2.4.4 Testing

There are a number of steps you can take to make sure everything is OK.

1. If you have setup the daemon for `systemd` then you can use the `systemctl` command to get the status of the daemon.
2. Use the *Performance and Troubleshooting* for further steps to take

The `pattoo_ingester`

There is also a `bin/pattoo_ingester.py` script that can be used as needed for troubleshooting. Here's how to use it:

1. Edit your `PATTOO_CONFIGDIR` path accordingly.
2. Stop the `pattoo_ingesterd` daemon.
3. Edit your `PATTOO_CONFIGDIR` path accordingly.
4. Check your log files for any possible errors.

2.4.5 Making `pattoo_ingesterd` Start Automatically After Reboot

The easiest way to do this is to consider *Configuring systemd Daemons*. Otherwise you will need to manually restart the daemon after a reboot.

CHAPTER 3

Using the CLI

How to use the Command Line Interface (CLI).

3.1 Using the CLI

The command line interface allows you to interact with the database in a number of ways.

3.1.1 Location of the CLI Script

The CLI script is located in the `bin/` directory and is called `bin/pattoo_cli.py`.

Running the CLI script without any parameters will display the usage options as seen below.

```
$ bin/pattoo_cli.py
usage: pattoo_cli.py [-h] {show,create,set,import,assign} ...

This program is the CLI interface to configuring pattoo

positional arguments:
  {show,create,set,import,assign}
    show                Show contents of pattoo DB.
    create              Create entries in pattoo DB.
    set                 Show contents of pattoo DB.
    import              Import data into the pattoo DB.
    assign              Assign contents of pattoo DB.

optional arguments:
  -h, --help            show this help message and exit
```

3.1.2 Language

`pattoo` is meant to support multiple languages. The default language is English with a language code of `en`. Agents do not post language specific data, but the keys used to define the data will need to be translated to descriptions that are meaningful to the end user.

Language translation files should be provided with any pattoo agent you install. You may have to create your own translation files for agents that poll data from non-standard data sources.

Viewing Languages

To view languages configured in the `pattoo` database use the `bin/pattoo_cli.py show language` command.

```
$ bin/pattoo_cli.py show language

idx_language  code  name

1             en    English
```

Upadating Language Names

You can update the name of a language using the `bin/pattoo_cli.py set language` command. You must specify the language code and provide a name using the `--name` qualifier

```
$ bin/pattoo_cli.py set language --code 'en' --name 'English (Jamaican)'
```

In this example we have changed the name to 'English (Jamaican)'

Creating Languages

To create a new language use the `bin/pattoo_cli.py create language` command.

```
$ bin/pattoo_cli.py create language --code 'es' --name 'Spanish'
```

In this case we create a new language with the name "Spanish" and identifying code "es"

3.1.3 Agents

As stated before, `pattoo agents` report data to the central `pattoo` server.

Viewing Agents

To view the agents posting data to the `pattoo` server use the `bin/pattoo_cli.py show agent` command.

```
$ bin/pattoo_cli.py show agent

idx_agent  agent_program                agent_target  enabled

1          pattoo_agent_snmp_ifmibd     localhost     1
2          pattoo_agent_snmpd           localhost     1
```

(continues on next page)

(continued from previous page)

```
3          pattoo_agent_os_autonomousd  nada          1
4          pattoo_agent_os_spoked       nada          1
```

Assigning Agents to Key-Pair Translations Groups

There are some important things to know first.

1. Each agent has an `idx_agent` number that can be seen in the first column of the `bin/pattoo_cli.py show agent` command. 1. Each agent group has an `idx_pair_xlate_group` number that can be seen in the first column of the `bin/pattoo_cli.py show key_translation` command.

To assign an `agent` to an `agent group` use the `bin/pattoo_cli.py assign agent` command.

```
$ bin/pattoo_cli.py assign agent --idx_agent 2 --idx_pair_xlate_group 4
```

In this case we have assigned agent with an `idx_agent` agent number of `2` to the `idx_pair_xlate_group` group number `4`

3.1.4 Key-Pair Translations

Agents only post key-value pairs to the `pattoo` server. Translations are short descriptions of what each key means. The aim is for you to see these descriptions instead of the keys when you look at `pattoo` data with the `pattoo-web` UI.

When a translation for a key reported by an `agent` is installed, the translation is seen in `pattoo-web` instead of the key itself. This makes `pattoo` data more meaningful.

You don't have to install translations for every `agent` that reports data. You just have to assign `agents` to `agent groups`, then you assign a single set of translations to the `agent group`.

Viewing Agent Group Key-Pair Translation Assignments

You can view these `agent group` to `translation group` assignments using the `bin/pattoo_cli.py show key_translation_group` command.

```
$ bin/pattoo_cli.py show key_translation_group

idx_pair_xlate_group  translation_group_name         enabled

1                     Pattoo Default                 1

2                     IfMIB Agents                   1

3                     OS Agents                      1
```

Creating Translation Groups

To create a new translation group use the `bin/pattoo_cli.py create key_translation` command.

```
$ bin/pattoo_cli.py create key_translation --name "Stock Market Symbol Translations"
```

In this case we create a new translation group with the name "Stock Market Symbol Translations"

Upadating Translation Group Names

You can update the name of a translation group using the `bin/pattoo_cli.py set key_translation_group` command. You must specify the group's `idx_pair_xlate_group` value and a name.

```
$ bin/pattoo_cli.py set key_translation_group --idx_pair_xlate_group 20 --name 'New␣
↪Translation Group Name'
```

In this example we have changed the name to 'New Translation Group Name' for `idx_pair_xlate_group` 20.

