# GeoHealthCheck Documentation

*Release 0.9.0*

**Tom Kralidis**

**Aug 11, 2022**

# CONTENTS


**1 Overview** **3**

**2 Features** **5**

**3 Links** **7**

3.1 Installation . . . . . 7
3.2 Configuration . . . . . 12
3.3 Administration . . . . . 16
3.4 User Guide . . . . . 18
3.5 Architecture . . . . . 34
3.6 Plugins . . . . . 38
3.7 License . . . . . 66
3.8 Contact . . . . . 67

**4 Indices and tables** **69**

**Python Module Index** **71**

**Index** **73**


GeoHealthCheck

CHAPTER

ONE

# OVERVIEW

GeoHealthCheck (GHC) is a Python application to support monitoring OGC services uptime, availability and Quality of Service (QoS).

GHC can be used to monitor overall health of OGC services (OWS) like WMS, WFS, WCS, WMTS, SOS, CSW and more, plus some recent OGC APIs like SensorThings API and WFS v3 (OGC Features API). But also standard web REST APIs and ordinary URLs can be monitored.

CHAPTER

TWO

# FEATURES

- lightweight (Python with Flask)
- easy setup
- support for numerous OGC resources
- flexible and customizable: look and feel, scoring matrix
- user management
- database agnostic: any SQLAlchemy supported backend
- database upgrades: using Alembic with Flask-Migrate
- extensible healthchecks via Plugins
- per-resource scheduling and notifications
- per-resource HTTP-authentication like Basic, Token (optional)
- regular status summary report via email (optional)

CHAPTER

# THREE

# LINKS

- website: http://geohealthcheck.org
- GitHub: https://github.com/geopython/geohealthcheck
- Demo: https://demo.geohealthcheck.org (official demo, master branch)
- Presentation: http://geohealthcheck.org/presentation
- Gitter Chat: https://gitter.im/geopython/GeoHealthCheck

This document applies to GHC version 0.9.0 and was generated on Aug 11, 2022. The latest version is always available at http://docs.geohealthcheck.org.

Contents:

## 3.1 Installation

Below are installation notes for GeoHealthCheck (GHC).

### 3.1.1 Docker

The easiest and quickest install for GHC is with Docker/Docker Compose using the GHC images hosted on Docker Hub.

See the GHC Docker Readme for a full guide.

### 3.1.2 Requirements

GeoHealthCheck is built on the awesome Flask micro-framework and uses *Flask-SQLAlchemy* for database interaction and Flask-Login for authorization. *Flask-Migrate* with *Alembic* and *Flask-Script* is used for database upgrades.

*OWSLib* is used to interact with OGC Web Services.

*APScheduler* is used to run scheduled healthchecks.

These dependencies are automatically installed (see below). `Paver` is used for installation and management. `Cron` was used for scheduling the actual healthchecks before v0.5.0.

Starting from version v0.8.0.0 GeoHealthCheck requires **python 3**. Previous versions require **python 2**.

### 3.1.3 Install

**Note:** It is strongly recommended to install GeoHealthCheck in a Python `virtualenv`. a `virtualenv` is self-contained and provides the flexibility to install / tear down / whatever packages without affecting system wide packages or settings. If installing on Ubuntu, you may need to install the python-dev package for installation to complete successfully.

- Download a GeoHealthCheck release from https://github.com/geopython/GeoHealthCheck/releases, or clone manually from GitHub.

```
python3 -m venv ghc && cd ghc
source ghc/bin/activate
git clone https://github.com/geopython/GeoHealthCheck.git
cd GeoHealthCheck

# install paver dependency for admin tool
pip install Paver

# setup app
paver setup

# create secret key to use for auth
paver create_secret_key

# almost there!  Customize config
vi instance/config_site.py
# edit:
# - SQLALCHEMY_DATABASE_URI
# - SECRET_KEY  # from paver create_secret_key
# - GHC_RETENTION_DAYS
# - GHC_SELF_REGISTER
# - GHC_NOTIFICATIONS
# - GHC_NOTIFICATIONS_VERBOSITY
# - GHC_ADMIN_EMAIL
# - GHC_NOTIFICATIONS_EMAIL
# - GHC_SITE_TITLE
# - GHC_SITE_URL
# - GHC_RUNNER_IN_WEBAPP # see 'running' section below
# - GHC_REQUIRE_WEBAPP_AUTH  # optional: to require authentication to access webapp
# - GHC_SMTP  # if GHC_NOTIFICATIONS is enabled
# - GHC_MAP  # or use default settings
# - GEOIP  # or use the default settings

# init database
python GeoHealthCheck/models.py create

# start web-app
python GeoHealthCheck/app.py  # http://localhost:8000/

# when you are done, you can exit the virtualenv
deactivate
```

NB GHC supports internal scheduling, no cronjobs required.

### 3.1.4 Upgrade

An existing GHC database installation can be upgraded with:

```
# In the top directory (e.g. the topdir cloned from github)
paver upgrade

# Notice any output, in particular errors
```

Notes:

- **Always backup your database first!!**
- make sure Flask-Migrate is installed (see requirements.txt), else: *pip install Flask-Migrate==2.5.2*, but best is to run *paver setup* also for other dependencies
- upgrading is "smart": you can always run *paver upgrade*, it has no effect when DB is already up to date
- when upgrading from earlier versions without Plugin-support:
  - adapt your *config_site.py* to Plugin settings from *config_main.py*
  - assign *Probes* and *Checks* to each *Resource* via the UI

When running with Docker see the GHC Docker Readme how to run *paver upgrade* within your Docker Container.

#### Upgrade notes v0.5.0

In GHC v0.5.0 a new run-architecture was introduced. By default, healthchecks run under the control of an internal scheduler, i.s.o. of external cron-jobs. See also the *Architecture* chapter and *Healthcheck Scheduling* and below.

#### Upgrade notes v0.6.0

In GHC v0.6.0 encryption was added for password storage. Existing passwords should be migrated via the *paver upgrade* command. Also password recovery was changed: a user can create a new password via a unique, personal URL that GHC sends by email. This requires a working email configuration and a reachable *SITE_URL* config value. See *User Management* for solving password problems.

See closed issues for related Milestone 0.6.0

#### Upgrade notes v0.7.0

No database changes. Many fixes and enhancements, see closed issues for related Milestone 0.7.0.

#### Upgrade notes v0.8.0

Main change: migrated from Python 2 to Python 3. No DB upgrades required. One major improvement was more robust (HTTP) retries using the *requests Session* object.

See closed issues for related Milestone 0.8.0.

#### Upgrade notes v0.8.3

Main change: Bugfixes and small new features on 0.8.0 (0.8.1 and 0.8.2 were skipped). No DB upgrades required.

OWSLib was upgraded to 0.20.0. Some Py2 to Py3 string encoding issues.

One major improvement was adding *User-Agent* HTTP header for Probe requests.

See closed issues for related Milestone 0.8.3.

#### Upgrade notes v0.9.0

Many (58!) issues and PRs went in. Also thanks to contributors some major features were added:

- configurable Geocoder (by @borrob)
- CI (from Travis CI) and Docker build/push migrated to Github Workflows
- OGC 3DTiles Probe (by SpotInfo)
- MapBox TileJSON Probe (by SpotInfo)
- additional WMTS Probes (by SpotInfo)
- use official OGC naming for OAFeat Probes (includes DB-upgrade)
- many bugfixes and security updates

Only a single DB-upgrade is required and only if your installation (DB) contains OGC OAFeat Resources and Probes, formerly called "WFS3".

See closed issues/merged PRs for related Milestone 0.9.0.

### 3.1.5 Running

Start using Flask's built-in WSGI server:

```
python GeoHealthCheck/app.py  # http://localhost:8000
python GeoHealthCheck/app.py 0.0.0.0:8881  # http://localhost:8881
python GeoHealthCheck/app.py 192.168.0.105:8957  # http://192.168.0.105:8957
```

This runs the (Flask) **GHC Webapp**, by default with the **GHC Runner** (scheduled healthchecker) internally. See also *Healthcheck Scheduling* for the different options running the **GHC Webapp** and **GHC Runner**. It is recommended to run these as separate processes. For this set **GHC_RUNNER_IN_WEBAPP** to *False* in your *site_config.py*. From the command-line run both processes, e.g. in background or different terminal sessions:

```
# run GHC Runner, here in background
python GeoHealthCheck/scheduler.py &

# run GHC Webapp for http://localhost:8000
python GeoHealthCheck/app.py
```

To enable in Apache, use `GeoHealthCheck.wsgi` and configure in Apache as per the main Flask documentation.

### 3.1.6 Running under a sub-path

By default GeoHealthCheck is configured to run under the root directory on the webserver. However, it can be configured to run under a sub-path. The method for doing this depends on the webserver you are using, but the general requirement is to pass Flask's `SCRIPT_NAME` environment variable when GeoHealthCheck is started.

