Sitch Sensor Documentation

Release 4.0

Ash Wilson

Jan 06, 2019

Contents:

Version 4.0

CHAPTER 1

Sensor Environment Variables

The SITCH Sensor requires some environment variables to be set in order to operate.

Environment Variable	Purpose
CGI_WHITELIST	(Optional) List of trusted CGIs.
FEED_RADIO_TARGETS	(Optional) Radio types to target for feed ingestion. Defaults to `GSM`
FEED_URL_BASE	(Optional) Base URL for Sensor feed. Defaults to SITCH auto-built public feed
GSM_MODEM_BAND	Restrict GSM modem to this band. Options: (EGSM_MODE | PGSM_MODE | DCS_MODE | GSM850_MODE | PCS_MODE | EGSM_DCS_MODE | GSM850_PCS_MODE | EGSM_PCS_MODE | ALL_BAND) Defaults to `ALL_BAND`
GSM_MODEM_PORT	(Optional) Set the tty for the GSM modem. If unset, the Sensor will attempt to auto-configure
KAL_BAND	Band for Kalibrate to scan. (GSM850 | GSM-R | GSM900 | EGSM | DCS | PCS) Defaults to `GSM850`
KAL_GAIN	Gain value for Kalibrate. Defaults to `60`
KAL_THRESHOLD	Hard threshold for Kalibrate channel power level. Defaults to `1000000`
LOCATION_NAME	Name of the location for this sensor. No spaces.
LOG_HOST	Logstash endpoint. Formatted like this: `hostname:port`
MCC_LIST	(Optional) List of Mobile Country Codes to ingest from feed. List is comma-separated: `310,311,316` Defaults to `310,311,312,316`
MODE	Set to `clutch` to go into a wait loop on start. Useful for troubleshooting. Defaults to `full`
NO_FEED_UPDATE	(Optional) If set, do not attempt to update the feed on boot.
STATE_LIST	Comma-separated list of states for feed ingestion. California and Texas would be: `CA,TX`
VAULT_PATH	Path to Logstash/Filebeat credentials in Vault. Defaults to `secret/client`, which will work with the demo environment.
VAULT_TOKEN	Client token used to retrieve credentials from Vault.
VAULT_URL	URL for Vault instance containing Logstash/Filebeat credentials. Looks like: `https://server.com:8200`
NO_FEED_UPDATE	If set, do not attempt to update the feed on boot.
GSM_MODEM_PORT	(Optional) GSM modem USB-TTY port. This should be autodetected and not need to be set. Looks like: `/dev/ttyUSB0` See: “Found but undetected TTY ” in the docs
GPS_DEVICE_PORT	(Optional) GPS device USB-TTY port. This should be autodetected and not need to be set. Looks like: `/dev/ttyUSB0` See: “Found but undetected TTY ” in the docs

CHAPTER 2

SITCH Sensor Alert Types

SITCH has a well-defined set of alerts, which are meant to be easy to parse with a log management or SIEM system.

The alert log message format is defined here: http://sensor.readthedocs.io/en/test/data.html#sitch-alert-log

The supported message types are listed here (in the __init__ function): http://sensor.readthedocs.io/en/test/_modules/sitchlib/alert_manager.html#AlertManager

CHAPTER 3

Understanding the Log Data Collected by Sitch

The following sections describe the data for the files found in '/data/sitch/log/'.

3.1 cells.log


```
{"scan_results": [
  {"cgi_str": "310:260:275:20000",
   "site_name": "sitch-site-testing",
   "bsic": "16",
   "mcc": "310",
   "rla": 0,
   "lac": "275",
   "band": "ALL_BAND",
   "feed_info": {
     "mcc": "310",
     "lon": "-122.464146",
     "lac": "275",
     "range": 325,
     "lat": "37.776641",
     "mnc": "260",
     "cellid": "20082"},
   "scan_location": "sitch-site-testing",
   "mnc": "260",
   "txp": 03,
   "distance": 534.3820159387475,
   "scan_finish": "2017-05-06T06:25:49.837957",
   "rxl": 20.0,
   "cell": 0,
   "scanner_public_ip": "1.1.1.1",
   "rxq": 0.0,
   "ta": 255,
   "cellid": "20082",
   "cgi_int": 31026027520082,
```


(continues on next page)

(continued from previous page)


```
  "arfcn": 684}
  ...],
 "band": "ALL_BAND",
 "site_name": "sitch-site-testing",
 "platform": "Unspecified",
 "scan_start": "",
 "scan_location": "sitch-site-testing",
 "scanner_public_ip": "1.1.1.1",
 "scan_finish": "2017-05-06T06:25:49.837957",
 "scan_program": "gsm_modem"
 "event_timestamp": "2017-05-06T06:25:49.837957"}
```


3.1.1 <cell>

possible values	description
0	The serving cell
1-6	The index of the neighboring cell

3.1.2 <arfcn>

[Absolute radio frequency channel number](https://en.wikipedia.org/wiki/Absolute_radio-frequency_channel_number)

3.1.3 <rxl>

Receive level

The measured signal level shall be mapped to an RXLEV value between 0 and 63, as follows:

possible values	description
0	less than -110 dBm.
1	-110 dBm to -109 dBm.
2	-109 dBm to -108 dBm.
...	
...	
62	-49 dBm to -48 dBm.
63	greater than -48 dBm.

3.1.4 <rxq>

Receive quality

possible values	description
0...7	as [RXQUAL](https://en.wikipedia.org/wiki/Rxqual) values
99	not known or not detectable

3.1.5 <mcc>

[Mobile country code](https://en.wikipedia.org/wiki/Mobile_country_code)

3.1.6 <mnc>

[Mobile network code](https://en.wikipedia.org/wiki/Mobile_country_code)

3.1.7 <bsic>

[Base station identity code](https://en.wikipedia.org/wiki/Base_station_identity_code)

3.1.8 <cellid>

[Cell id](https://en.wikipedia.org/wiki/Cell_ID)

NOTE: In a 7-item line, cellid is not provided. We set it to 0 to prevent barfing elsewhere.

3.1.9 <lac>

[Location area code](http://www.telecomabc.com/l/lac.html)

3.1.10 <rla>

Receive level access minimum

GUESS: Minimum receiving level permitted to access the system Per: similar AT engineering mode (AT+QENG) command in [M95 AT commands manual](http://eddywireless.com/yahoo_site_admin/assets/docs/M95_AT_Commands_Manual_V12.196112248.pdf)

3.1.11 <txp>

Transmit power maximum CCCH

3.1.12 <TA>

[Timing Advance](https://en.wikipedia.org/wiki/Timing_advance)

3.2 geoip.log


```
{"geometry":
  {"type": "Point",
   "coordinates": [-122, 37]
   },
 "scan_program": "geo_ip",
 "type": "Feature",
 "event_timestamp": "2017-05-06T06:25:49.837957"}
```

This is in geojson structure, with the addition of an event_timestamp field.