Viewing Agent Key-Pair Translation Groups

To view translation groups use the `bin/pattoo_cli.py show key_translation` command.

```
$ bin/pattoo_cli.py show key_translation

idx_pair_xlate_group  name            language  key                                    ␣
↪                  translation                                  units                ␣
↪ enabled

1                     Pattoo Default                                                  ␣
↪                                                                                     ␣
↪        1

2                     IfMIB Agents    en        pattoo_agent_snmp_ifmibd_ifalias     ␣
↪                  Interface Alias                                                  ␣
↪ 1
                                      en        pattoo_agent_snmp_ifmibd_ifdescr     ␣
↪                  Interface Description
                                      en        pattoo_agent_snmp_ifmibd_
↪ifhcinbroadcastpkts            Interface Broadcast Packets (HC inbound)   Packets /␣
↪Second
                                      en        pattoo_agent_snmp_ifmibd_
↪ifhcinmulticastpkts            Interface Multicast Packets (HC inbound)   Packets /␣
↪Second
...
...
...

3                     OS Agents       en        pattoo_agent_os_autonomousd_cpu_
↪frequency              CPU Frequency                              Frequency       ␣
↪       1
                                      en        pattoo_agent_os_autonomousd_cpu_stats_
↪ctx_switches      CPU (Context Switches)                  Events / Second
                                      en        pattoo_agent_os_autonomousd_cpu_stats_
↪interrupts        CPU (Context Switches)                  Events / Second
                                      en        pattoo_agent_os_autonomousd_cpu_stats_
↪soft_interrupts   CPU (Soft Interrupts)                   Events / Second
                                      en        pattoo_agent_os_autonomousd_cpu_stats_
↪syscalls          CPU (System Calls)                      Events / Second
```

Creating Agent Key-Pair Translation Group CSV Files

Creating a CSV key-pair translation file is easy. Follow these steps.

1. Make sure the first row has the following headings separated by commas.

```
language,key,translation,units
```

1. Each subsequent row must have values that correspond to the headings. Each value must be separated by a comma.

    1. The `language` must correspond to the language configured in your `pattoo.yaml` configuration file. `pattoo-web` will only evaluate translation entries that match to the configured language.
    1. The `key` value must correspond to any expected keys from key-value pairs reported by an agent.
    1. The `translation` must correspond to the brief text you want to use to describe the `key` 1. The `units` value is used to let users know the unit of measure to be used for the data being tracked by the `key`

```
language,key,translation,units
en,pattoo_agent_os_spoked_disk_io_write_bytes,Disk I/O (Bytes␣
↪Written),Bytes / Second
en,pattoo_agent_os_spoked_disk_io_write_count,Disk I/O (Write␣
↪Count),Writes / Second
en,pattoo_agent_os_spoked_disk_io_write_merged_count,Disk I/O␣
↪(Write Merged Count),Writes / Second
en,pattoo_agent_os_spoked_disk_io_write_time,Disk I/O (Write Time),
en,pattoo_agent_os_spoked_disk_partition,Disk Partition,
en,pattoo_agent_os_spoked_disk_partition_device,Disk Partition,
```

    Not all key-value pairs will need `units`. For example, `agent` metadata won't have them. In this case don't put a value for `units` and end the line with a comma (`,`). The previous example shows three lines of translations including `units` followed by three without `units`.

Importing Agent Key-Pair Translation Group Files

There are some important things to know first.

1. Each `translation group` has an `idx_pair_xlate_group` number that can be seen in the first column of the `bin/pattoo_cli.py show key_translation_group` command. 1. The translations for the `translation group` must be in a CSV file formatted according to the guidelines mentioned previously.

To import a translation file's data and assign it to a `translation group` use the `bin/pattoo_cli.py import key_translation` command.

```
$ bin/pattoo_cli.py import key_translation --idx_pair_xlate_group 7 --filename agent_
↪name_1_english.csv
```

In this case we have imported translations from a file named `agent_name_1_english.csv` and assigned it to a `translation group` with an `idx_pair_xlate_group` number of `7`.

You only need to import translations for the key-pairs you require. Any previously existing translation for an key-pair configured in the file will be updated. key-pairs not in the file will not be updated.

3.1.5 Agent Translations

Not only do an agent's key-pairs need translations, but the agents themselves need translations too. This is because an `agent` only reports its name when posting which, through translations, allows `pattoo` to be more flexible in supporting many different spoken languages.

Without translations, all references to a `pattoo` agent will just be by its name, which could be confusing.

Viewing Agent Translations

To view agent translations use the `bin/pattoo_cli.py show agent_translation` command.

```
$ bin/pattoo_cli.py show agent_translation

language  agent_program                translation                          enabled

en        pattoo_agent_os_autonomousd  Pattoo Standard OS Autonomous Agent  1
          pattoo_agent_os_spoked       Pattoo Standard OS Spoked Agent
          pattoo_agent_snmpd           Pattoo Standard SNMP Agent
          pattoo_agent_snmp_ifmibd     Pattoo Standard IfMIB SNMP Agent
          pattoo_agent_modbustcpd      Pattoo Standard Modbus TCP Agent
          pattoo_agent_bacnetipd       Pattoo Standard BACnet IP Agent
```

Creating Agent Translation CSV Files

Creating a CSV agent translation file is easy. Follow these steps.

1. Make sure the first row has the following headings separated by commas.

   ```
   language,key,translation
   ```

1. Each subsequent row must have values that correspond to the headings. Each value must be separated by a comma.

   1. The `language` must correspond to the language configured in your `pattoo.yaml` configuration file. `pattoo-web` will only evaluate translation entries that match to the configured language.
   1. The `key` value must correspond to the name of an agent. 1. The `translation` must correspond to the brief text you want to use to describe the `key`

```
language,key,translation
en,pattoo_agent_os_autonomousd,Pattoo Standard OS Autonomous Agent
en,pattoo_agent_os_spoked,Pattoo Standard OS Spoked Agent
en,pattoo_agent_snmpd,Pattoo Standard SNMP Agent
en,pattoo_agent_snmp_ifmibd,Pattoo Standard IfMIB SNMP Agent
en,pattoo_agent_modbustcpd,Pattoo Standard Modbus TCP Agent
en,pattoo_agent_bacnetipd,Pattoo Standard BACnet IP Agent
```

Importing Agent Translation Files

To import an agent translation file's data use the `bin/pattoo_cli.py import agent_translation` command.

```
$ bin/pattoo_cli.py import agent_translation --filename agent_name_translation_
↪english.csv
```

In this case we have imported translations from a file named `agent_name_translation_english.csv`.