Below is an example of how to use nginx and gunicorn to run GeoHealthCheck in a directory "geohealthcheck", assuming that you have nginx and gunicorn already set up and configured:

- In nginx add a section to the server block you are running GeoHealthCheck under:

```
location /geohealthcheck {
  proxy_pass http://127.0.0.1:8000/geohealthcheck;
}
```

- Include the parameter "-e SCRIPT_NAME=/geohealthcheck" in your command for running gunicorn:

```
gunicorn -e SCRIPT_NAME=/geohealthcheck app:app
```

### 3.1.7 Production Recommendations

#### Use Docker!

When running GHC in long-term production environment the following is recommended:

- use Docker, see the Docker Readme

Using Docker, especially with Docker Compose (sample files provided) is our #1 recommendation. It saves all the hassle from installing the requirements, upgrades etc. Docker (Compose) is also used to run the GHC demo site and almost all of our other deployments.

#### Use PostgreSQL

Although GHC will work with *SQLite*, this is not a good option for production use, in particular for reliability starting with GHC v0.5.0:

- reliability: **GHC Runner** will do concurrent updates to the database, this will be unreliable under *SQLite*
- performance: PostgreSQL has been proven superior, especially in query-performance

#### Use a WSGI Server

Although GHC can be run from the commandline using the Flask internal WSGI web-server, this is a fragile and possibly insecure option in production use (as also the Flask manual states). Best is to use a WSGI-server as stated in the Flask deployment options.

See for example the GHC Docker run.sh script to run the GHC Webapp with *gunicorn* and the GHC Runner run-runner.sh script to run the scheduled healthchecks.

#### Use virtualenv

This is a general Python-recommendation. Save yourself from classpath and library hells by using *virtualenv*! Starting with python 3.3 a venv script is provided and from python 3.6 the venv module is included in the standard library.

#### Use SSL (HTTPS)

As users and admin may login, running on plain http will send passwords in the clear. These days it has become almost trivial to automatically install SSL certificates with Let's Encrypt.

### 3.1.8 Running on RaspberryPi

Running GeoHealthCheck on a RaspberryPi works with Docker. But the standard Docker image cannot be used, because it is not targeted at RaspberryPi's ARM architecture. However, it is possible to manually build the Docker image for this architecture by replacing the Python base image of the Dockerfile with `arm32v7/python:3.7.9-alpine`. The image needs to be build on a machine with that architecuture. The RaspberryPi itself can be used for that, but it takes up to one hour.

## 3.2 Configuration

This chapter provides guidance for configuring a GeoHealthCheck instance.

### 3.2.1 Configuration Parameters

The core configuration is in `GeoHealthCheck/config_main.py`. Optionally override these settings for your instance in `instance/config_site.py`. You can specify a configuration file in the environment settings that will override settings in both previous files. The configuration options are:

- **SQLALCHEMY_DATABASE_URI**: the database configuration. See the SQLAlchemy documentation for more info
- **SQLALCHEMY_ENGINE_OPTION_PRE_PING**: DB Disconnect Handling, emitting a test statement on the SQL connection at the start of each connection pool checkout (default: `False`)
- **SECRET_KEY**: secret key to set when enabling authentication. Use the output of `paver create_secret_key` to set this value
- **GHC_RETENTION_DAYS**: the number of days to keep Run history
- **GHC_PROBE_HTTP_TIMEOUT_SECS**: stop waiting for the first byte of a Probe response after the given number of seconds
- **GHC_MINIMAL_RUN_FREQUENCY_MINS**: minimal run frequency for Resource that can be set in web UI
- **GHC_SELF_REGISTER**: allow registrations from users on the website
- **GHC_NOTIFICATIONS**: turn on email and webhook notifications
- **GHC_NOTIFICATIONS_VERBOSITY**: receive additional email notifications than just `Failing` and `Fixed` (default `True`)
- **GHC_WWW_LINK_EXCEPTION_CHECK**: turn on checking for OGC Exceptions in `WWW:LINK` Resource responses (default `False`)

- **GHC_LARGE_XML**: allows GeoHealthCheck to receive large XML files from the servers under test (default `False`). Note: setting this to `True` might pose a security risk (see this link).
- **GHC_ADMIN_EMAIL**: email address of administrator / contact- notification emails will come from this address
- **GHC_NOTIFICATIONS_EMAIL**: list of email addresses that notifications should come to. Use a different address to **GHC_ADMIN_EMAIL** if you have trouble receiving notification emails. Also, you can set separate notification emails to specific resources. Failing resource will send notification to emails from **GHC_NOTIFICATIONS_EMAIL** value and emails configured for that specific resource altogether.
- **GHC_SITE_TITLE**: title used for installation / deployment
- **GHC_SITE_URL**: full URL of the installation / deployment
- **GHC_SMTP**: configure SMTP settings if **GHC_NOTIFICATIONS** is enabled
- **GHC_RELIABILITY_MATRIX**: classification scheme for grading resource
- **GHC_PLUGINS**: list of Core/built-in Plugin classes or modules available on installation
- **GHC_USER_PLUGINS**: list of Plugin classes or modules provided by user (you)
- **GHC_PROBE_DEFAULTS**: Default *Probe* class to assign on "add" per Resource-type
- **GHC_METADATA_CACHE_SECS**: metadata, "Capabilities Docs", cache expiry time, default 900 secs, -1 to disable
- **GHC_REQUIRE_WEBAPP_AUTH**: require authentication (login or Basic Auth) to access GHC webapp and APIs (default: `False`)
- **GHC_BASIC_AUTH_DISABLED**: disable Basic Authentication to access GHC webapp and APIs (default: `False`), see below when to set to *True*
- **GHC_VERIFY_SSL**: perform SSL verification for Probe HTTPS requests (default: `True`)
- **GHC_RUNNER_IN_WEBAPP**: should the GHC Runner Daemon be run in webapp (default: `True`), more below
- **GHC_LOG_LEVEL**: logging level: 10=DEBUG 20=INFO 30=WARN(ING) 40=ERROR 50=FATAL/CRITICAL (default: 30, WARNING)
- **GHC_MAP**: default map settings
  - **url**: URL of TileLayer
  - **centre_lat**: Centre latitude for homepage map
  - **centre_long**: Centre longitude for homepage map
  - **maxzoom**: maximum zoom level
  - **subdomains**: available subdomains to help with parallel requests
  - **GEOIP**: configuration for the geolocator service plugin. Default is the ip-api.com api.

Example on overriding the configuration with an environment variable:

```
export GHC_SETTINGS=/tmp/my_GHC_settings.py
paver run_tests
```

As an example: the *my_GHC_settings.py* file can contain a single line to define a test database:

```
SQLALCHEMY_DATABASE_URI='sqlite:////tmp/GHCtest.db'
```

**NOTE**: do not forget to reset the environment variable afterwards.

### 3.2.2 Email Configuration

A working email-configuration is required for notifications and password recovery. This can sometimes be tricky, below is a working configuration for the GMail account *my_gmail_name@gmail.com*.

```
GHC_SMTP = {
    'server': 'smtp.gmail.com',
    'port': 587,
    'tls': True,
    'ssl': False,
    'username': 'my_gmail_name@gmail.com',
    'password': '<my gmail password>'
}
```

In your Google Account settings for that GMail address you should turn on *"Allow less secure apps"* as explained here.

### 3.2.3 Healthcheck Scheduling

Healthchecks (Runs) for each Resource can be scheduled via *cron* or (starting with v0.5.0) by running the **GHC Runner** app standalone (as daemon) or within the **GHC Webapp**.

#### Scheduling via Cron

**Applies only to pre-0.5.0 versions.**

Edit the file `jobs.cron` so that the paths reflect the path to the virtualenv. Set the first argument to the desired monitoring time step. If finished editing, copy the command line calls e.g. `/YOURvirtualenv/bin_or_SCRIPTSonwindows/python /path/to/GeoHealthCheck/GeoHealthCheck/healthcheck.py run` to the commandline to test if they work sucessfully. On Windows - do not forget to include the ''.exe." file extension to the python executable. For documentation how to create cron jobs see your operating system: on *NIX systems e.g. `crontab -e` and on windows e.g. the nssm.

NB the limitation of cron is that the per *Resource* schedule cannot be applied as the cron job will run healthchecks on all *Resources*.

#### GHC Runner as Daemon

In this mode GHC applies internal scheduling for each individual *Resource*. This is the preferred mode as each *Resource* can have its own schedule (configurable via Dashboard) and *cron* has dependencies on local environment. Later versions may phase out cron-scheduling completely.

The **GHC Runner** can be run via the command *paver runner_daemon* or can run internally within the **GHC Webapp** by setting the config variable **GHC_RUNNER_IN_WEBAPP** to *True* (the default). NB it is still possible to run GHC as in the pre-v0.5.0 mode using cron-jobs: just run the **GHC Webapp** with **GHC_RUNNER_IN_WEBAPP** set to *False* and have your cron-jobs scheduled.

In summary there are three options to run GHC and its healthchecks:

- run **GHC Runner** within the **GHC Webapp**: set **GHC_RUNNER_IN_WEBAPP** to *True* and run only the GHC webapp
- (recommended): run **GHC Webapp** and **GHC Runner** separately (set **GHC_RUNNER_IN_WEBAPP** to *False*)

- (deprecated): run **GHC Webapp** with **GHC_RUNNER_IN_WEBAPP** set to *False* and schedule healthchecks via external cron-jobs

### 3.2.4 Language Translations

GHC supports multiple languages by using [Babel](http://babel.pocoo.org) with [Flask-Babel](https://pythonhosted.org/Flask-Babel/).