3.3 gps.log


```
{"sat_time": "2017-05-02T06:26:08.000Z",
 "geometry": {
   "type": "Point",
   "coordinates":
     [-122, 37]
     },
 "time_drift": 0.006355733333333334,
 "sys_time": "2017-05-02T06:26:08.381344",
 "scan_program": "gpsd",
 "type": "Feature"
 "event_timestamp": "2017-05-06T06:25:49.837957"}
```


3.4 gsm_modem_channel.log


```
{"cgi_str": "310:260:275:20082",
"site_name": "sitch-site-testing",
"bsic": "16",
"mcc": "310",
"rla": 8,
"lac": "275",
"band": "ALL_BAND",
"feed_info": {
    "mcc": "310",
    "lon": "-122.123",
    "lac": "275",
    "range": 325,
    "lat": "37.123",
    "mnc": "260",
    "cellid": "20082"
    },
"scan_location": "sitch-site-testing",
"mnc": "260",
"txp": 3,
"distance": 568.12345,
"scan_finish": "2017-05-16T02:21:23.901298",
"event_timestamp": "2017-05-16T02:21:23.901298",
"rxl": 24.0,
"cell": 0,
"scanner_public_ip": "1.1.1.1",
"rxq": 0.0,
"ta": 255,
"cellid": "20082",
"cgi_int": 31026027520082,
"arfcn": 684}
```

3.4.1 <cell>

possible values	description
0	The serving cell
1-6	The index of the neighboring cell

3.4.2 <arfcn>

[Absolute radio frequency channel number](https://en.wikipedia.org/wiki/Absolute_radio-frequency_channel_number)

3.4.3 <rxl>

Receive level

The measured signal level shall be mapped to an RXLEV value between 0 and 63, as follows:

possible values	description
0	less than -110 dBm.
1	-110 dBm to -109 dBm.
2	-109 dBm to -108 dBm.
. . .	
. . .	
62	-49 dBm to -48 dBm.
63	greater than -48 dBm.

3.4.4 <rxq>

Receive quality

possible values	description
0. . . 7	as [RXQUAL](https://en.wikipedia.org/wiki/Rxqual) values
99	not known or not detectable

3.4.5 <mcc>

[Mobile country code](https://en.wikipedia.org/wiki/Mobile_country_code)

3.4.6 <mnc>

[Mobile network code](https://en.wikipedia.org/wiki/Mobile_country_code)

3.4.7 <bsic>

[Base station identity code](https://en.wikipedia.org/wiki/Base_station_identity_code)

3.4.8 <cellid>

[Cell id](https://en.wikipedia.org/wiki/Cell_ID)

NOTE: In a 7-item line, cellid is not provided. We set it to 0 to prevent barfing elsewhere.

3.4.9 <lac>

[Location area code](http://www.telecomabc.com/l/lac.html)

3.4.10 <rla>

Receive level access minimum

GUESS: Minimum receiving level permitted to access the system Per: similar AT engineering mode (AT+QENG) command in [M95 AT commands manual](http://eddywireless.com/yahoo_site_admin/assets/docs/M95_AT_Commands_Manual_V12.196112248.pdf)

3.4.11 <txp>

Transmit power maximum CCCH

3.4.12 <TA>

[Timing Advance](https://en.wikipedia.org/wiki/Timing_advance)

3.5 health_check.log


```
{"cpu_times":
  {"iowait": 4694.23,
   "idle": 3089452.32,
   "user": 1786751.62,
   "system": 125489.34},
 "data_vol": 5.5,
 "root_vol": 5.5,
 "cpu_percent": [42.0, 53.0, 35.9, 38.0],
 "mem":
   {"swap_percent_used": 0.0,
    "free": 464707584},
 "queue_sizes": {
   "arfcn_correlator": 0,
   "geo_correlator": 0,
   "scan_results": 0,
   "cgi_correlator": 0},
 "application_uptime_seconds": 32461,
 "event_timestamp": "2017-05-07T06:32:09.816725",
 "scan_program": "health_check"}
```


The frequency with which these events are generated is determined by the `HEALTH_CHECK_INTERVAL` environment variable.

How is this information useful?

If you notice a trend where a metric under "queue_sizes" is always-increasing, you may have a failed processing thread. Correlate this with the events coming from heartbeat.log. Look for the absence of a heartbeat event for the corresponding thread). If you've confirmed that a thread has failed, the fastest fix is to just restart the sensor. If you can get a traceback for the thread failure, please submit it as an issue at https://github.com/sitch-io/sensor/issues/new.

3.6 heartbeat.log


```
{"heartbeat_service_name": "MainThread", "event_timestamp": "2017-05-07T06:32:09.
↪815061", "scan_program": "heartbeat"}
{"heartbeat_service_name": "kalibrate_consumer", "event_timestamp": "2017-05-
↪07T06:32:09.815243", "scan_program": "heartbeat"}
{"heartbeat_service_name": "arfcn_correlator", "event_timestamp": "2017-05-
↪07T06:32:09.815323", "scan_program": "heartbeat"}
{"heartbeat_service_name": "decomposer", "event_timestamp": "2017-05-07T06:32:09.
↪815391", "scan_program": "heartbeat"}
{"heartbeat_service_name": "gsm_modem_consumer", "event_timestamp": "2017-05-
↪07T06:32:09.815456", "scan_program": "heartbeat"}
{"heartbeat_service_name": "geoip_consumer", "event_timestamp": "2017-05-07T06:32:09.
↪815520", "scan_program": "heartbeat"}
{"heartbeat_service_name": "writer", "event_timestamp": "2017-05-07T06:32:09.815584",
↪"scan_program": "heartbeat"}
{"heartbeat_service_name": "geo_correlator", "event_timestamp": "2017-05-07T06:32:09.
↪815648", "scan_program": "heartbeat"}
{"heartbeat_service_name": "gps_consumer", "event_timestamp": "2017-05-07T06:32:09.
↪815711", "scan_program": "heartbeat"}
{"heartbeat_service_name": "cgi_correlator", "event_timestamp": "2017-05-07T06:32:09.
↪815780", "scan_program": "heartbeat"}
```


These events are most useful when chasing down thread failure. It doesn't happen often, but when it does, you can look at these events as a time-series and see where one ceases to appear. This is most useful when correlated with queue sizes as reflected in health_check.log.

3.7 kal_channel.log


```
{"site_name": "sitch-site-testing",
 "power": 854930.16,
 "final_freq": "874979084",
 "band": "GSM-850",
 "scan_finish": "2017-05-07T06:28:38.545421",
 "event_timestamp": "2017-05-07T06:28:38.545421",
 "sample_rate": "270833.002142",
 "gain": "80.0",
 "scanner_public_ip": "1.1.1.1",
 "scan_start": "2017-05-07T06:23:39.482440",
 "scan_program": "kalibrate",
 "arfcn_int": 157,
 "channel": "157"}
```

3.8 scanner.log


```
{"site_name": "sitch-site-testing",
 "scan_results": [
 {"channel_detect_threshold": "105949.217083",
  "power": "854930.16",
  "final_freq": "874979084",
  "mod_freq": 20916.0,
  "band": "GSM-850",
  "sample_rate": "270833.002142",
  "gain": "80.0",
  "base_freq": 875000000.0,
  "device": "0: Generic RTL2832U OEM",
  "modifier": "-",
  "channel": "157"}
  ],
 "platform": "Unspecified",
 "scan_start": "2017-05-07T06:23:39.482440",
 "scan_location": "sitch-site-testing",
 "scanner_public_ip": "1.1.1.1",
 "scan_finish": "2017-05-07T06:28:38.545421",
 "event_timestamp": "2017-05-07T06:28:38.545421",
 "scan_program": "kalibrate",
 "scanner_name": "sitch-site-testing"}
```


The list of items under scan_results is used by the Decomposer to produce messages that end up in the kal_channel log file.

3.9 sitch_alert.log


```
{"details": "Primary BTS was 310:260:275:20082 now 310:260:275:42302. Site: sitch-
↪site-testing",
 "type": "Primary BTS metadata change.",
 "id": 110,
 "device_id": "sitch-site-testing"
 "event_timestamp": "2017-05-07T06:28:38.545421"}
```


- `details` is a human-readable representation of the event, with details.
- `type` is a human-readable description of the alert type. For a list of supported event types, look in the `__init__` section of http://sensor.readthedocs.io/en/test/_modules/sitchlib/alert_manager.html#AlertManager
- `id` is an ID that maps to a specific event type. This is meant to simplify integration with SIEM and log management systems.
- `device_id` is the device ID (see device configuration environment vars)
- `event_timestamp` is generated when the alert is detected.