You only need to import translations for the `agents` you require. Any previously existing translation for an `agent` configured in the file will be updated. `agents` not in the file will not be updated.

CHAPTER 4

Testing GraphQL Queries

Developer testing tools.

4.1 GraphQL API

You can use the *pattoo* API to retrieve data using a GraphQL interface. It's best to become familiar with GraphQL before reading further.

After completing this tutorial you'll be able to do programmatic GraphQL queries.

4.1.1 Queries with GraphQL

By default the `pattoo` server will run on port TCP 20202.

Interactive GraphQL

Interactive GraphQL allows you to test your queries using your web browser.

If you are running it on your local machine go to the http://localhost:20202/pattoo/api/v1/web/igraphql to see the interactive query tool.

Non Interactive GraphQL

If you want to access GraphQL programmatically, without using your browser then you'll need to access the non-interactive GraphQL URL.

If you are running it on your local machine go to the http://localhost:20202/pattoo/api/v1/web/graphql URL to get your results.

Retrieving GraphQL data with Pattoo-Web

You can use the *get* function in this file to get GraphQL data from the pattoo API server. https://github.com/PalisadoesFoundation/pattoo-web/blob/master/pattoo_web/phttp.py

4.1.2 How The Database Maps to GraphQL Queries

Note This section is very detailed, but it will help you with understanding how the GraphQL keywords required for your queries were created.

There are two important files in the repository's pattoo/db directory.

1. *models.py*: Defines the database structure using the python SQLAlchemy package
2. *schema.py*: Maps the database structure from SQLAlchemy to GraphQL queries using the graphene-sqlalchemy package.

Models.py

This file defines the tables and columns in the database.

1. Each class defines a table
2. Each variable in the class defines the columns. The variable name is the column name

The python *graphene-sqlalchemy* package used to present GraphQL will convert column names into camelCase, removing any underscores. Therefore a column named *idx_datapoint* will be *idxDatapoint* in your GraphQL queries.

You will notice some tables will have foreign keys as part of the RDBMS structure. Here is an example in the AgentXlate table:

```
class AgentXlate(BASE):
    """Class defining the pt_agent_xlate table of the database."""

    __tablename__ = 'pt_agent_xlate'
    __table_args__ = (
        UniqueConstraint('idx_language', 'agent_program'),
        {'mysql_engine': 'InnoDB'}
    )

    idx_agent_xlate = Column(
        BIGINT(unsigned=True), primary_key=True,
        autoincrement=True, nullable=False)

    idx_language = Column(
        BIGINT(unsigned=True),
        ForeignKey('pt_language.idx_language'),
        index=True, nullable=False, server_default='1')
```

You will also notice that this class also has a backref relationship near the bottom. This is what *graphene-sqlalchemy* uses to track the relationships for queries. In this case, the backref has the name *Agent_xlate_language* which will be converted to *agentXlateLanguage* camelCase in your GraphQL queries

```
language = relationship(
    Language,
    backref=backref(
        'Agent_xlate_language', uselist=True,cascade='delete,all'))
```

Schemas.py

This file contains the mappings from SQLAlchemy table definitions to GraphQL queries.

1. Database tables defined as SQLAlchemy classes in models.py are imported as *Model* classes in this file.
2. You'll notice that if you manually type in your GraphQL queries in the */igraphql* URL that you'll see lists of each available table column with explanations. These explanations are defined in the *Attribute* classes in this file.
3. Attributes and models are tied together in the *SQLAlchemyObjectType* classes.

```
from pattoo.db.models import AgentXlate as AgentXlateModel

class InstrumentedQuery(SQLAlchemyConnectionField):
    """Class to allow GraphQL filtering by SQlAlchemycolumn name."""

    def __init__(self, type_, **kwargs):
        ...
        ...
        ...


class AgentXlateAttribute():
    """Descriptive attributes of the AgentXlate table.
    A generic class to mutualize description of attributes for both queries
    and mutations.
    """

    idx_agent_xlate = graphene.String(
        description='AgentXlate table index.')

    idx_language = graphene.String(
        description='Language table index (ForeignKey).')

    agent_program = graphene.String(
        resolver=resolve_agent_program,
        description=('Agent progam'))

    translation = graphene.String(
        resolver=resolve_translation,
        description='Translation of the agent program name.')

    enabled = graphene.String(
        description='"True" if enabled.')


class AgentXlate(SQLAlchemyObjectType, AgentXlateAttribute):
    """AgentXlate node."""

    class Meta:
        """Define the metadata."""

        model = AgentXlateModel
        interfaces = (graphene.relay.Node,)
```

Next we'll discuss the *Query* class you'll find further down the file. This class:

1. **Uses the *InstrumentedQuery* class to filter queries by database column values. This *InstrumentedQuery* class makes things**

1. Every row of every database table has a fixed unique automatically generated GraphQL ID which is a *graphene.relay.node.GlobalID* object. You can filter specifically on this ID.
2. You also get lists of database row results containing the first X and last X rows.
3. Lists of database row results can also be obtained for values before and/or after X GraphQL ID values retrieved from a database table.
4. Custom filtering for specific values in a database column can be using resolvers, but you have to manually create a resolver for each table's column. This per query customization is not ideal.