*"Babel is an integrated collection of utilities that assist in internationalizing and localizing Python applications, with an emphasis on web-based applications."*

#### Enabling/Disabling a Language

Open the file *GeoHealthCheck/app.py* and look for the language switcher (e.g. 'en','fr') and remove or add the desired languages. In case of a new language, a new translation file (called a **.po*) has to be added as follows:

- make a copy of one of the folders in *GeoHealthCheck/translations/*;
- rename the folder to the desired language (e.g. *'de'* for German) using the language ISO codes
- edit the file *<your_lang>/LC_MESSAGES/messages.po*, adding your translations to the *msgstr*

Don't forget the change the specified language in the *messages.po* file as well. For example the *messages.po* file for the German case has an English *msgid* string, which needs to be translated in *msgstr'* as seen below.

```
#: GeoHealthCheck/app.py:394
msgid "This site is not configured for self-registration"
msgstr "Diese Webseite unterstützt keine Selbstregistrierung"
```

#### Compiling Language Files

At runtime compiled versions, *.mo* files, of the language-files are used. Easiest to compile is via: *paver compile_translations* in the project root dir. This basically calls *pybabel compile* with the proper options. Now you can e.g. test your new translations by starting GHC.

#### Updating Language Files

Once a language-file (*.po*) is present, it will need updating as development progresses. In order to know what to update (which strings are untranslated) it best to first update the *messages.po* file with all language strings, their location(s) within project files and whether the translation is missing. Missing translations will have *msgstr ""* like in this excerpt:

```
#: GeoHealthCheck/notifications.py:245 GeoHealthCheck/notifications.py:247
msgid "Passing"
msgstr "Jetzt geht's"

#: GeoHealthCheck/plugins/probe/ghcreport.py:115
msgid "Status summary"
msgstr ""
```

Next all empty *msgstr* can be filled.

Updating is easiest using the command *paver update_translations* within the root dir of the project. This will basically call *pybabel extract* followed by *pybabel update* with the proper parameters.

### 3.2.5 Customizing the Score Matrix

GeoHealthCheck uses a simple matrix to provide an indication of overall health and / or reliability of a resource. This matrix drives the CSS which displays a given resource's state with a colour. The default matrix is defined as follows:

| low | high | score/colour |
|---|---|---|
| 0 | 49 | red |
| 50 | 79 | orange |
| 80 | 100 | green |

To adjust this matrix, edit **GHC_RELIABILITY_MATRIX** in `instance/config_site.py`.

### 3.2.6 Securing GHC Webapp

In some cases it is required that only logged-in (authenticated) users like the `admin` user can access the entire GHC webapp and its APIs. In that case the config setting **GHC_REQUIRE_WEBAPP_AUTH** should be set to `True`. (version 0.7+). Non-authenticated users will be presented with the login screen. Initially only the `admin` user will be able to login, but it is possible to register and allow additional users by registering these within the `admin` login session. Note that password reset is still enabled. For remote REST API calls standard HTTP Basic Authentication (via the HTTP *Authentication* request header) can be used.

In some cases where an external web- or proxy server provides HTTP Basic Authentication, a conflict may arise when GHC also checks the *Authorization* HTTP header used for the external Basic Auth. In those cases GHC Basic Auth checking can be disabled using the **GHC_BASIC_AUTH_DISABLED** to *True*. TODO: provide API Token auth to allow both external Basic Auth and GHC API auth.

## 3.3 Administration

This chapter describes maintenance tasks for the administrator of a GHC instance. There is a separate *User Guide* that provides guidance to the end-user to configure the actual Resource healthchecks.

Each of the sections below is geared at a specific administrative task area.

### 3.3.1 Database

For database administration the following commands are available.

#### create db

To create the database execute the following:

Open a command line, (if needed activate your virtualenv), and do

```
python GeoHealthCheck/models.py create
```

#### drop db

To delete the database execute the following, however you will loose all your information. So please ensure backup if needed:

Open a command line, (if needed activate your virtualenv), and do

```
python GeoHealthCheck/models.py drop
```

Note: you need to create a Database again before you can start GHC again.

#### load data

To load a JSON data file, do (WARN: deletes existing data!)

```
python GeoHealthCheck/models.py load <datafile.json> [y/n]
```

Hint: see *tests/data* for example JSON data files.

#### export data

Exporting database-data to a .json file with or without Runs is still to be done.

Exporting Resource and Run data from a running GHC instance can be effected via a REST API, for example:

- all Resources: https://demo.geohealthcheck.org/json (or as CSV)
- one Resource: https://demo.geohealthcheck.org/resource/1/json (or CSV)
- all history (Runs) of one Resource: https://demo.geohealthcheck.org/resource/1/history/json (or in csv)

NB for detailed reporting data only JSON is supported.

### 3.3.2 User Management

During initial setup, a single *admin* user is created interactively.

Via the **GHC_SELF_REGISTER** config setting, you allow/disallow registrations from users on the webapp (UI).

#### Passwords

Passwords are stored encrypted. Even the same password-string will have different "hashes". There is no way that GHC can decrypt a stored password. This can become a challenge in cases where a password is forgotten and somehow the email-based reset is not available nor working. In that case, password-hashes can be created from the command-line using the Python library passlib within an interactive Python-shell as follows:

```
$ pip install passlib
# or in Debian/Ubuntu: apt-get install python-passlib

python
>>> from passlib.hash import pbkdf2_sha256
>>>
>>> hash = pbkdf2_sha256.hash("mynewpassword")
>>> print(hash)
```

(continues on next page)

(continued from previous page)

```
'$pbkdf2-sha256$29000$da51rlVKKWVsLSWEsBYCoA$2/shIdqAxGJkDq6TTeIOgQKbtYAOPSi5EA3TDij1L6Y'
>>> pbkdf2_sha256.verify("mynewpassword", hash)
True
```

Or more compact within the root dir of your GHC installation:

```
>>> from GeoHealthCheck.util import create_hash
>>> create_hash('mynewpassword')
'$pbkdf2-sha256$29000$8X4PAUAIAcC4V2rNea9Vqg$XnMx1SfEiBzBAMOQOOC7uxCcyzVuKaHENLj3IfXvfu0'
```

Or even more compact within the root dir of your GHC installation via Paver:

```
$ paver create_hash -p mypass
---> pavement.create_hash
Copy/paste the entire token below for example to set password
$pbkdf2-sha256$29000$FkJoTYnxPqc0pjQG4HxP6Q$C3SZb8jqtM7zKS1DSLcouc/CL9XMI9cL5xT6DRTOEd4
```

Then copy-paste the hash-string into the *password*-field of the User-record in the User-table. For example in SQL something like:

```
$ sqlite3 data.db
# or psql equivalent for Postgres

sqlite> UPDATE user SET password = '<above hash-value>' WHERE username == 'myusername';
```

### 3.3.3 Build Documentation

Open a command line, (if needed activate your virtualenv) and move into the directory `GeoHealthCheck/doc/`. In there, type `make html` plus ENTER and the documentation should be built locally.

## 3.4 User Guide

This chapter provides guidance for configuring GeoHealthCheck's (GHC) actual tasks: healthchecking API services on (OGC) URL Endpoints. It is written from the perspective of the end-user who interacts with GHC's webapp (UI).

This chapter contains figures of screenshots. Click on a figure to see a larger version of the image. Use the back-button to get back into this document. This chapter can also be found by pressing Help (top menu) within the Web UI.

### 3.4.1 Terminology

The following terminology applies:

- *Resource*: basically an endpoint URL, like an OGC WMS, FTP URL, or plain old weblink. For OGC-Resources this is always the root-URL, **not the Capabilities-URL**. Each Resource has a Type (see below).
- *Probe*: each *Resource* is tested via one or more *Probes*, a Probe is typically a single HTTP request, like *GetCapabilities*, *GetMap* etc. Each *Resource* (Type) has a default *Probe*.
- *Check*: each *Probe* invokes one or more *Checks*, typically on the HTTP response. For example if a WMS *GetMap* returns an image object.
- *Run*: the execution and scoring of a single *Probe*. Its *Checks* determine the *Run* outcome.

- A *Run* in addition has a single verdict: *Ok* or *NotOk*.
- Each *User* owns one or more *Resources*

The main user task within the web UI is to manage (add, update, delete) a set of *Resources*. For each *Resource* its various properties (scheduling, notifications, tags etc) and *Probes* is managed. Subsequently, for each *Probe* its various *Checks* are managed.

### 3.4.2 Registration

If the administrator of the GHC instance has enabled User Registration (**GHC_SELF_REGISTER** = *True*), any person can register and manage *Resources* on that GHC instance. A User can only manage its own Resources. The Admin user can always edit/manage any *Resource*.