3.10 sitch_init.log


```
{"evt_data": "T-Mobile",
 "evt_type": "registration",
```


(continues on next page)

(continued from previous page)


```
 "evt_cls": "gsm_consumer",
 "event_timestamp": "2017-05-06T06:25:49.837957"}

{"evt_data": "\r\n | OK\r\n | ATV1Q0&V \r\r\n | DEFAULT PROFILE\r\n | S0: 0\r\n | S3:␣
↪13\r\n | S4: 10\r\n | S5: 8\r\n | S6: 2\r\n | S7: 60\r\n | S8: 2\r\n | S10: 15\r\n␣
↪| +CRLP: 61,61,48,6\r\n | V: 1\r\n | E: 1\r\n | Q: 0\r\n | X: 4\r\n | &C: 1\r\n | &
↪D: 1\r\n | +CLTS: 0\r\n| +CREG: 0\r\n | +CGREG: 0\r\n | +CMEE: 0\r\n | +CIURC:␣
↪1\r\n | +CFGRI: 2\r\n | +CMTE: 0\r\n | +CANT: 0,0,10\r\n | +STKPCIS: 0\r\n | +CMGF:␣
↪0\r\n | +CNMI: 2,1,0,0,0\r\n | +CSCS: \"IRA\"\r\n | +VTD: 1\r\n | +CALS: 1\r\n |␣
↪+CHF: 0\r\n | +CAAS: 1\r\n | +CBUZZERRING: 0\r\n | +DDET: 0\r\n | +MORING: 0\r\n |␣
↪+SVR: 16\r\n | +CCPD: 1\r\n | +CSNS: 0\r\n | +CSGS: 1\r\n | +CNETLIGHT: 1\r\n |␣
↪+SLEDS: 64,64,64,800,3000,300\r\n | +CSDT: 0\r\n | +CSMINS: 0\r\n | +EXUNSOL: 0\r\n␣
↪| +FSHEX: 0\r\n | +FSEXT: 0\r\n | +IPR: 0\r\n | +IFC: 0,0\r\n | +CSCLK: 0\r\n |␣
↪\r\n | USER PROFILE\r\n | S0: 0\r\n | S3: 13\r\n | S4: 10\r\n | S5: 8\r\n | S6:␣
↪2\r\n | S7: 60\r\n | S8: 2\r\n | S10: 15\r\n | +CRLP: 61,61,48,6\r\n | V: 1\r\n |␣
↪E: 1\r\n | Q: 0\r\n | X: 4\r\n | &C: 1\r\n | &D: 1\r\n | +CLTS: 0\r\n | +CREG:␣
↪0\r\n | +CGREG: 0\r\n | +CMEE: 0\r\n |+CIURC: 1\r\n | +CFGRI: 2\r\n | +CMTE: 0\r\n␣
↪| +CANT: 0,0,10\r\n | +STKPCIS: 0\r\n | +CMGF: 0\r\n | +CNMI: 2,1,0,0,0\r\n |␣
↪+CSCS: \"IRA\"\r\n | +VTD: 1\r\n | +CALS: 1\r\n | +CHF: 0\r\n | +CAAS: 1\r\n |␣
↪+CBUZZERRING: 0\r\n | +DDET: 0\r\n | +MORING: 0\r\n | +SVR: 16\r\n | +CCPD: 1\r\n |␣
↪+CSNS: 0\r\n | +CSGS: 1\r\n | +CNETLIGHT: 1\r\n | +SLEDS: 64,64,64,800,3000,300\r\n␣
↪| +CSDT: 0\r\n | +CSMINS: 0\r\n | +EXUNSOL:0\r\n | +FSHEX: 0\r\n | +FSEXT: 0\r\n |␣
↪+IPR: 0\r\n | +IFC: 0,0\r\n | +CSCLK: 0\r\n | \r\n | ACTIVE PROFILE\r\n | S0: 0\r\n␣
↪| S3: 13\r\n | S4: 10\r\n | S5: 8\r\n | S6: 2\r\n | S7: 60\r\n | S8: 2\r\n | S10:␣
↪15\r\n | +CRLP: 61,61,48,6\r\n | V: 1\r\n | E: 1\r\n | Q: 0\r\n | X: 4\r\n | &C:␣
↪1\r\n | &D: 1\r\n | +CLTS: 0\r\n | +CREG: 0\r\n | +CGREG: 0\r\n | +CMEE: 0\r\n |␣
↪+CIURC: 1\r\n | +CFGRI: 2\r\n | +CMTE: 0\r\n | +CANT: 0,0,10\r\n | +STKPCIS: 0\r\n␣
↪| +CMGF: 0\r\n | +CNMI: 2,1,0,0,0\r\n | +CSCS: \"IRA\"\r\n | +VTD: 1\r\n | +CALS:␣
↪1\r\n | +CHF: 0\r\n | +CAAS: 1\r\n | +CBUZZERRING: 0\r\n | +DDET: 0\r\n | +MORING:␣
↪0\r\n | +SVR: 16\r\n | +CCPD: 1\r\n | +CSNS: 0\r\n | +CSGS: 1\r\n | +CNETLIGHT:␣
↪1\r\n | +SLEDS: 64,64,64,800,3000,300\r\n | +CSDT: 0\r\n | +CSMINS: 0\r\n |␣
↪+EXUNSOL: 0\r\n | +FSHEX: 0\r\n | +FSEXT: 0\r\n | +IPR:0\r\n | +IFC: 0,0\r\n |␣
↪+CSCLK: 0\r\n | \r\n | OK\r\n",
"evt_type": "device_config",
"evt_cls": "gsm_consumer",
"event_timestamp": "2017-05-06T06:25:49.837957"}

{"evt_data": "IMSI_GOES_HERE",
"evt_type": "sim_imsi",
"evt_cls": "gsm_consumer",
"event_timestamp": "2017-05-06T06:25:49.837957"}
```


These messages are only generated when the application starts.

- `registration` records the cell service provider, according to the GSM modem.
- `device_config` dumps the profiles in use from the GSM modem.
- `sim_imsi` records the IMSI from your cell modem's SIM card.

CHAPTER 4

Event Lifecycle

The lifecycle of an event in SITCH begins in the Sensor, and ends with the user's (or alert management system's) consumption. We'll follow the most frequent event, the GSM modem scan event.

4.1 Ingestion

The Sensor runs the gsm_modem_consumer() function as a thread in runner.py. This thread produces events from the output of the GSM modem being in engineering mode. gsm_modem_consumer() wraps the GsmModem class (found in gsm_modem.py), takes the output from the __iter__() in GsmModem, and places it into the `scan_results_queue` FIFO buffer.

4.2 Decomposition

The decomposer() function in runner.py is also run in a thread, and as scan results are placed into the `scan_results_queue` FIFO, it pulls them out and decomposes them into individual events, one for each cell. Copies of these decomposed events (labeled `gsm_modem_channel`) are placed into the `cgi_correlator_queue`, `arfcn_correlator_queue`, and `message_write_queue` FIFO buffers.

4.3 Correlation

The cgi_correlator() and arfcn_correlator() functions are run in threads and consume events from the `cgi_correlator_queue` and `arfcn_correlator_queue` FIFO buffers, respectively. The cgi_correlator() correlates the information contained in the event with the feed information based on the OpenCellID database, taking the geolocation of the sensor into account. If any alarms are produced, they are placed in the `message_write_queue`. The arfcn_correlator() function compares the ARFCN in the event metadata with information contained in the feed based on the FCC license database, taking into account the geolocation of the sensor.

4.4 Transmission

The output() function is run in a thread and listens for events being placed into the `message_write_queue` FIFO. It takes the events it finds there and writes them to disk, appending them to files by event type.