2. Has *Node* entries for single value GraphQL queries, or as a definition inside an "edges" section of a GraphQL query. You can filter Nodes by the GraphQL *graphene.relay.node.GlobalID* too. This will be shown later.

```
class Query(graphene.ObjectType):
    """Define GraphQL queries."""

    node = relay.Node.Field()

    # Results as a single entry filtered by 'id' and as a list
    agent_xlate = graphene.relay.Node.Field(AgentXlate)
    all_agent_xlate = InstrumentedQuery(AgentXlate)
```

4.1.3 Query Examples

Here are some query examples using the example database table we have been using. Run these queries in the /igraphql url.

Note:

1. In all the examples in this section the "id" represents the *graphene.relay.node.GlobalID* string. You can use this to get information on a specific row of a specific table.
2. The *InstrumentedQuery* related queries in the Query class can only filter on a database table value, not the *graphene.relay.node.GlobalID* string.

Agent Table Queries

This section covers Agent table queries.

All Known Agents

This will provide information on all the known polling agents.

The agentProgram value will be used later for getting a translation into a meaningful name.

```
{
  allAgent {
    edges {
      node {
        id
        idxAgent
        agentPolledTarget
        agentProgram
      }
    }
```

(continues on next page)

(continued from previous page)

```
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
  }
}
```

All Datapoints Polled by Agent where id = "X"

You'll notice that this query also gives you the following information that will be required for translations later on: #. key-value pair *key* value for translating Datapoint metadata #. *agentProgram* for translating the program name into something meaningful #. *idxPairXlateGroup* for translating the key values

```
{
  agent(id: "QWdlbnQ6Mg==") {
    datapointAgent {
      edges {
        cursor
        node {
          id
          idxDatapoint
          idxAgent
          agent {
            agentProgram
            agentPolledTarget
            idxPairXlateGroup
            pairXlateGroup {
              id
            }
          }
          glueDatapoint {
            edges {
              node {
                pair {
                  key
                  value
                }
              }
            }
          }
        }
      }
      pageInfo {
        startCursor
        endCursor
        hasNextPage
        hasPreviousPage
      }
    }
  }
}
```

All Charts in which Datapoints Polled by Agent appear. Where id = “X”

This query will show:

1. All Datapoints for an Agent
2. The charts to which each datapoint belongs
3. The favorites to which the charts belong

```
{
  agent(id: "QWdlbnQ6MQ==") {
    datapointAgent {
      edges {
        cursor
        node {
          id
          idxDatapoint
          idxAgent
          chartDatapointDatapoint {
            edges {
              node {
                idxChartDatapoint
                chart {
                  id
                  idxChart
                  name
                  checksum
                  favoriteChart {
                    edges {
                      node {
                        idxFavorite
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
      pageInfo {
        startCursor
        endCursor
        hasNextPage
        hasPreviousPage
      }
    }
  }
}
```

DataPoint Table Queries

Here we have some representative queries you can do:

View All DataPoints

To see all DataPoints and their data enter this query on the left hand side of the viewer.

```
{
  allDatapoints {
    edges {
      node {
        id
                              idxDatapoint
        checksum
        dataType
        lastTimestamp
        pollingInterval
        enabled
      }
    }
  }
}
```

Sample Result

Here is the result of all DataPoints. Take note of `(id: "RGF0YVBvaW50OjE=")` as we'll use it for querying timeseries data.


```
{
  "data": {
    "allDatapoints": {
      "edges": [
        {
          "node": {
            "id": "RGF0YVBvaW50OjE=",
            "idxDatapoint": "1",
            "checksum":
↪"ea5ee349b38fa7dc195b3689872c8487e7696201407ef27231b19be837fbc6da0847f5227f1813d893100802c70ffb1864
↪",
            "dataType": 99,
            "lastTimestamp": 1575174588079,
            "pollingInterval": 10000,
            "enabled": "1"
          }
        },
        {
          "node": {
            "id": "RGF0YVBvaW50OjI=",
            "idxDatapoint": "2",
            "checksum":
↪"2b15d147330183c49a1672790bf09f54f8e849f9391c82385fd8758204e87940ab1ffef1bb67ac725de7cc0aa6aba9b6ba
↪",
            "dataType": 99,
            "lastTimestamp": 1575174588084,
            "pollingInterval": 10000,
            "enabled": "1"
          }
        }
      ]
```


(continues on next page)

(continued from previous page)

```
    }
  }
}
```

Pair Table Queries

Here we have some representative queries you can do:

View All Key-Pair-Values

To see all Key-Pair-Values enter this query on the left hand side of the viewer.

```
{
  allPairs {
    edges {
      node {
        id
        idxPair
        key
        value
      }
    }
  }
}
```

Sample Result

Here is the result of all Key-Pair-Values.

```
{
  "data": {
    "allPairs": {
      "edges": [
        {
          "node": {
            "id": "UGFpcjox",
            "idxPair": "1",
            "key":  "pattoo_agent_hostname",
            "value":  "palisadoes"
          }
        },
        {
          "node": {
            "id": "UGFpcjoy",
            "idxPair": "2",
            "key":  "pattoo_agent_id",
            "value":
↪"23a224313e4aaa4678a81638025ab02b42cb8a5b7c47b3dd2efced06d1a13d39"
          }
        },
        {
          "node": {
```

(continues on next page)

(continued from previous page)


```
            "id": "UGFpcjoz",
            "idxPair": "3",
            "key":  "pattoo_agent_polled_device",
            "value":  "device.example.com"
          }
        },
        {
          "node": {
            "id": "UGFpcjo0",
            "idxPair": "4",
            "key":  "pattoo_agent_program",
            "value":  "pattoo_agent_modbustcpd"
          }
        }
      ]
    }
  }
}
```


Glue Table Queries

Here we have some representative queries you can do:

View All GluePoints

To see all GluePoints enter this query on the left hand side of the viewer. This table maps all the key-value pairs associated with an individual DataPoint

```
{
  allGlues {
    edges {
      node {
        id
        idxPair
        idxDatapoint
      }
    }
  }
}
```

Sample Result


```
{
  "data": {
    "allGlues": {
      "edges": [
        {
          "node": {
            "id": "R2x1ZTooMSwgMSk=",
            "idxPair": "1",
            "idxDatapoint": "1"
```


(continues on next page)

(continued from previous page)


```
          }
        },
        {
          "node": {
            "id": "R2x1ZTooMSwgMik=",
            "idxPair": "1",
            "idxDatapoint": "2"
          }
        },
        {
          "node": {
            "id": "R2x1ZTooMSwgMyk=",
            "idxPair": "1",
            "idxDatapoint": "3"
          }
        },
        {
          "node": {
            "id": "R2x1ZTooMSwgNCk=",
            "idxPair": "1",
            "idxDatapoint": "4"
          }
        }
      ]
    }
  }
}
```


Data Table Queries

Here we have some representative queries you can do:

View All Numeric Timeseries Data for DataPoint id "x"

To see all numeric data for a specific datapoint `(id: "RGF0YVBvaW50OjE=")`, enter this query on the left hand side of the viewer.

```
{
  datapoint(id: "RGF0YVBvaW50OjE=") {
    id
    idxDatapoint
    checksum
    dataType
    pollingInterval
    dataChecksum {
      edges {
        node {
          id
          timestamp
          value
        }
      }
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Sample Result

Here is all the timeseries data from `(id: "RGF0YVBvaW50OjE=")`.


```
{
  "data": {
    "datapoint": {
      "id": "RGF0YVBvaW50OjE=",
      "idxDatapoint": "1",
      "checksum":
↪"ea5ee349b38fa7dc195b3689872c8487e7696201407ef27231b19be837fbc6da0847f5227f1813d893100802c70ffb1864
↪",
      "dataType": 99,
      "pollingInterval": 10000,
      "dataChecksum": {
        "edges": [
          {
            "node": {
              "id": "RGF0YTooMSwgMTU3NTE3MjgzNTAyOCk=",
              "timestamp": "1575172835028",
              "value": "738.0000000000"
            }
          },
          {
            "node": {
              "id": "RGF0YTooMSwgMTU3NTE3Mjg0NTIxOSk=",
              "timestamp": "1575172845219",
              "value": "738.0000000000"
            }
          },
          {
            "node": {
              "id": "RGF0YTooMSwgMTU3NTE3Mjg1NTM2NCk=",
              "timestamp": "1575172855364",
              "value": "738.0000000000"
            }
          }
        ]
      }
    }
  }
}
```


Language Table Queries

This query provides all the configured languages. The *code* returned is the language code. In the results, a code of *en* is english. Make translation queries based on this code value.

```
{
  allLanguage {
```

(continues on next page)

(continued from previous page)

```
    edges {
      node {
        id
        idxLanguage
        code
        name
      }
    }
  }
}
```

Agent Translation Table Queries

This section outlines how to view Agent translation data.

All Agent Translation Table Entries

You can use this query to get the translation for an agentProgram name for a specific language.This is useful for the home page.

```
{
  allAgentXlate {
    edges {
      node {
        id
        idxAgentXlate
        idxLanguage
        agentProgram
        translation
        enabled
        tsCreated
        tsModified
        language {
          id
          name
          code
          idxLanguage
        }
      }
    }
  }
}
```

Translation for a Specific agentProgram (all Languages)

In this case we get translations for the *agentProgram* named *pattoo_agent_snmp_ifmibd*.

```
{
  allAgentXlate(agentProgram: "pattoo_agent_snmp_ifmibd") {
    edges {
      node {
        id
```

(continues on next page)

(continued from previous page)

```
        idxAgentXlate
        idxLanguage
        agentProgram
        translation
        enabled
        tsCreated
        tsModified
      }
    }
  }
}
```

Single Node from Agent Translation table filtered by an ID

In this case:

1. The ID is a *graphene.relay.node.GlobalID* string.
2. The translation for the *agentProgram* is in the "translation" field.

```
{
  agentXlate(id: "QWdlbnRYbGF0ZToy") {
    id
    idxAgentXlate
    idxLanguage
    agentProgram
    translation
    enabled
    tsCreated
    tsModified
  }
}
```

Filtered Agent Translation table entry with Language where idxAgentXlate = "4"

There are some things to note:

1. This will provide a list of translations for all configured languages. The translation for the agentProgram is in the "translation" field.
2. Normally you'd be able to filter by "id" with GraphQL. Unfortunately this capability was lost when we added the customized ability to filter by any database table column. Hopefully the Python Graphene (GraphQL) team will be able to fix this later as part of their standard build.

```
{
  allAgentXlate(idxAgentXlate: "4") {
    edges {
      node {
        id
        idxAgentXlate
        idxLanguage
        agentProgram
        translation
        enabled
```

(continues on next page)

(continued from previous page)

```
        tsCreated
        tsModified
        language {
          id
          name
        }
      }
    }
  }
}
```

Key-Pair Translation Queries

This section outlines how to view key-pair translation data.

View all key-pair Translations

Here's the query you'll need to view all translations:

```
{
  allPairXlate {
    edges {
      node {
        id
        idxLanguage
        idxPairXlate
        idxPairXlateGroup
        key
        translation
      }
    }
  }
}
```

View key-pair Translations for idxPairXlateGroup = "x"

In this example, we filter by *idxPairXlateGroup*

```
{
  allPairXlate (idxPairXlateGroup: "2"){
    edges {
      node {
        id
        idxLanguage
        idxPairXlate
        idxPairXlateGroup
        key
        translation
      }
    }
  }
}
```

Favorites Table Queries

This section outlines how to view favorites data.

View all Favorites and Their Assigned Charts

This is the query string you'll need to see all the favorites in the database.

