# Quartx Call Logger

***Release 0.3.0***

**Jun 27, 2020**

# Contents:

# CHAPTER 1

# Quartx Call Logger

Call logger component for the Quartx phone system monitoring frontend. https://quartx.ie/

This logger can monitor phone systems for CDR(Call Data Records) and send the records to the monitoring frontend. The monitoring frontend will then analyze the records and display them in a easy to view web interface.

The currently supported phone systems are:

- Siemens Hipath

Support for new phone systems can be easily added through plugins. With the plugin system any system can be supported as long as the system has some sort of API or Serial Interface. The documentation on how to create a plugin can be found here. https://quartx-call-logger.readthedocs.io/en/latest/?badge=latest.

CHAPTER 2

# Install

Currently only install from git repo is supported, but PyPI support will be added later.

Production

```
sudo pip3 install git+https://github.com/quartx-software/quartx-call-logger.git
```

Development

```
git clone https://github.com/quartx-software/quartx-call-logger.git
cd quartx-call-logger
pip install pipenv
pipenv install --dev
```

# CHAPTER 3

# Configuration

The Configuration for this package is located in `/Library/Application Support/quartx` on MacOS, `/etc/xdg/quartx` on Unix/Linux or `C:\ProgramData\quartx\quartx` on Windows.

First we download the base configuration file from github so we can modifiy it. The following commands are for Linux/Unix.

```
sudo mkdir -p /etc/xdg/quartx
sudo curl https://raw.githubusercontent.com/quartx-software/quartx-call-logger/master/
↪data/call-logger.yml > /etc/xdg/quartx/call-logger.yml
```

Currently the only required settings is the `token`. The token is the authentication key used to authenticate the user and identify who the call logs belong to. Contact Quartx Call Monitoring for the token key.

```
sudo nano /etc/xdg/quartx/call-logger.yml
...
token: bd6a567386f79329b156a042a1aac9f44726e736
...
```

The plugin settings may also need to be changed depending on the phone system. You can read the configuration comments to see what changes may be required.

CHAPTER 4

# Usage

The call logger can be run with just one single command when on Linux.

```
call-logger
```

To run the call-logger as a service you can install the systemd service file

```
sudo curl https://raw.githubusercontent.com/quartx-software/quartx-call-logger/master/
↪data/quartx-call-logger.service > /etc/systemd/system/call-logger.service
sudo systemctl enable --now call-logger.service
```

CHAPTER 5

# Contribution

Support for other phone systems can be added through plugins. Documentation for creating plugins can be found at readthedocs. https://quartx-call-logger.readthedocs.io/en/latest/?badge=latest.

## 5.1 Plugins

### 5.1.1 Crreating Plugin

To create a plugin you first need to create a new module in the plugins directory of the quartx_call_logger package. Then create a class that inherits from either *quartx_call_logger.plugins.Plugin* or *quartx_call_logger.plugins.SerialPlugin*.

When the call logger starts up it scans the plugin directory for plugins and registers them automatically.

There is one method that is required in the plugin class, "run". This method is the main entry point for the plugin. This method should be using a loop that checks the state of the plugin property `self.running`, and when true the loop should continue monitoring the call logs.

```
from . import Plugin, Record

class VoipService(Plugin):
    def run(self):
        while self.running:
            # Run code that monitors for call logs
            ...

            # Create a call record
            record = Record(Record.INCOMING, number="0876156584", line=2, ext=101)

            # Send the call record to the monitoring frontend
            self.push(record)
```

### 5.1.2 Settings

If there are any required settings for the plugin, they should be added to the `defaults.yml` configuration file. This file is located in `/quartx_call_logger/data/defaults.yml`. These settings are then passed to the plugin constructor whenever the plugin is initialized.

Example settings configuration:

```
SiemensHipathSerial:
  # Port & baud rate settings required to communicate with the Siemen Hipath serial␣
↪interface
  # port: The port where the serial device is located. e.g. /dev/ttyUSB0 on GNU/Linux␣
↪or COM3 on Windows
  # rate: Baud rate used when communicating with the serial interface, such as 9600
  port: /dev/ttyUSB0
  rate: 9600
```

**Note:** The name for the plugin settings needs to be the exact name given to the plugin class.

### 5.1.3 Record API

**class** quartx_call_logger.record.**Record**

This class is a dictionary like object.

Fields that are common to all call types.

- **call_type** (*int*) - The type of call record, incoming/received/outgoing
- **line** (*int*) - The line number that the call is on.
- **ext** (*int*) - The extention number that the call is on.
- **number** (*str*) - (*optional*) The phone number of the caller. If not givin, '+353000000000' is used.
- **date** (*datetime*) - (*optional*) The datetime of the call, optional but recommended.

Extra fields that are used for Received & Outgoing calls:

- **ring** (*int*) - (*optional*) The time in seconds that the caller was ringing for. Defaults = 0
- **duration** (*int*) - (*optional*) The duration of the call in seconds. Defaults = 0
- **answered** (*int/bool*) - (*optional*) Indicate if call was answered. Determined by duration if not given.

**Note:** **duration** & **ring** may also be in the format of `HH:MM:SS`.

**Note:** **date** must be in the ISO 8601 format e.g. `2019-08-11T01:49:49+00:00`. UTC is preferred.

There are 10 possible call types. Currently only the first 3 are processed, this will change in the future when we have more data to determine best way to process them.:

- **0** Incoming call.
- **1** Received call.
- **2** Outgoing call.

- **3** Received call (Other Service).
- **4** Outgoing call (Other Service).
- **5** Received call (Farwarded).
- **6** Outgoing call (Farwarded).
- **7** Received conference call.
- **8** Outgoing conference call.
- **9** Outgoing call Via Farwarded.

### 5.1.4 Plugin API

**class** quartx_call_logger.plugins.**Plugin**
This is the Base Plugin class for all phone system plugins.

**Note:** This class is not ment to be called directly, but subclassed by a Plugin.

**timeout = 10**
The timeout setting in seconds. Can be changed in the user config.

**timeout_decay = 1.5**
The timeout decay, used to increase the timeout after each failed connection. Can be changed in the user config.

**max_timeout = 300**
The max timeout in seconds, the timeout will not decay past this point. Can be changed in the user config.

**base_timeout = 10**
The base timeout value without decay

**logger = logging.Logger**
The logger object associated with this plugin

**push**(*record: quartx_call_logger.record.Record*) → NoReturn
Send a call log record to the call monitoring API.

**run**() → NoReturn
Main entry point for plugin. Must be overridden

**running**
Flag to indicate that everything is working and ready to keep monitoring.

**settings = <module 'quartx_call_logger.settings' from '/home/docs/checkouts/readthedoc**
Public settings

**class** quartx_call_logger.plugins.**SerialPlugin**(*port: str*, *rate: int*)
This is an extended plugin with serial interface support.

**Note:** This class is not ment to be called directly, but subclassed by a Plugin.

**Parameters**

- **port** – The port/path to the serial interface.
- **rate** – The serial baud rate to use.

**decode**(*data: bytes*) → str
Overide this method to handel decoding of serial data.

**Parameters** **data** – The raw data line from the serial interface as type `bytes`.

**Returns** The decoded into data line as type `str`.

**parse**(*data: str*) → quartx_call_logger.record.Record
Overide this method to handel parsing of serial data.

**Parameters** **data** – The decoded data line.

**Returns** A *quartx_call_logger.record.Record* object.

# Index