At this point, you have the original scan event, each decomposed channel event, and any alerts produced, logged on disk in specific files, based on event type.

These events are picked up from disk by filebeat, and transmitted to Logstash, which runs in the service side of SITCH.

4.5 Reception

Logstash splits the information between two data stores. The events themselves get sent to Elasticsearch, and you can query them with Kibana. Some of the measurement metadata is sent to influxDB, and can be viewed with Chronograf.

Events with type `sitch_alert` are sent to Slack by Logstash.

CHAPTER 5

Sensor Troubleshooting

5.1 GSM modem device detection

If you're using a GSM modem that's not recognized by the device detector, please add the output from running the `ATI` command against your GSM modem in the variable named `positive_match` in the `is_a_gsm_modem()` ` method, in the `sensor/sitch/sitchlib/device_detector.py` file. Then send a pull request so that everyone can get the benefit of your discovery.

You can do this using the resin.io terminal on the device by doing the following steps.

1. Set the environment variable `GSM_MODEM_BAND` to `nope` to disable the scanner.
2. Identify which TTY port your device is running on. You can find this in the startup logs under the string `DeviceDetector: Detected USB devices`.
3. Run python from the sensors virtual environment

```
/app/sitch/venv/bin/python
```

4. Create a serial connection to the GSM modem.

```
> import serial
> port = '/dev/[THE_MODEMS_TTY_SYS_NAME]'
> serconn = serial.Serial(port, 4800, timeout=1)
```

5. Run the following snippet to get the string you need.

```
> test_command = "ATI \r\n"
> serconn.flush()
> for i in xrange(10):
> line = None
> line = serconn.readline()
> if line is None:
>     time.sleep(1)
>     pass
```

(continues on next page)

(continued from previous page)

```
> else:
>     print("Use this GSM Modem String in your pull request: {0}".format(line))
> serconn.flush()
> serconn.close()
```

5.2 Found but undetected TTY

The DeviceDetector shows it found my GSM Modem or GPS Device by the Configurator does not detect it

5.2.1 How to identify if this is your issue

You will be able to recognize this issue if three conditions are met.

1. You are receiving an error that the device is not configured or cannot bind to its socket.
2. Your Configurator returns an empty array instead of a USB-TTY device name when it attempts to detect a device.
3. Your device detector is detecting these devices

If the device detector cannot find the devices, as the following log message shows, then *this is not your issue.*

5.2.2 How to fix this issue

To fix this issue you can set the hard-coded environment variable for the device that is not detected.

In the following example the GSM modem is not detected.

```
> 22.04.17 08:53:27 (-0400) Configurator: Detected GSM modems:
> 22.04.17 08:53:27 (-0400) []
> 22.04.17 08:53:27 (-0400) Configurator: Detected GPS devices:
> 22.04.17 08:53:27 (-0400) [u'/dev/ttyUSB0']
```

This shows me that the GSM modem was not detected and that my GPS device can be found at '/dev/ttyUSB0'.

By looking at my DeviceDetector I can see that I have two USB devices connected. It also gives me the 'sys_name' of each device.

Since I know that my GPS device has a sys_name of `ttyUSB0` I know that the sys_name GSM device is `ttyUSB1`.

I can now set the `GSM_MODEM_PORT` environment variable to point to /dev/ttyUSB1 in the resin.io `Environment Variables` interface.

(NOTE: for those unfamiliar with python strings it should be noted that the `u` in front of each quoted value in these example logs is specifying that the string is a Unicode string. You do not want to enter the `u` in front of /dev/ttyUSB1 when setting your environment variables.)

If you have successfully set the environment variable you will no longer receive an error message.

In the case of the GSM modem you will also see that the following message has replaced the original error.

5.3 GPS device not detected (U-Blox7)

The U-Blox7 USB GPS device registers as a ttyACM device. If everything else (with respect to the sensor hardware stack) is built to spec, the U-Blox7 GPS will land at `/dev/ttyACM0`. Set the `GSM_MODEM_PORT` Sensor environment variable in resin.io to `/dev/ttyACM0`. The application on the sensor will then restart. Open the terminal in resin.io and `tail -f /data/sitch/log/gps.log` to confirm that the GPS is correctly configured and able to get a location fix. You may have to wait for a few minutes. If this does not work, you can also use the terminal to run `gpsmon` to see if gpsd is able to communicate with the device.

5.4 No events in Kibana

The SITCH sensor relies on Filebeat to read events from log files and transmit them to the Logstash instance running in the SITCH service. There are a few indicators when the transmission process is broken:

1. Confirm that log files are being written:

- Confirm that log files are being written at `/data/sitch/log/`. If your sensor isn't populating log files, the most likely reason is that the sensor is in a reboot loop due to mis-configuration.
- Check the Device Summary page in Resin, for the affected sensor. If the reason that the sensor isn't coming online cleanly isn't celarly explained in the log messages, please reach out in the gitter channel (https://gitter.im/sitch-io/sensor) or open an issue in the sensor project on Github: https://github.com/sitch-io/sensor/issues

2. Make sure that the filebeat process is running on the sensor:

- Check using `ps ef` from the terminal: you should see a line containing: `/usr/local/bin/filebeat-linux-arm -c /etc/filebeat.yml`. If you don't, you can try to start the process manually and look for errors printed to stdout.
- Your crypto certs and keys are retieved in the sensor initialization process and dropped at `/host/run/dbus/crypto/`. You should see three files there: `ca.crt`, `logstash.crt`, and `logstash.key`. If you don't have those files on your system, there's a really good chance that your sensor environment variables aren't set correctly.
- You should confirm that the `VAULT_PATH`, `VAULT_TOKEN`, and `VAULT_URL` environment variables are correct, and that the network path is open between your sensor and Vault.
- You can confirm the network path is open by running this command: `openssl s_client -connect VAULT_HOSTNAME:8200`. Replace `VAULT_HOSTNAME` with the DNS name from the output of `echo $VAULT_URL`, when run in the terminal on the sensor. So if your $VAULT_URL is `https://myvault.mydomain.com:8200`, the command you should run in the terminal on the sensor is: `openssl s_client -connect myvault.mydomain.com:8200`.
- An error message containing `gethostbyname failure` indicates a failure in DNS resolution.
- A message containing `connect: Connection refused` indicates that the OpenSSL client is unable to connect to the port that Vault is running on, and you need to check your iptables and security groups settings, and confirm that Vault is actually listening on that port.
- You should also confirm that Vault is actually running.
- If the Vault container is stopped and restarted, you will need to unseal the Vault again. See the docs for the demo environment (https://github.com/sitch-io/demo) for details on how to unseal the vault.

3. Confirm that Filebeat is processing the log files:

- There's a registry file managed by Filebeat, located at `/data/sitch/log/fb_registry`. This file is what Filebeat uses to maintain its place in your log files, in the event it gets restarted. If this file is empty, confirm that the network path to Logstash is open by running this command: `openssl s_client -connect`

`${LOG_HOST}` If the connection times out, you should take a hard look at your network ACLs and iptables rules.

4. If Filebeat appears to be transmitting events to Logstash and you still don't see events in Kibana, you can run the logstash container in debug mode by changing the docker-compose.yml file's setting for `services.logstash.image` from `docker.io/sitch/logstash` to `docker.io/sitch/logstash:debug`. Then, use docker-compose to take your stack down and back up again. This will be very verbose, and can cause a substantial amount of disk space consumption. Don't leave it like that forever.
5. If there is no indication that Logstash is having trouble getting events into Elasticsearch, check that you have an index for logstash built in Kibana.