```
{
  allFavorite {
    edges {
      node {
        id
        idxFavorite
        order
        user {
          id
          idxUser
          username
          firstName
          lastName
        }
        chart {
          name
          chartDatapointChart {
            edges {
              node {
                idxDatapoint
              }
            }
          }
        }
      }
    }
  }
}
```

User Table Queries

This section outlines how to view favorites data.

View all Favorites for All Users

This query will show:

1. All users
2. Their favorites
3. The charts associated with each favorite

```
{
  allUser {
    edges {
```

(continues on next page)

(continued from previous page)

```
      node {
        id
        username
        firstName
        lastName
        enabled
        favoriteUser {
          edges {
            node {
              order
              chart {
                id
                idxChart
                name
              }
            }
          }
        }
      }
    }
  }
}
```

View all Favorites for a Specific User (by filter other than ID)

This query will show:

1. The filtered username ("pattoo")
2. Its favorites
3. The charts associated with each favorite

```
{
  allUser(username: "pattoo") {
    edges {
      node {
        id
        username
        favoriteUser {
          edges {
            node {
              order
              chart {
                id
                idxChart
                name
              }
            }
          }
        }
      }
    }
  }
}
```

View all Favorites for a Specific User (by ID)

This query will show:

1. The user
2. Its favorites
3. The charts associated with each favorite

```
{
  user(id: "VXNlcjox") {
    id
    username
    favoriteUser {
      edges {
        node {
          order
          chart {
            id
            idxChart
            name
          }
        }
      }
    }
  }
}
```

Authenticate Username and Password

This is a custom query that requires you enter a username and password. Regular query results are returned when found, a Null result is returned upon failure.

```
{
  authenticate(username: "palisadoes@example.org", password: "123456") {
    id
  }
}
```

Result

Results are returned when found.

```
{
  "data": {
    "authenticate": [
      {
        "id": "VXNlcjo3"
      }
    ]
  }
}
```

A Null result is returned when not found.

```
{
  "data": {
    "authenticate": null
  }
}
```

Pagination

This section outlines how to do simple pagination

View all Datapoints

This query will return all Datapoint values.

```
{
  allDatapoints {
    edges {
      node {
        idxDatapoint
        idxAgent
        id
        tsCreated
        tsModified
      }
    }
  }
}
```

View First X Datapoints

It's important to note the *startCursor* and *endCursor* values when wanting to paginate. They are useful in subsequent queries where you may want ranges of values that are not relative to the very start and very end of database table rows.

```
{
  allDatapoints(first: x) {
    edges {
      node {
        idxDatapoint
        idxAgent
        id
        tsCreated
        tsModified
      }
    }
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
  }
}
```

View Last X Datapoints

It's important to note the *startCursor* and *endCursor* values when wanting to paginate. They are useful in subsequent queries where you may want ranges of values that are not relative to the very start and very end of database table rows.

```
{
  allDatapoints(last: x) {
    edges {
      node {
        idxDatapoint
        idxAgent
        id
        tsCreated
        tsModified
      }
    }
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
  }
}
```

Next X Datapoints

Note:

1. It's important to note the *endCursor* of the previous query.
2. The next X results would need a query like the one below, starting at the *endCursor* value of the previous query.

```
{
  allDatapoints(first: X, after: "END_CURSOR_VALUE") {
    edges {
      node {
        idxDatapoint
        idxAgent
        id
        tsCreated
        tsModified
      }
    }
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
  }
}
```

Previous X Datapoints

Note:

1. It's important to note the startCursor of the previous query.
2. The previous X results would need a query like the one below, starting at the *startCursor* value of the previous query.

```
{
  allDatapoints(last: X, before: "START_CURSOR_VALUE") {
    edges {
      node {
        idxDatapoint
        idxAgent
        id
        tsCreated
        tsModified
      }
    }
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
  }
}
```

4.1.4 Mutation Examples

Mutation is the terminology that GraphQL uses for database updates. Here are some query examples using the example database table we have been using. Run these queries in the */igraphql* url.

Chart Table Mutation

This section outlines how to mutate chart data.

Add a New Chart

This mutation will add the chart then return the resulting fields:

1. *id*
2. *name*
3. Enabled status

Mutation

```
mutation {
  createChart(Input: {name: "Flying Fish"}) {
    chart {
```

(continues on next page)

(continued from previous page)

```
      id
      name
      enabled
    }
  }
}
```

Result

```
{
  "data": {
    "createChart": {
      "chart": {
        "id": "Q2hhcnQ6MjM5",
        "name": "Flying Fish",
        "enabled": "1"
      }
    }
  }
}
```

Modify Chart Name

This mutation will change the chart *name* from “Flying Fish” to “Teddy Bear”:

Mutation

```
mutation {
  updateChart(Input: {idxChart: "239", name: "Teddy Bear"}) {
    chart {
      id
      name
      enabled
    }
  }
}
```

Result

```
{
  "data": {
    "updateChart": {
      "chart": {
        "id": "Q2hhcnQ6MjM5",
        "name": "Teddy Bear",
        "enabled": "1"
      }
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

ChartDataPoint Table Mutation

This section outlines how to mutate ChartDataPoint data.