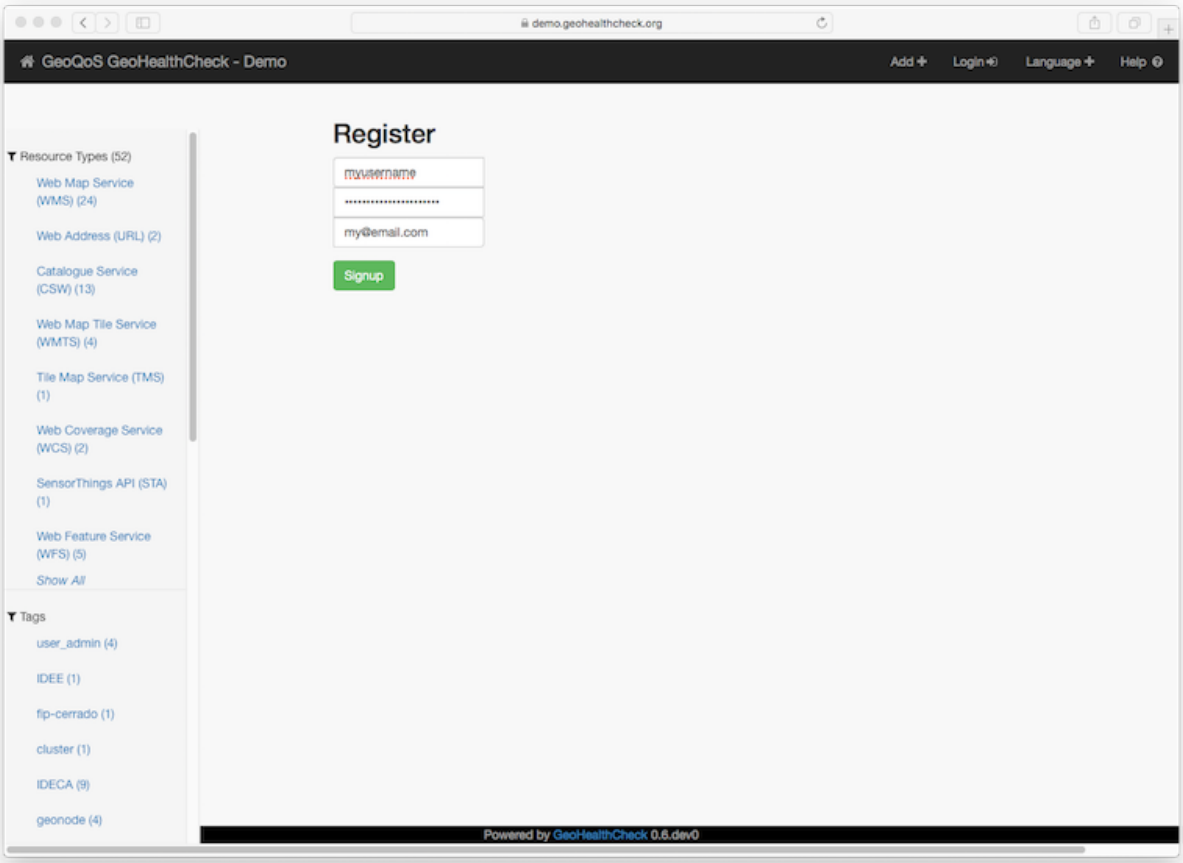


Fig. 1: *Registration*

Start registration by clicking Login in menu and then the *Register* link within the Login screen. When registering, a working email adress is required if you want to receive Resource notifications by email and for password-recovery.

### 3.4.3 Home Screen

The initial home screen always shows failing Resources (if any). The badges on the top show percentages:

- *Operational*: percentage of all Resources that is currently "up"/healthy
- *Failing*: percentage of all Resources that is currently"down"/failing
- *Reliable*: percentage of time that Resources are "up"/healthy within the retention window

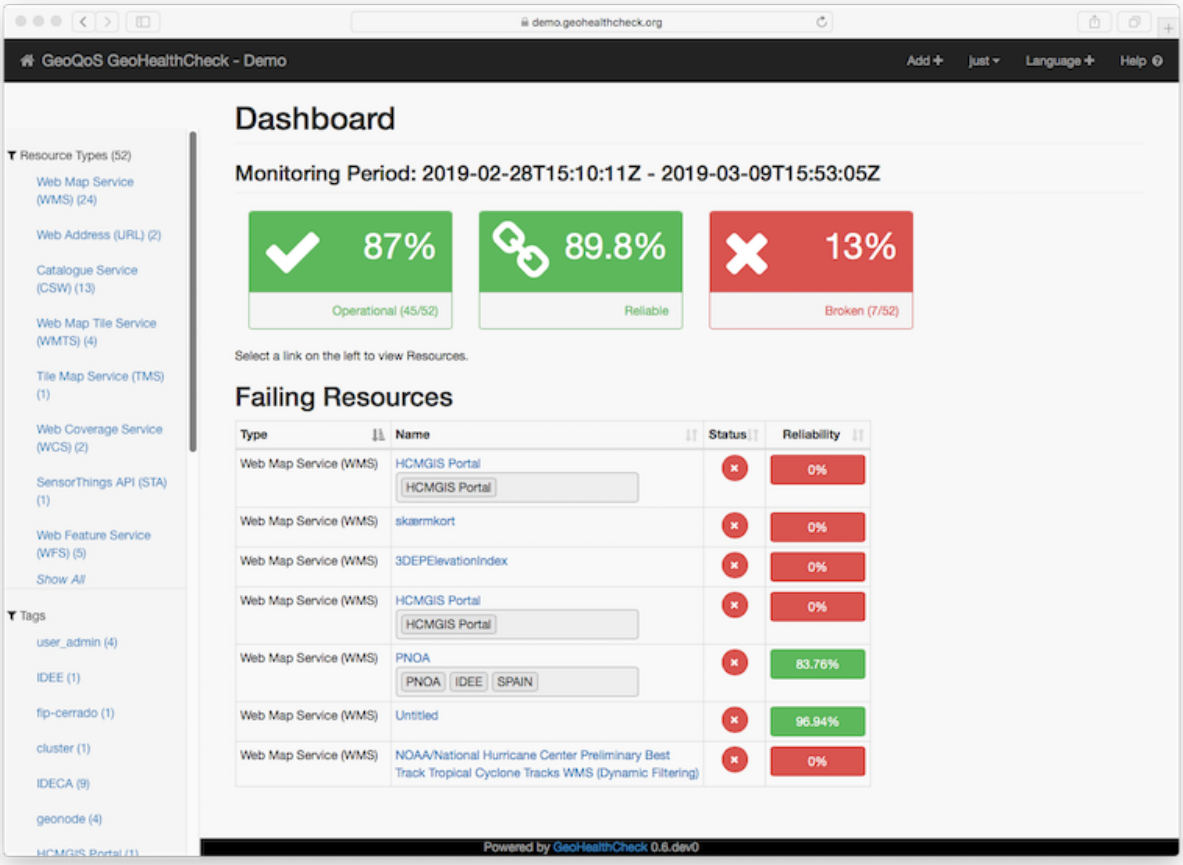


Fig. 2: *Home Screen*

Using the vertical menu items on the left different lists of Resources can be shown: either by Resource Type (like WMS in Figure below), or by Tags (discussed later).

Clicking the Home icon (top left) brings back the initial home screen.