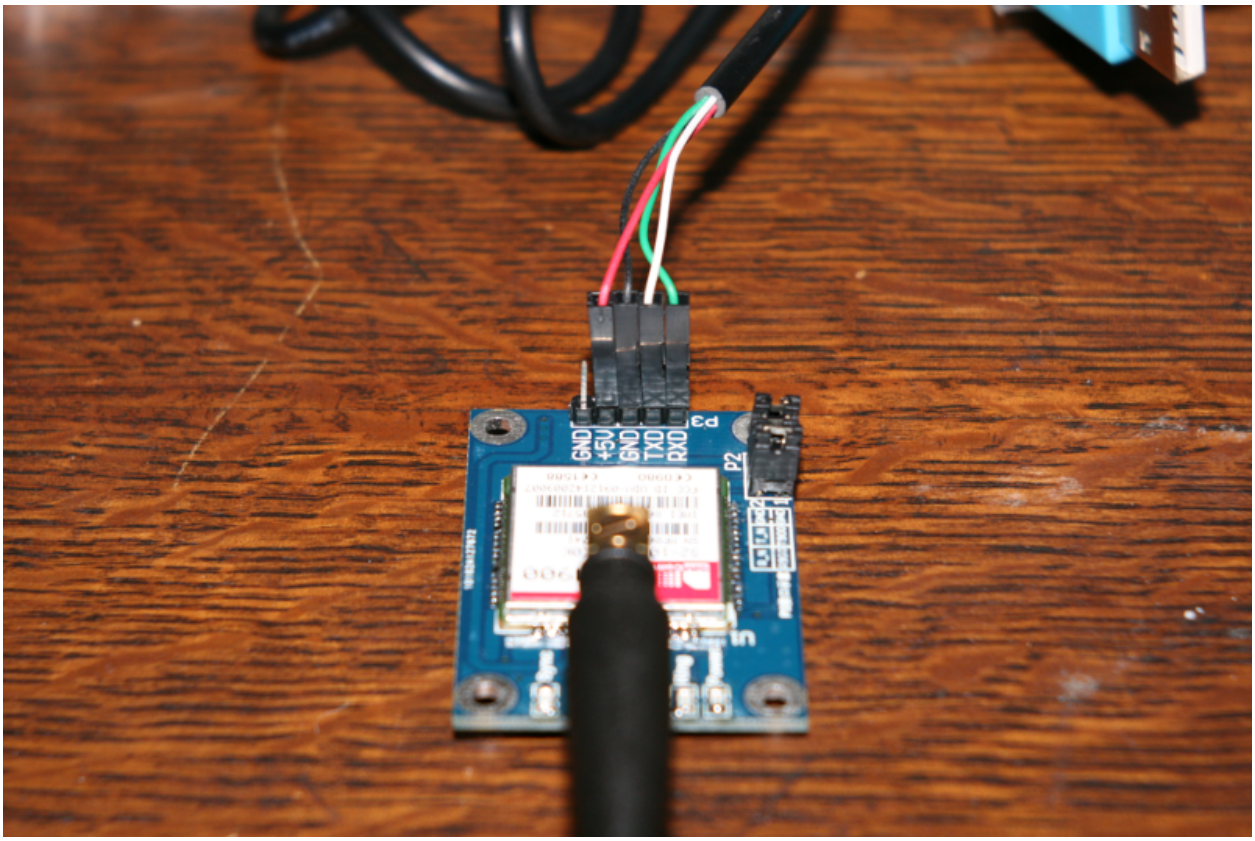
- Navigate to this URL: https://MY_SITCH_SERVICE_HOSTNAME:8443/app/kibana#/management/kibana/indices , replacing MY_SITCH_SERVICE_HOSTNAME with the hostname of your SITCH service-side environment. If you have an index, you should have events.
- Try adjusting your time window, and confirm that the system clocks in your SITCH service side components are correct.
- Time drift can not only cause the query in Kibana to look weird, but it can cause an SSL connection negotiation failure between the sensor and service.

If none of the above lead you to success, please don't hesitate to file an issue in the sensor's Github repository: https://github.com/sitch-io/sensor/issues and/or reach out in the Gitter channel: https://gitter.im/sitch-io/sensor.

CHAPTER 6

Reference Images

6.1 Connecting the USB TTY cable to the SIM 900 GSM modem



CHAPTER 7

SITCH Sensor Internal API

7.1 AlertManager

class sitchlib.**AlertManager**(*device_id*)

AlertManager is used to ensure alerts are consistently formatted.

build_alert(*alert_id*, *alert_message*, *location=None*)

Build the actual alert and returns it, formatted.

DEPRECATION NOTICE: The 'alert_id' field has been introduced for better readability. It's currently set to be the same as 'id'. At some point in the future, the 'id' field will be removed.

Parameters

- **alert_id** (*int*) – The ID of the alert you want to build
- **alert_message** (*str*) – The message to be embedded in the alert.

Returns

Position 0 contains the string 'sitch_alert'. Position 1 contains the alert and metadata.

Return type tuple

get_alert_type(*alert_id*)

Return the alert description for alert_id.

7.2 ArfcnCorrelator

class sitchlib.**ArfcnCorrelator**(*feed_dir*, *whitelist*, *power_threshold*, *device_id*)

The ArfcnCorrelator compares ARFCN metadata against feeds and threshold.

The feed data is put in place by the FeedManager class, prior to instantiating the ArfcnCorrelator.

classmethod arfcn_from_scan(*scan_type*, *scan_doc*)

Pull the ARFCN from different scan types.

Parameters

- **scan_type** (*str*) – "kal_channel", "gsm_modem_channel", or "gps".
- **scan_doc** (*dict*) – Scan document

Returns

ARFCN from scan, or None if scan is unrecognized or unsupported.

Return type str

arfcn_over_threshold(*arfcn_power*)
Compare the ARFCN power against the thresholdset on instantiation.

Parameters **arfcn_power** (*float*) – If this isn't a float already, it will be coerced to float.

Returns True if arfcn_power is over threshold, False if not.

Return type bool

compare_arfcn_to_feed(*arfcn*, *site_name*, *sensor_name*)
Wrap other functions that dig into the FCC license DB.

This relies on the observed_arfcn instance variable for caching, to skip DB comparison, that way we (probably) won't end up with a forever-increasing queue size.

Parameters **arfcn** (*str*) – This is the text representation of the ARFCN we want to compare against the FCC license database.

Returns

You get back a list of alerts as tuples, where position 0 is 'sitch_alert' and position 1 is the actual alert.

Return type list

correlate(*scan_bolus*)
Entrypoint for correlation, wraps individual checks.

Parameters **scan_bolus** (*tuple*) – Position 0 contains a string defining scan type. If it's type 'gps', the geo_state instance variable will be updated with Position 1's contents. If the scan type is 'kal_channel', we perform feed and threshold comparison. any other scan type will be compared against the feed only.

Returns

Returns a list of alerts. If no alerts are generated, an empty list is returned.

Return type list

feed_alert_generator(*arfcn*, *site_name*, *sensor_name*)
Wrap the yield_arfcn_from_feed function, and generates alerts.

Parameters **arfcn** (*str*) – This is the string representation of the ARFCN to be correlated.

Returns This returns a list of alert tuples.

Return type list

classmethod is_in_range(*item_gps*, *state_gps*)
Return True if items are within 40km.

manage_arfcn_lists(*direction*, *arfcn*, *aspect*)
Manage the instance variable lists of ARFCNs.

This is necessary to maintain an accurate state over time, and reduce unnecessary noise.

Parameters

- **direction** (*str*) – Only will take action if this is "in" or "out"
- **arfcn** (*str*) – This is the ARFCN that will be moved in or our of the list
- **aspect** (*str*) – This is used to match the ARFCN with the list it should be moved in or out of. This should be either "threshold" or "not_in_range".

match_arfcn_against_feed(*arfcn*, *state_gps*)
Get a match for the ARFCN within range of the sensor.