Add a New ChartDataPoint

This mutation will add a *DataPoint* to an existing chart then return the resulting fields:

Mutation

```
mutation {
  createChartDataPoint(Input: {idxDatapoint: "3", idxChart: "239"}) {
    chartDatapoint {
      id
      idxChartDatapoint
      idxDatapoint
      idxChart
    }
  }
}
```

Result

```
{
  "data": {
    "createChartDataPoint": {
      "chartDatapoint": {
        "id": "Q2hhcnREYXRhUG9pbnQ6MjQy",
        "idxChartDatapoint": "242",
        "idxDatapoint": "3",
        "idxChart": "239"
      }
    }
  }
}
```

Modify ChartDataPoint Name

This mutation will remove a DataPoint from the ChartDataPoint entry (Disable the entry for the chart):

Mutation

```
mutation {
  updateChartDataPoint(Input: {idxChartDatapoint: "242", enabled: "0"}) {
    chartDatapoint {
      id
      idxChartDatapoint
      idxDatapoint
      idxChart
      enabled
    }
  }
}
```

Result

```
 {
   "data": {
     "updateChartDataPoint": {
       "chartDatapoint": {
         "id": "Q2hhcnREYXRhUG9pbnQ6MjQy",
         "idxChartDatapoint": "242",
         "idxDatapoint": "3",
         "idxChart": "239",
         "enabled": "0"
       }
     }
   }
}
```

User Table Mutation

This section outlines how to mutate user data.

Add a New User

This mutation will add a User then return the resulting fields:

Mutation

```
mutation {
  createUser(Input: {username: "foo@example.org", firstName: "Foo", lastName: "Fighter
↪", password: "123456"}) {
    user {
      id
      idxUser
      firstName
      lastName
      username
      enabled
    }
  }
}
```

Result

```
{
  "data": {
    "createUser": {
      "user": {
        "id": "VXNlcjoz",
        "idxUser": "3",
        "firstName": "Foo",
        "lastName": "Fighter",
        "username": "foo@example.org",
        "enabled": "1"
      }
    }
  }
}
```

Modify User FirstName

This mutation will remove a DataPoint from the ChartDataPoint entry (Disable the entry for the chart):

Mutation

```
mutation {
  updateUser(Input: {idxUser: "3", firstName: "Street"}) {
    user {
      idxUser
      firstName
      lastName
      username
      enabled
    }
  }
}
```

Result

```
{
  "data": {
    "updateUser": {
      "user": {
        "idxUser": "3",
        "firstName": "Street",
        "lastName": "Fighter",
        "username": "foo@example.org",
        "enabled": "1"
      }
    }
  }
}
```

Favorite Table Mutation

This section outlines how to mutate favorite data.

Add a New Favorite

This mutation will add a Favorite then return the resulting fields:

Mutation

```
mutation {
  createFavorite(Input: {idxUser: "3", idxChart: "149", order: "2"}) {
    favorite{
      id
      idxFavorite
      idxChart
      idxUser
      enabled
    }
  }
}
```

Result

```
{
  "data": {
    "createFavorite": {
      "favorite": {
        "id": "RmF2b3JpdGU6Mg==",
        "idxFavorite": "2",
        "idxChart": "149",
        "idxUser": "3",
        "enabled": "1"
      }
    }
  }
}
```

Modify Favorite

This mutation will remove the *Favorite* entry (Disable the entry):

Mutation

```
mutation {
  updateFavorite(Input: {idxFavorite: "2", enabled: "0"}) {
    favorite {
      idxFavorite
      idxChart
```

(continues on next page)

(continued from previous page)

```
      idxUser
      enabled
    }
  }
}
```

Result

```
{
  "data": {
    "updateFavorite": {
      "favorite": {
        "idxFavorite": "2",
        "idxChart": "149",
        "idxUser": "3",
        "enabled": "0"
      }
    }
  }
}
```

4.2 Queries with REST

It's best to become familiar with REST before trying these tests.

Here are some things to remember.

1. By default the `pattoo` server will run on port TCP 20202.
2. If you are running it on your local machine the RESTful API URLs will all start with http://localhost:20202/pattoo/api/v1/web/rest. All the examples assume make reference to this fact. So if the uri `/data` is mentioned then assume we are referring to the complete URL of http://localhost:20202/pattoo/api/v1/web/rest/data

Let's begin.

4.2.1 Why a RESTful Interface?

We provide a RESTful interface for ease of comparison with GraphQL for a limited set of functions. There will be no further development of the RESTful beyond what is listed here. Do not write any `pattoo` related code based on REST. This feature is deprecated and will soon be removed.

4.2.2 Using the RESTful Interface (Deprecated)

View DataPoint data

To view data for generated by a specific DataPoint visit the `/data` URI. Add the `idx_datapoint` value to the end to get `/data/1` for `idx_datapoint` value of 1.

1. By default a week's worth of data is returned.
2. There is no ability to get data for all DataPoints simultaneously.

3. You can use the `?secondsago=X` query string to get data starting `X` seconds ago to the most recently stored data.

In this case we have data from `/data/1?secondsago=3600`


```
[
    {
        "1573619400" : 3878839847
    },
    {
        "1573619700" : 3879239629
    },
    {
        "1573620000" : 3879652192
    },
    {
        "1573620300" : 3880050372
    },
    {
        "1573620600" : 3880449410
    },
    {
        "1573620900" : 3880856015
    },
    {
        "1573621200" : 3881272430
    },
    {
        "1573621500" : 3881704477
    },
    {
        "1573621800" : 3882250116
    },
    {
        "1573622100" : 3882650025
    },
    {
        "1573622400" : 3883064936
    }
]
```

CHAPTER 5

Miscellaneous Information

Technical background information on the project.

5.1 Creating `pattoo` Agents

Documentation on how to create your own custom agents can be found here.

5.2 Performance and Troubleshooting

Performance tuning and troubleshooting are related. So we created a page for both!

5.2.1 Troubleshooting

Troubleshooting steps can be found in the PattooShared troubleshooting documentation

5.2.2 `pattoo` Performance Tuning

There are a number of ways to improve `pattoo` performance.

Use RAM disks for Caching

We have seen a 10X improvement in the `pattoo_ingester` records / second performance when using a RAM disk versus SSDs. We recommend using RAM disks and SSDs for your cache directory over regular hard drives in production environments.