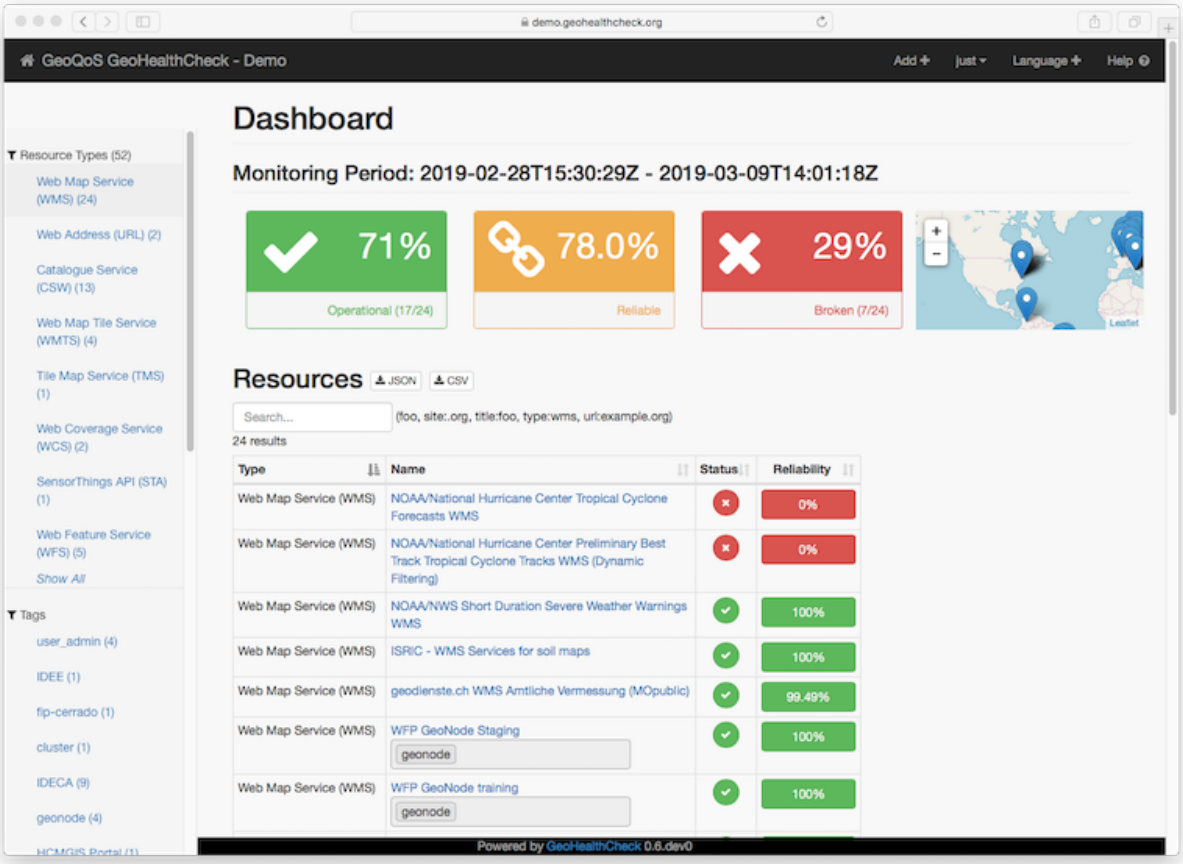


Fig. 3: *WMS Type Resource List*

### 3.4.4 Adding Resources

Click the Add+ button in the top menu to add a new Resource.

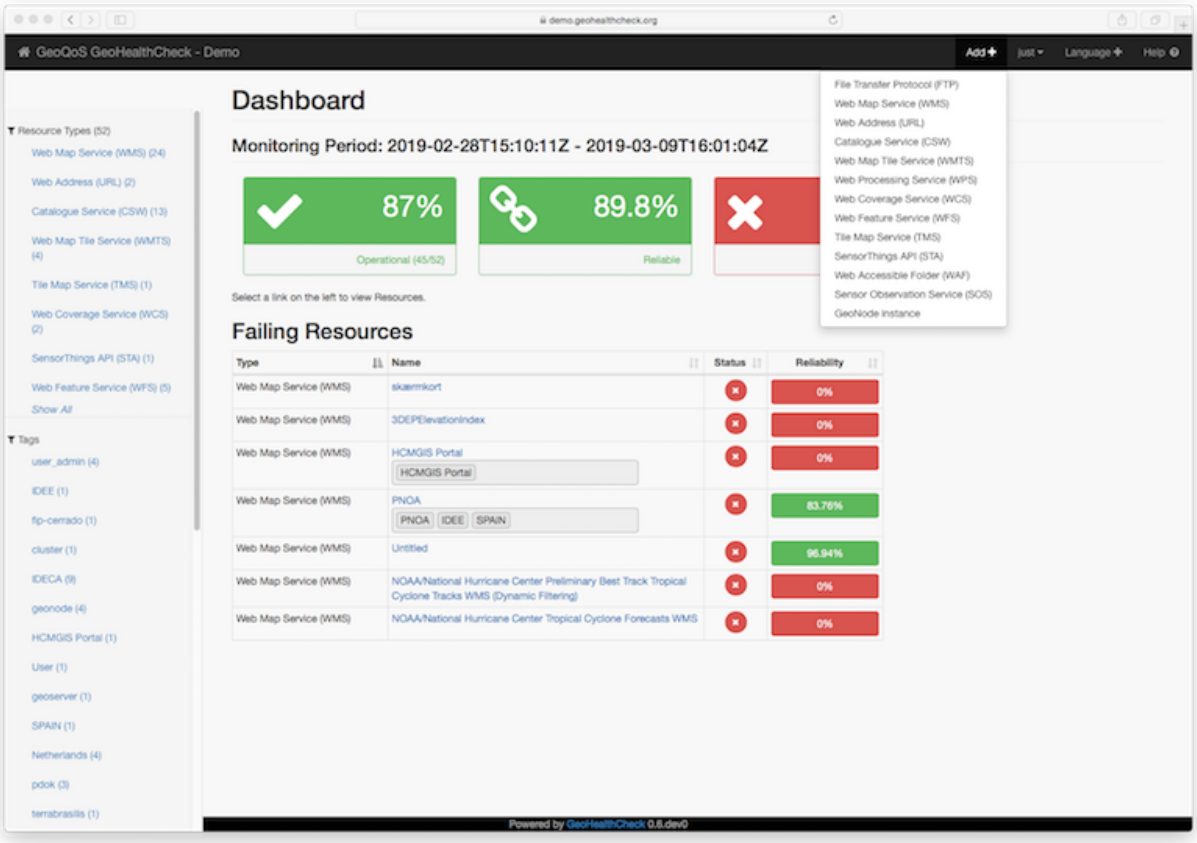


Fig. 4: *Add Resource - Select Type*

First choose a Resource Type from the dropdown menu. The following Resource Types are available:

- Web Map Service (WMS)
- Web Feature Service (WFS)
- Web Map Tile Service (WMTS)
- Tile Map Service (TMS)
- Web Coverage Service (WCS)
- Catalogue Service (CSW)
- Web Processing Service (WPS)
- Sensor Observation Service (SOS)
- SensorThings API (STA)
- OGC Features API (OAFeat)
- Web Accessible Folder (WAF)
- Web Address (URL)

- File Transfer Protocol (FTP)
- GeoNode autodiscovery (see *GeoNode Resource Type*)
- GeoHealthCheck Reporter (GHC-R) (see *GeoHealthCheck Reporter Type*)

Next fill in the URL and optional tags for the Resource.

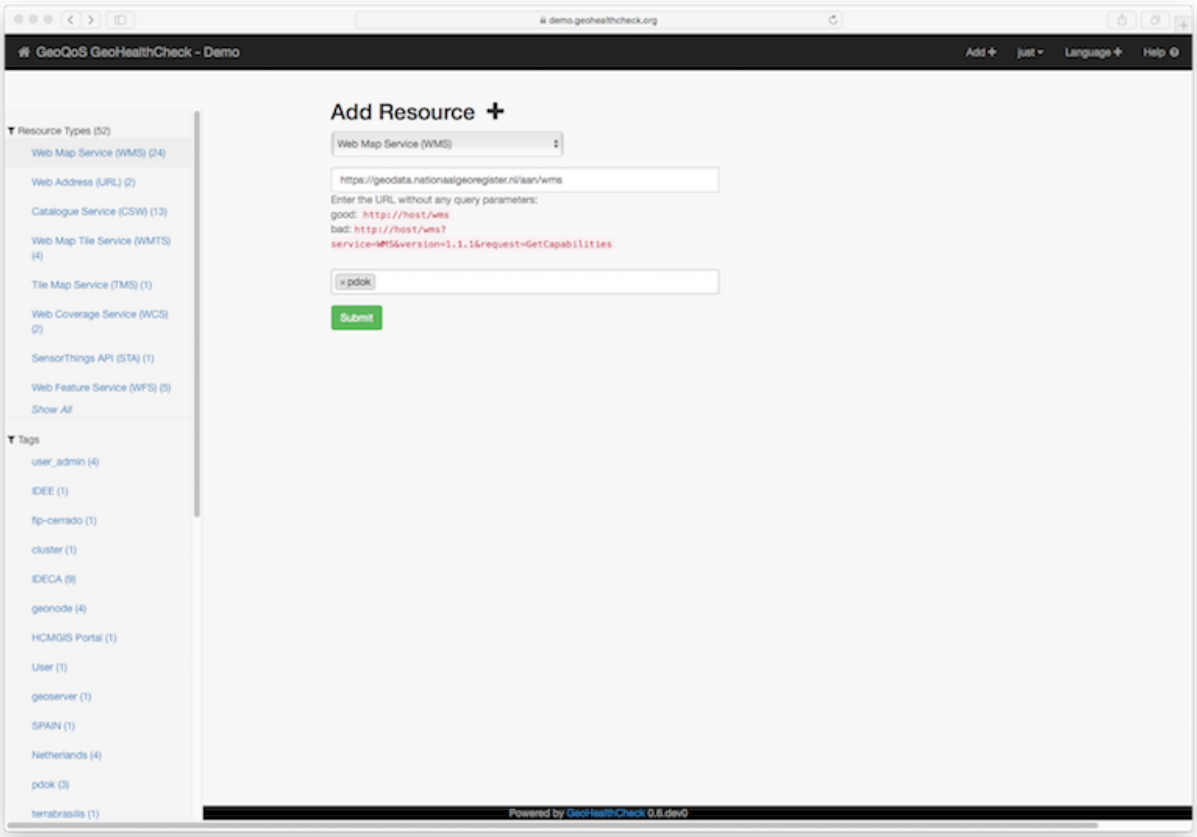


Fig. 5: *Add Resource - specify URL and optional Tags*

Fill in the endpoint URL, like an OGC WMS, FTP URL or a weblink for the Web Address Type. For OGC-Resources **this should be the root-endpoint-URL, not the Capabilities-URL**.

You can add new or existing tags as well here. On Submit, the Resource will get a single default Probe assigned. For OGC-Resources this is usually a *CapabilitiesProbe*. If successful you are directed to the Resource Edit screen (see next).