Parameters **arfcn** (*str*) – Absolute Radio Frequency Channel Number

Returns True if there is an ARFCN in range, False if not.

Return type bool

7.3 CgiCorrelator

class sitchlib.**CgiCorrelator**(*feed_dir*, *cgi_whitelist*, *mcc_list*, *device_id*)
The CgiCorrelator compares CGI addressing against the OpenCellID DB.

The feed data is put in place by the FeedManager class, prior to instantiating the CgiCorrelator.

classmethod bts_from_channel(*channel*)
Create a simplified representation of BTS metadata.

Parameters **channel** (*dict*) –

Returns Contains MCC, MNC, LAC, and cellid

Return type dict

classmethod build_chan_here(*channel*, *state*)
Build geo information for channel, to aid in geo correlation.

Parameters

- **channel** (*dict*) – Channel metadata
- **state** (*dict*) – Geo-json representing the current location of the sensor

Returns

Original channel structure, with the current sensor location embedded.

Return type dict

classmethod cell_matches(*cell*, *mcc*, *mnc*, *lac*, *cellid*)
Compare cell metadata against mcc, mnc, lac, cellid.

classmethod cgi_whitelist_message(*cgi_wl*)
Format and return the CGI whitelist initialization message.

Parameters **cgi_wl** (*list*) – CGI whitelist

Returns Formatted message

Return type str

classmethod channel_in_feed_db(*channel*)
Return True if channel geo metadata is complete.

classmethod channel_out_of_range(*channel*)
Check to see if sensor is out of range for CGI.

Parameters **channel** (*dict*) – Channel metadata

Returns True if the sensor is in range of the detected CGI

Return type bool

check_channel_against_feed(*channel*)
Determine whether or not to fire an alert for CGI presence in feed.

Parameters **channel** (*dict*) – Channel metadata

Returns

Empty if there is no alert, a two-item tuple if an alert is generated.

Return type tuple

check_channel_range(*channel*)
Check to see if the detected CGI is in range.

Parameters **channel** (*dict*) – Channel metadata, enriched with feed info.

Returns

Empry if no alert is generated. A two-item tuple if an alert condition is detected.

Return type tuple

check_scan_document(*scan_document*)
Check to see if there are no in-LAI neighbors for channel 0

classmethod convert_float_targets(*channel*)
Convert string values for rxq and rxl to floating point.

classmethod convert_hex_targets(*channel*)
Convert lac and cellid from hex to decimal.

correlate(*scan_bolus*)
Entrypoint for the CGI correlation component.

Parameters **scan_bolus** (*tuple*) – scan_bolus[0] contains the scan type. If the type is 'gps', it will set the correlator's geo location. For other scan types, we expect them to look like gsm_modem_channel events, and they are compared against the feed database as well as state history, tracking things like the current active cell's CGI.

Returns

Returns a list of tuples, representing alerts. If no alerts fire, the list will be empty.

Return type list

feed_comparison(*channel*)
Compare channel metadata against the feed DB.

This function wraps a few checks against the feed DB. It first checks if the bts is in the feed DB. Next, it checks that the sensor is within range of the BTS in the feed DB. Finally, if it's the primary channel, it checks to see if the primary BTS has changed.

Parameters **channel** (*dict*) – Channel, enriched with geo information

Returns

If alarms are generated, they'll be returned in a list of tuples. Otherwise, an empty list comes back.

Return type list

classmethod get_cell_by_id(*scan_document*, *cell_no*)
Get cell from doc by ID

classmethod get_cgi_int(*channel*)
Attempt to create an integer representation of CGI.

get_feed_info(*mcc*, *mnc*, *lac*, *cellid*)
Check CGI against cache, then against the feed DB.

Parameters

- **mcc** (*str*) – Mobile Country Code
- **mnc** (*str*) – Mobile Network Code
- **lac** (*str*) – Location Area Code
- **cellid** (*str*) – Cell ID

Returns Dictionary containing feed information for CGI

Return type dict

get_feed_info_from_db(*mcc*, *mnc*, *lac*, *cellid*)
Interrogate DB for CGI information.

Parameters

- **(str)** – Mobile Country Code
- **mnc** (*str*) – Mobile Network Code
- **lac** (*str*) – Location Area Code
- **cellid** (*str*) – Cell ID

Returns

Dictionary containing feed information for CGI. If no information exists, the feed geo information will be zeroed out...

Return type dict

classmethod make_bts_friendly(*bts_struct*)
Create a human-friendly representation of CGI.

Parameters **bts_struct** (*dict*) – Simple structure containing CGI components.

Returns

String reperesentation of CGI, with items being colon-separated.

Return type str

classmethod normalize_feed_info_for_cache(*feed_item*)
Normalize field keys for the feed cache.

classmethod primary_bts_changed(*prior_bts*, *channel*, *cgi_whitelist*)
Create alarms if primary BTS metadats changed.

Parameters

- **prior_bts** (*str*) – Current primary BTS

- **channel** (*dict*) – Channel metadata
- **cgi_whitelist** – Whitelist of CGIs to NOT alert on

Returns

True if the primary BTS has changed and the new BTS in not on the whitelist. False otherwise.

Return type bool

process_cell_zero(*channel*)
Process channel zero.

Parameters **channel** (*dict*) – Channel metadata.

Returns

Empry if there is no alert, a two-item tuple if an alert condition is detected.

Return type tuple

classmethod should_skip_feed(*channel*)
Examine channel info to determine if feed comparison should happen.

Parameters **channel** (*dict*) – Channel information.

Returns True if channel information is complete, False if not.

Return type bool

7.4 ConfigHelper

class sitchlib.**ConfigHelper**(*sitch_var_base_dir='/data/sitch/'*)
Manage configuration information for entire SITCH Sensor.

build_logrotate_config()
Generate logrotate config file contents.

classmethod get_db_schema_translations(*filename='/etc/schemas/feed_db_translation.yaml'*)
Get the feed DB schema translations from file.

classmethod get_db_schemas(*filename='/etc/schemas/feed_db_schema.yaml'*)
Get the feed DB schemas from file.

classmethod get_filebeat_template(*filename='/etc/templates/filebeat.json'*)
Get the filebeat config from template file.

classmethod get_from_env(*k*)
Get configuration items from env vars. Hard exit if not set.

get_gps_device_port()
Get GPS device from detector, override with env var.

get_gsm_modem_port()
Get GSM modem port from detector, override with env var.

classmethod get_list_from_env(*k*, *optional=False*)
Get a list from environment variables.

If optional=True, the absence of this var will cause a hard exit.

get_secret_from_vault()
Retrieve secrets from Vault.

print_devices_as_detected()
Print detected GPS and GSM devices.

classmethod set_filebeat_logfile_paths(*log_prefix*, *filebeat_config*)
Sets all log file paths to align with configured log prefix.

write_filebeat_config()
Write out filebeat config to file.

7.5 Decomposer

class sitchlib.**Decomposer**
Decompose device messages into normalized log messages.

classmethod decompose(*scan*)
Direct messages to the correct decomposer.

7.6 DeviceDetector

class sitchlib.**DeviceDetector**
Interrogate all USB TTY ports.

gsm_radios
list – This is a list of GSM radios, represented in dict type objects.

gps_devices
list – This is a list of GPS devices. Just strings like '/dev/ttyUSB0'.

classmethod find_gps_radios(*usbtty_ports*)
Interrogate USB TTY ports, return a list of GPS devices.

classmethod find_gsm_radios(*usbtty_ports*)
Interrogate USB TTY ports, return GSM radios.

classmethod get_devices_by_subsys(*subsys_type*)
Get devices from udev, by type.

classmethod get_gsm_modem_info(*port*)
Get modem information.