Run the `pattoo_ingester` More Frequently

The `pattoo_ingester` needs to run periodically to import agent data files into the database. You want to ensure that it can keep up with this task. Check your logs to make sure that the completion time of each `pattoo_ingester` run is less than the configured `polling_interval`. Increase the `crontab` frequency if it isn't.

Database Performance Improvements

The `pattoo_ingester` makes many connections to the database. You have a number of options if it crashes. Check your logs for the cause of errors to help you choose the best corrective action.

1. If the errors state that you have too many connections, then increase `max_connections` value in the server configuration file. The default is 200. Try 300 and increase as needed.

```
[mysqld]
max_connections = 300
```

2. If the errors mention `pool_size` or `max_overflow`, then edit your configuration file and adjust those values.

```
db_pool_size: 10
db_max_overflow: 10
```

Reduce Logging

The default `pattoo debug` logging level can create large files. This is done to make it easier to troubleshoot the initial installation. Set the level to `info` for most scenarios.

5.3 JSON Formatting for pattoo-agents

JSON data formatting can be found in the Pattoo Shared data documentation

5.4 Pattoo Terminology

A complete glossary of terms can be found in the Pattoo Shared glossary documentation

CHAPTER 6

Developers

6.1 How To Contribute

Start contributing today!

6.1.1 Introduction

Below is the workflow for having your contribution accepted into the `pattoo` repository.

1. Create an Issue or comment on an existing issue to discuss the feature
2. If the feature is approved, assign the issue to yourself
3. Fork the project
4. Clone the fork to your local machine
5. Add the original project as a remote (git remote add upstream https://github.com/PalisadoesFoundation/pattoo, check with: git remote -v)
6. Create a topic branch for your change (git checkout -b `BranchName`)
7. you may create additional branches if modifying multiple parts of the code
8. Write code and Commit your changes locally. An example of a proper `git commit` message can be seen below:

9. When you need to synch with upstream (pull the latest changes from main repo into your current branch), do:
    1. `git fetch upstream`
    2. `git merge upstream/master`
10. Check for unnecessary white space with `git diff --check`.
11. Write the necessary unit tests for your changes.

12. Run all the tests to assure nothing else was accidentally broken
13. Push your changes to your forked repository (git push origin `branch`)
14. Perform a pull request on GitHub
15. Your code will be reviewed
16. If your code passes review, your pull request will be accepted

6.1.2 Code Style Guide

For ease of readability and maintainability code for all `pattoo` projects must follow these guidelines. Code that does not comply will not be added to the `master` branch.

1. All `pattoo` projects use the Google Python Style Guide for general style requirements
2. All `pattoo` python projects use the The Chromium Projects Python Style Guidelines for docstrings.
3. Indentations must be multiples of 4 blank spaces. No tabs.
4. All strings must be enclosed in single quotes
5. In addition too being `pylint` compliant, the code must be PEP8 and PEP257 compliant too.
6. There should be no trailing spaces in files

Guidelines to remember

- Always opt for the most pythonic solution to a problem
- Avoid applying idioms from other programming languages
- Import each module with its full path name. ie: from pack.subpack import module
- Use exceptions where appropriate
- Use doc strings
- Try not to have returns at multiple points in a function unless they are failure state returns.
- If you are in the middle of a development session and have to interrupt your work, it is a good idea to write a broken unit test about what you want to develop next. When coming back to work, you will have a pointer to where you were and get back on track faster.

Commits

The `pattoo` projects strive to maintain a proper log of development through well structured git commits. The links below offer insight and advice on the topic of commit messages:

1. https://robots.thoughtbot.com/5-useful-tips-for-a-better-commit-message
2. http://chris.beams.io/posts/git-commit/

Sample .vimrc File for Compliance

You can use this sample .vimrc file to help meet our style requirements

6.2 Testing Your Code

Make sure you create your own unittests for all the classes, methods, and functions you have created or modified. Place them in the *tests/* directory in a subdirectory that matches the relative location of your production code under the *pattoo/* directory.

6.2.1 Create Unittest Database

Assuming you already have the *pattoo* database set up, these steps will copy the contents of the pattoo database to a new *pattoo_unittest* database so that it can be used by the local unittests.

:: $ mysqldump -u root -p pattoo > pattoo.sql Enter password: $ mysql -u root -p pattoo_unittest < pattoo.sql Enter password:

6.2.2 Database Setup

The unittests assume that a MySQL / MariaDB testing database has been created with the following parameters:

- *db_hostname*: localhost
- *db_username*: travis
- *db_password*: K2nJ8kFdthEbuwXE
- *db_name*: pattoo_unittest

Create a database and grant full privileges to the *travis* user.

```
# mysql
MariaDB [(none)]> CREATE DATABASE pattoo_unittest;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON pattoo_unittest.* TO 'travis'@'localhost'␣
↪identified by 'K2nJ8kFdthEbuwXE';
MariaDB [(none)]> FLUSH PRIVILEGES;
MariaDB [(none)]> exit;
#
```

6.2.3 Setting up Syslog Error Codes

pattoo uses unique error code numbers for syslog messages to make troubleshooting easier. Run the *tests/bin/error_code_report.py* script before the unittests to make there are no duplicates. The unittests will fail if there are duplicates.

```
$ tests/bin/error_code_report.py

Pattoo Logging Error Code Summary Report
----------------------------------------
Starting Code              : 20001
Ending Code                : 20141
Duplicate Codes to Resolve : []
Available Codes            : [20141, 20142, 20143, 20144, 20145]
$
```

Everything is OK if there are no *Duplicate Codes to Resolve*.

6.2.4 Running Tests

The *tests/bin/do_all_tests.py* script will recursively run all unittests in the *tests/* directory.