### 3.4.5 Editing Resources

Open the Resource details by clicking its name in the Resources list at the Dashboard/Home page. Under the Resource title is a blue Edit button (if you own the Resource or as admin). When Adding a Resource (see above), you are automatically directed to the Resource Edit Screen.

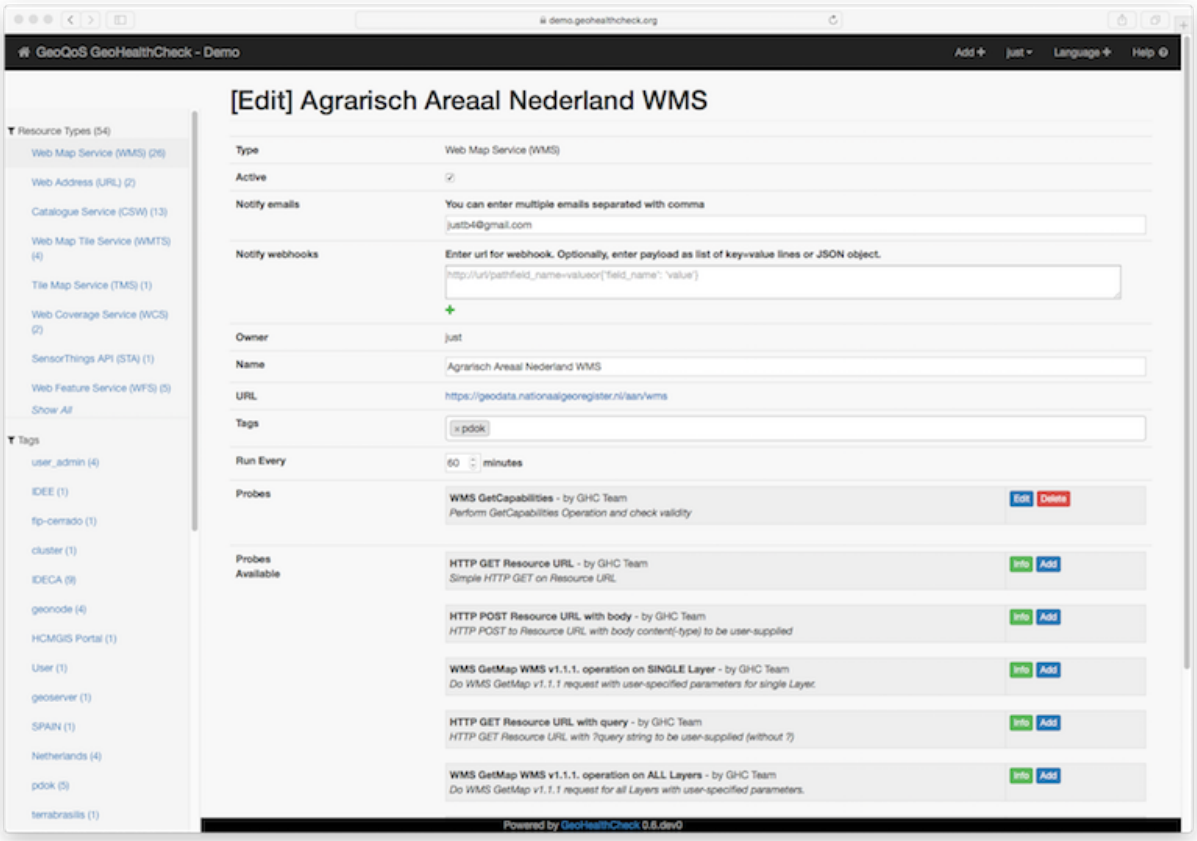


Fig. 6: *Edit Resource - Basics*

The following general aspects of a *Resource* can be edited:

- Resource name (initial Name may come from Capabilities or HTML *title* element if present)
- Resource Tags
- Resource active/non-active (makes Probes (in)active, e.g. when repairing a Resource)
- Notification: recipient(s) for email (see *Per-Resource Notifications*)
- Notification: target(s) and parameters for webhooks (advanced: see *Per-Resource Notifications*)
- Resource run schedule, "Run Every" N minutes
- Optional HTTP authentication (*Basic* or *Bearer Token*) for secured Resource endpoints

By default, when a Resource is created, the owner's email will be added to the email-notifications.

The most important/functional aspects for a Resource are its Probes.

- Manage Probes for the Resource: select a Probe from "Probes Available"
- Optionally edit Probe parameters, fixed values have grey background

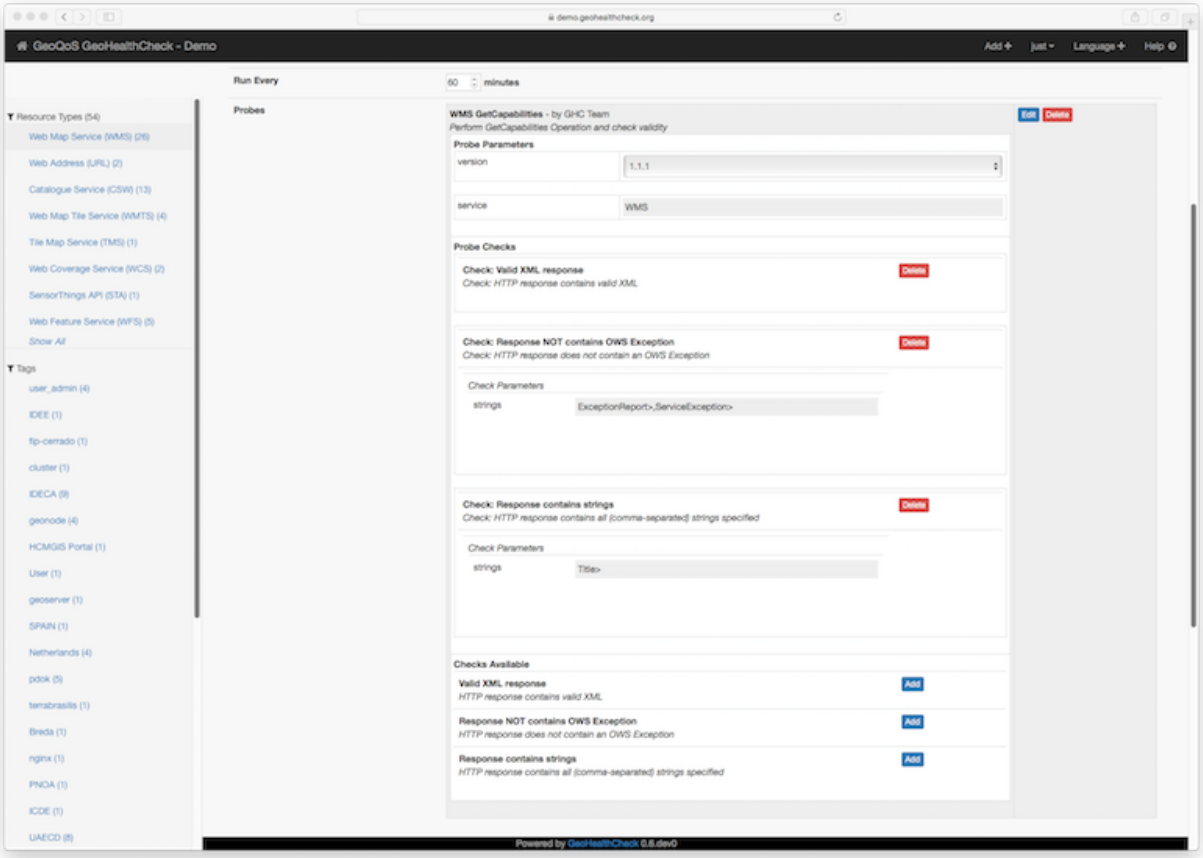


Fig. 7: *Edit Resource - Edit Probe*

- Manage Checks for the Probe, add by selecting from "Checks Available"
- Optionally edit Check parameters