Parameters **port** (*str*) – Device/port to interrogate.

Returns

metadata describing modem manufacturer, model, revision, and serial.

Return type dict

classmethod interrogate_gsm_modem(*port*, *command*)
Issue command on port, return output.

Parameters

- **port** (*str*) – Port/device to interrogate.
- **commmand** (*str*) – Command to be issued.

Returns

Response from device, if any. If none, returns an empty string.

Return type str

classmethod interrogator(*match_list*, *port*, *test_command=None*)
Interrogate serial port, and attempt to match output.

Parameters

- **match_list** (*list*) – List of strings that positively identify a device of a specific type.
- **port** (*str*) – Port to be interrogated.
- **test_command** (*str*) – Command to trigger output to match against match_list.

Returns True if the device is a positive match, False if not.

Return type bool

classmethod interrogator_matcher(*matchers*, *line*)
Attempt to match output against known identifing strings.

Parameters

- **matchers** (*list*) – List of strings which represent positive matches.
- **line** (*str*) – Output from USB TTY device.

Returns True if it's a match, False if not.

Return type bool

classmethod is_a_gps(*port*)
Wrap interrogator for determining when a GPS is discovered.

classmethod is_a_gsm_modem(*port*)
Wrap interrogator for determining when a GSM modem is discovered.

7.7 FeedManager

class sitchlib.**FeedManager**(*config*)
Manage downloading the feed DB, and merging it into the sqlite DB.

apply_mode(*mode*)
Adjust FeedManager behavior according to operating mode.

classmethod create_and_populate_db(*db_schema*, *db_translate_schema*, *feed_files*, *db_file*, *target_radios*)
Create DB, then merge all records from file.

Parameters

- **db_schema** (*list*) – One K:V set from the top level of feed_db_schema.yaml
- **feed_files** (*list*) – List of feed files to be merged.
- **db_file** (*str*) – Full path of CGI DB file.

Returns Most recent timestamp from merge.

Return type str

classmethod create_db(*db_file*, *db_schema*)
Create a DB.

This creates either the CGI or ARFCN database.

Parameters

- **db_file** (*str*) – Path to DB file.
- **db_schema** (*dict*) – One top-level k:v from feed_db_schema.yaml

classmethod create_db_init_string(*db_schema*)
Create DB initialization string based on db_schema input.

Expects a dictionary like this:

{"table_name":

{"fields": ["field_1",

"field_2" "field_3"],

"unique": ["field_1", "field_2"]}}

Parameters **db_schema** (*dict*) – Dictionary describing the DB schema

classmethod dump_csv_to_db(*db_schema*, *db_translate_schema*, *feed_file*, *db_file*, *target_radios*, *last_upd=0*)
Merge CSV into DB, taking into account the record update time.

Parameters

- **db_schema** (*dict*) – Dictionary produced from feed_db_schema.yaml. Only one key, `cgi` or `arfcn`.
- **feed_file** (*str*) – Path to feed CSV file.
- **db_file** (*str*) – Path to sqlite DB file.
- **last_upd** (`int`, optional) – Epoch time. Records updated before this date will not be inserted into the DB.

get_newest_record_time(*db_type*)
Get the newest record time from file in feed dir.

classmethod get_source_url(*url_base*, *mcc*)
Create source URL for MCC file.

Parameters

- **url_base** (*str*) – Base URL for MCC file.
- **mcc** (*str*) – MCC for feed file.

classmethod mass_insert(*table*, *fields*, *rows*, *db_file*)
Mass-insert records into the DB.

Parameters

- **schema** (*list*) – List of DB fields.
- **rows** (*list*) – List of tuples, each tuple contains values corresponding to the keys in *schema*.
- **db_file** (*str*) – Path to sqlite file.

classmethod merge_feed_files_into_db(*db_schema*, *db_translate_schema*, *feed_files*, *db_file*, *target_radios*, *last_upd*)
Wrapper for merging feed file data into CGI DB.

Parameters

- **schema** (*list*) – List of fields in DB
- **feed_file** (*str*) – Path to feed file to be merged into CGI DB.
- **db_file** (*str*) – Path to CGI DB file.
- **last_upd** (*str*) – Epoch time stamp, will not attempt to merge any records with timestamps before this time.

Returns Most recent timestamp from merged feed file.

Return type str

classmethod place_feed_file(*feed_dir*, *url_base*, *item_id*)
Retrieve and places feed files for use by the Enricher modules.

Parameters

- **feed_dir** (*str*) – Destination directory for feed files
- **url_base** (*str*) – Base URL for hosted feed files
- **item_id** (*str*) – For FCC, this is the two-letter ("CA" or "TN", for example), which is used in the retrieval of the feed file

  as well as the construction of the local feed file name. For MCC this is the MCC, but in string form. Not integer.

classmethod reconcile_db(*db_schema*, *db_translate_schema*, *feed_files*, *db_file*, *target_radios*, *last_update*)
Reconcile feed files against the target DB.

Parameters

- **feed_files** (*list*) – List of paths to feed files.
- **db_file** (*str*) – Full path to CGI DB file.
- **last_update** (*str*) – Epoch time of most recent record in DB

Returns Epoch timestamp of most recently updated DB record.

Return type str

set_newest_record_time(*db_type*, *timestamp*)
Set the newest record time.

Parameters **timestamp** (*str*) – Epoch time to be written to file If not string, will be coerced to string.

classmethod should_update_record_epoch(*anchor_time*, *update_time*)
Compare timestamps to determine if a record should be updated.

classmethod should_update_record_iso(*anchor_time*, *update_time*)
Compare timestamps to determine if a record should be updated.

classmethod tup_from_row(*schema*, *row*)
Convert a row into a tuple, for insertion into DB.

Parameters

- **schema** (*list*) – Field list for DB.
- **row** (*dict*) – Row of data. Keys align with items in *schema*.

Returns

Tuple representing values to be inserted into DB, ordered by fields in *schema*.

Return type tuple

update_feed_db()
Wrapper for feed file reconciliation against DBs.

update_feed_files()
Wrapper for feed file retrieval routines.

7.8 GeoCorrelator

class sitchlib.**GeoCorrelator**(*device_id*)
Geographic correlator.

correlate(*scan_bolus*)
Correlate one geo event.

The first time we get a geo event, we set the state and print a message to stdout to that effect. Every subsequent message is compared against the geo_anchor. Once the anchor is set, it does not change for the life of the instance. Correlation of subsequent events causes the distance beween the anchor and current event to be determined and if the threshold of 10km is exceeded, an alert is returned.

Parameters **scan_bolus** (*tuple*) – Two-item tuple. Position 0 contains the scan type, which is not checked. We should only ever have geo events coming through this method. Position 1 is expected to contain geo json.

Returns

List of alerts. If no alerts are fired, the list returned is zero-length.

Return type list

classmethod geo_drift_check(*geo_anchor*, *gps_scan*, *threshold*, *device_id*)
Fire alarm if distance between points exceeds threshold.

Parameters

- **geo_anchor** (*dict*) – Geographic anchor point, usually stored in an instance variable and passed in via the *correlate()* method.
- **gps_scan** (*dict*) – Same format as geo_anchor, expects the same format as *geo_anchor*.
- **threshold** (*int*) – Alerting threshold in km.

Returns

list of alerts (usually just one) or an empty list of there are no alerts.

Return type list

classmethod time_drift_check(*gps_scan*, *threshold_mins*, *device_id*)
Checks drift value, alarms if beyond threshold.

7.9 GeoIp

class sitchlib.**GeoIp**(*delay=60*)
Generate GeoIP events.

set_geo()
Use public IP to determine GeoIP.

set_ip()
Set public IP address.