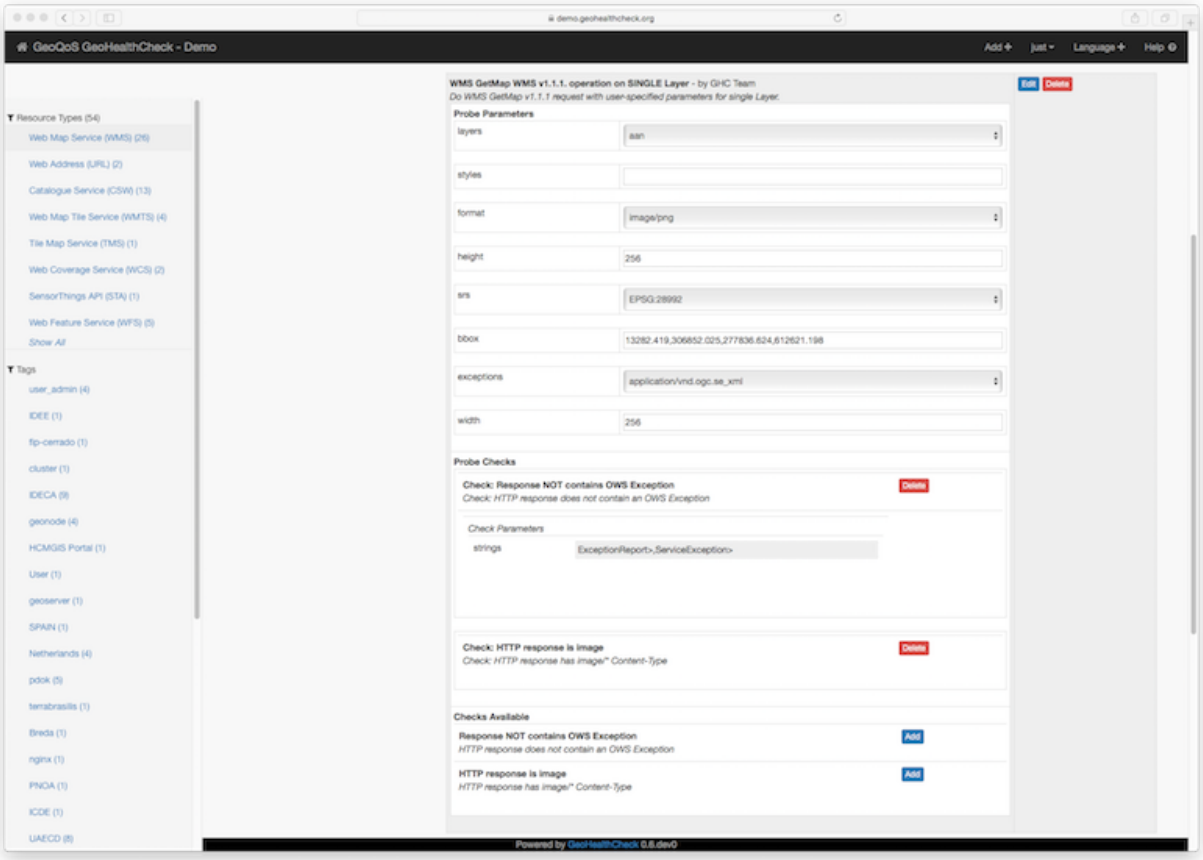


Fig. 8: *Edit Resource - Add Probe*

Note that all "Edit" buttons with Probes and Checks are toggles to show/hide a Probe and/or Check.

Click Save to save all Resource settings and then click Test to test your Probes and Checks. NB Test only works on the Resource settings as saved! So Save, then Test.

### 3.4.6 Deleting Resources

Open the Resource details by clicking its name in the Resources list. Under the Resource title is a red Delete button.

### 3.4.7 Tagging

Each Resource can be tagged with multiple tags. This provides a handy way to structure your Resources into any kind of categories/groups, like *Production* and *Test*, common servers any other grouping.

### 3.4.8 Failure Analysis

As history builds up for each Resource, Users may get notified, usually by email, when one or more Probes fail for a Resource (and again when the Resource is healthy again). In this section we analyse a failing Resource (WMS).

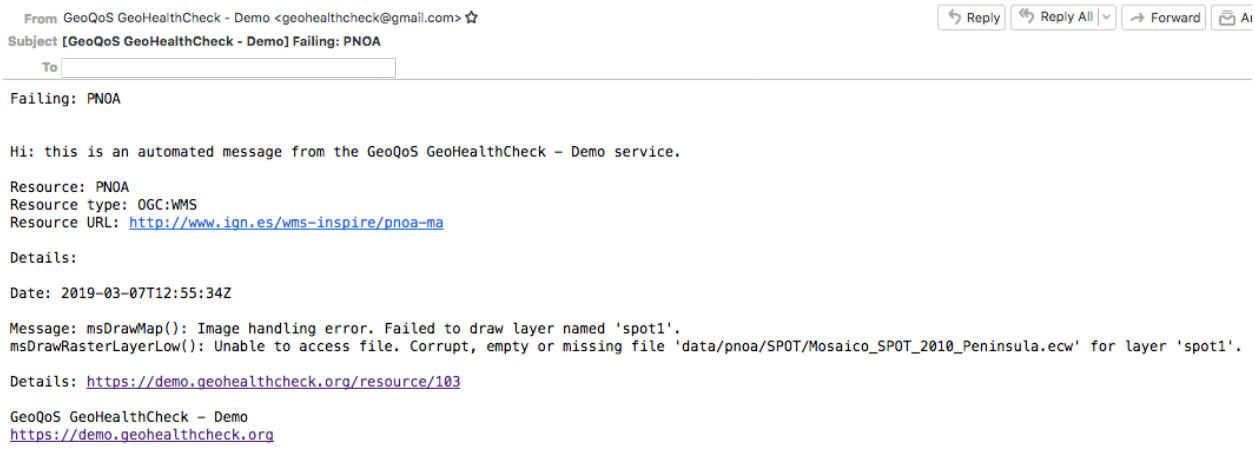


Fig. 9: *Email Notification - Failing Resource*

This kind of email is received when the Resource has failed. We can already see in the message (showing the last message from one or more failing Probes) that something is wrong with an *.ecw* (compressed raster image) file within the WMS. We can click on the link to go directly to the Resource view within the GHC demo site.

NB: Dependent on the **GHC_NOTIFICATIONS_VERBOSITY** config setting, this email is received only once on the first failure (False) or on each failing Run.

In order to analyse "what happened", the graph shown in the Resource view can be inspected. Below, this WMS Resource is shown.

As can be seen, this WMS Resource is now up (*Last Run Result* on top right) but has a Reliability of 57.56 percent. This means that within the retention window (one week for the demo site) it has been down for about half of the time. This Resource als has quite some Probes active, so is thoroughly tested each hour.

Scrolling down within the Resource view the History Graph is shown. Each Resource Run is presented by a dot. Red dots indicate that one or more Probes have failed in a Run. Green that all Probes gave success.

We see that this WMS has failed from somewhere on March 7, 2019 until March 11, 2019 when it became healthy again (last green dot right). Also the Resource has been made inactive for some time during failure as no dots are shown. The WMS itself may have been up though all the time! The is a classic case: the Capabilities Probe always succeeds, but more detailed WMS GetMap Probes may have failed. We can inspect this in more detail from the history graph.

The History Graph can be explored in detail by simply hovering the mouse over its dots. Also the graph can be zoomed in/out and panned, even with the mouse wheel. For each dot the overall result is shown: Date/Time of Run, Duration