7.10 GeoipDecomposer

class sitchlib.geoip_decomposer.**GeoipDecomposer**
GeoIP Decomposer.

classmethod decompose(*scan_document*)
Validate and decompose GeoIP Events.

Parameters **scan_document** (*dict*) – GeoIP scan document.

Returns

one item in list: a two-item tuple. Position 0 is *geo_ip*. Position 1 is the actual scan document. If the scan fails validation, you'll only get an empty list back

Return type list

classmethod scan_document_is_valid(*scan_document*)
Validate the scan document.

7.11 GpsDecomposer

class sitchlib.gps_decomposer.**GpsDecomposer**
GPS Decomposer.

classmethod decompose(*scan_document*)
Decompose a GPS event.

Parameters **scan_document** (*dict*) – Geo json from GPS device.

Returns

One two-item tuple in list. Position 0 is *gps*, position 1 is the validated geo scan. If the scan doesn't validate, an empty list is returned.

Return type list

classmethod scan_document_is_valid(*scan_document*)
Validate the scan document.

7.12 GpsListener

class sitchlib.**GpsListener**(*delay=60*)
Wrap the GPS device with an iterator.

classmethod get_time_delta(*iso_1*, *iso_2*)
Get the drift, in minutes, between two ISO times.

7.13 GsmDecomposer

class sitchlib.gsm_decomposer.**GsmDecomposer**
Decomposes GSM scans.

classmethod bts_from_channel(*channel*)
Return clean BTS from channel.

classmethod convert_float_targets(*channel*)
Convert rxq and rxl to float.

classmethod convert_hex_targets(*channel*)
Convert LAC anc CellID from hex to decimal.

classmethod decompose(*scan_document*)
Turn one scan document into a list of channel scan documents.

Parameters **scan_document** (*dict*) – GSM modem scan.

Returns

List of tuples. First position in tuple identifies scan type. Second position is the actual scan data.

Return type list

classmethod enrich_channel_with_scan(*channel*, *scan_document*)
Enrich channel with scan document metadata.

classmethod get_cgi_int(*channel*)
Attempt to create an integer representation of CGI.

classmethod make_bts_friendly(*bts_struct*)
Expect a dict with keys for mcc, mnc, lac, cellid.

7.14 GsmModem

class sitchlib.**GsmModem**(*ser_port*)
GSM Modem handler class. Interfaces with device over serial.

Calling GsmModem.set_eng_mode() causes the module to go into engineering mode, which will cause it to return cell network information. It has an iterator (generator) built in that cranks out dicts.

classmethod clean_operator_string(*operator_string*)
Clean up the operator string.

dump_config()
Dump modem's configuration.

eng_mode(*status*)
Set or unset engineering mode on the modem.

Parameters **status** (*bool*) – True to enable engineering mode, False to disable.

get_imsi()
Get the IMSI of the SIM installed in the modem.

get_reg_info()
Get registration information from the modem.

classmethod process_12(*parts*)
Process a 12-part CENG message.

Parameters **parts** (*list*) – Parts parsed from original CENG message.

Returns Structured cell channel metadata.

Return type dict

classmethod process_7(*parts*)
Process a 7-part CENG message.

In a 7-item line, cellid is not provided. We set it to 0 to prevent barfing elsewhere.

Parameters **parts** (*list*) – Parts parsed from original CENG message.

Returns Structured cell channel metadata.

Return type dict

classmethod process_8(*parts*)
Process an 8-part CENG message.

Parameters **parts** (*list*) – Parts parsed from original CENG message.

Returns Structured cell channel metadata.

Return type dict

classmethod process_line(*line*)
Process line output from GSM modem.

We expect to see only lines starting with *+CENG:*. Otherwise, it's an empty dictionary getting returned.

Parameters **line** (*str*) – Raw line output from GSM modem.

Returns Structured data parsed from *line*.

Return type dict

set_band(*band*)
Set the band the GSM modem should communicate on.

If the band does not set correctly, an error will print to stdout and the original setting will persist.

Parameters **band** (*str*) – Pick one: *EGSM_MODE*, *PGSM_MODE*, *DCS_MODE*, *GSM850_MODE*, *PCS_MODE*, *EGSM_DCS_MODE*, *GSM850_PCS_MODE*, *EGSM_PCS_MODE*, or *ALL_BAND*.

7.15 KalDecomposer

class sitchlib.kal_decomposer.**KalDecomposer**
Decompose Kalibrate scans.

classmethod decompose(*scan_document*)
Decompose Kalibrate scans into channels.

The first item in each returned tuple indicates the scan doc type. This module produces: "scan" (Kalibrate scan doc) and "kal_channel" (Individual channel from Kalibrate scan)

Parameters **scan_document** (*dict*) – Output from Kalibrate as interpreted by *kalibrate* Python module.

7.16 LocationTool

class sitchlib.**LocationTool**

Class with location-oriented functions.

classmethod get_distance_between_points(*point_1*, *point_2*)

Calculate distance between points.

Parameters

- **point_1** (*tuple*) – (lon, lat) for first point.
- **point_2** (*tuple*) – (lon, lat) for second point.

Returns Kilometers between *point_1* and *point_2*.

Return type int

classmethod get_geo_for_ip(*ip_address*)

Get geo coordinates for IP address.

Parameters **ip_address** (*str*) – IP address.

classmethod validate_geo(*latlon*)

Validate that lon/lat are valid numbers for Planet Earth

7.17 Logger

class sitchlib.**LogHandler**(*config*)

Instantiate this class with the log file prefix.

classmethod get_log_file_name(*ltype*)

Get the name of the appropriate log file for the message type.

Parameters **ltype** (*str*) – Log type

Returns Log file name

Return type str

record_log_message(*bolus*)

Determine log file for message and send to the writer.

write_log_message(*log_file_type*, *message*)

Write message to disk.

Parameters

- **log_file_type** (*str*) – Type of log message
- **message** (*str*) – Message to be logged to disk

7.18 Utility

class sitchlib.**Utility**

General utility class.

classmethod calculate_distance(*lon_1*, *lat_1*, *lon_2*, *lat_2*)
Wrap the LocationTool.get_distance_between_points() fn.

classmethod construct_feed_file_name(*feed_dir*, *prefix*)
Construct full path for feed file.

classmethod create_file_if_nonexistent(*path*, *lfile*)
Create file and path, if it doesn't already exist.

classmethod create_path_if_nonexistent(*path*)
Create filesystem directory path.

classmethod dt_delta_in_minutes(*dt_1*, *dt_2*)
Calculate the delta between two datetime objects, in minutes.

classmethod dt_from_iso(*iso_time*)
Exchange an ISO8601-formatted string for a datetime object.

classmethod epoch_to_iso8601(*unix_time*)
Transform epoch time to ISO8601 format.

classmethod get_now_string()
Get ISO8601 timestamp for now.

classmethod get_performance_metrics(*application_uptime_s*, *queue_sizes={}*)
Get sensor hardware and os performance statistics.

classmethod get_platform_info()
Get information on platform and hardware.

classmethod get_platform_name()
Get platform name from lshw output.

classmethod get_public_ip()
Get public IP.

classmethod heartbeat(*service_name*)
Generate heartbeat message.

classmethod hex_to_dec(*hx*)
Change hex to decimal.

classmethod is_valid_json(*in_str*)
Test string for json validity.

classmethod pretty_string(*structure*)
Pretty-print lines.

classmethod start_component(*runcmd*)
Start a thing.

classmethod str_to_float(*s*)
Change string to float.

classmethod strip_list(*raw_struct*)
Strip contents from single-item list.

classmethod validate_geojson(*geojson*)
Ensure that geojson contains the right fields

classmethod write_file(*location*, *contents*)
Write string to file.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

N

P

R

S

T

U

V

W