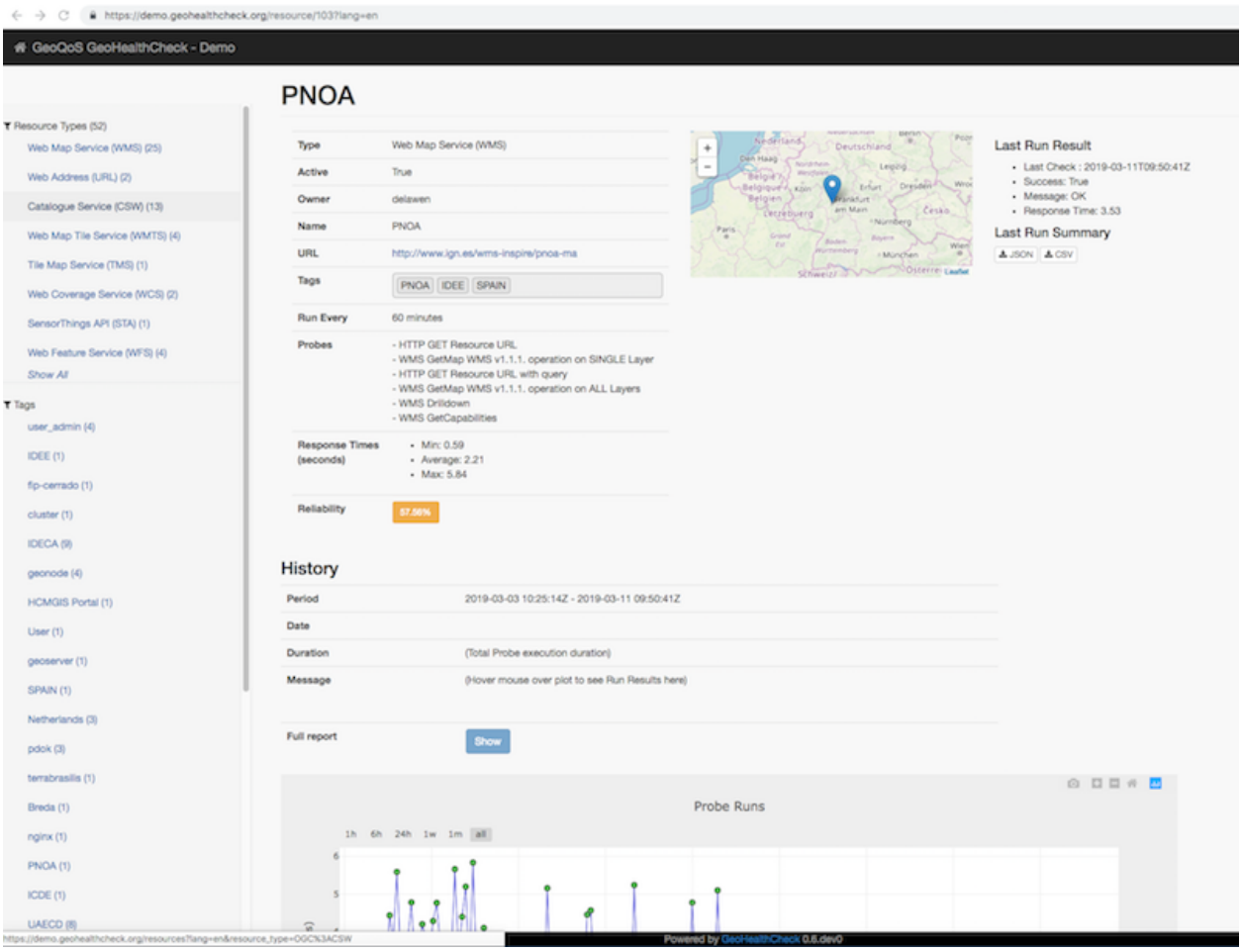


Fig. 10: *WMS Type Resource View*

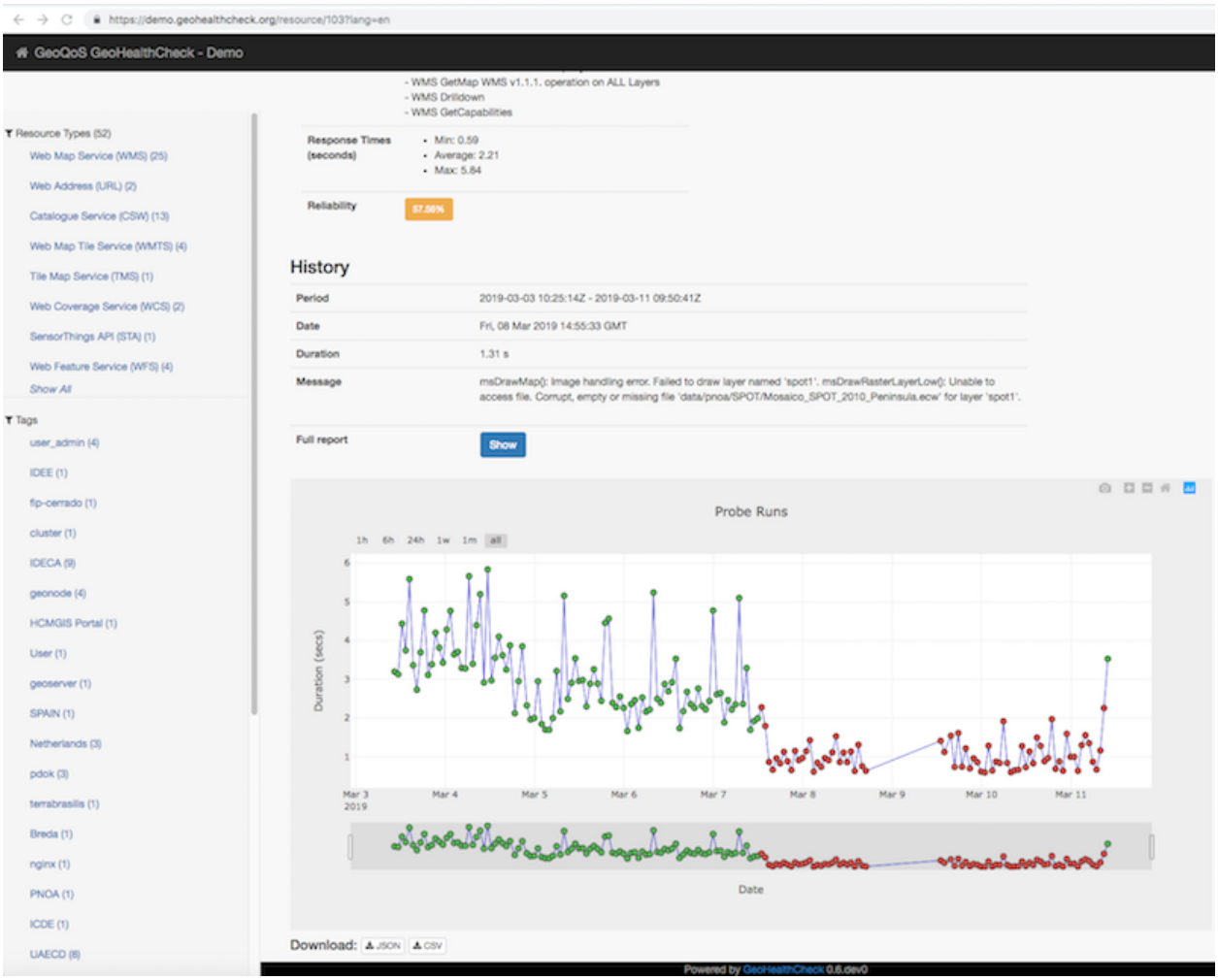


Fig. 11: *WMS Type Resource View - History*

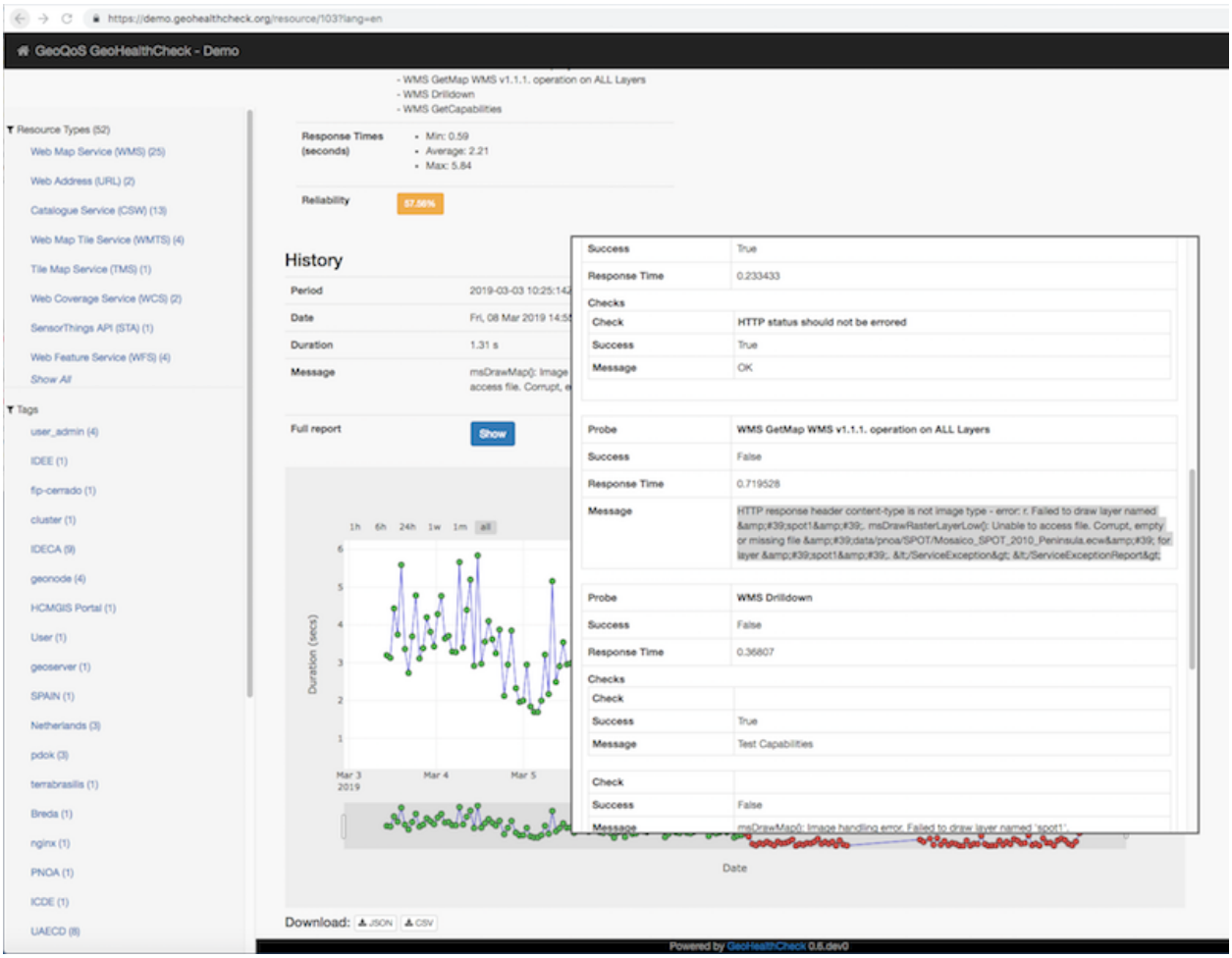


Fig. 12: *WMS Type Resource View - History Detail*

(of all Probe runs), Message (Ok, or error message). By clicking the Show-button the full Run report, i.e. all Probe and Check results for that Date/Time are shown in a popup panel.

Here we can see more detail for this WMS: the WMS *GetMap* and two other WMS *GetMap*-related Probes like *WMS-DrillDown*, have failed, because an image file (.ecw file) could not be opened/found. This is a classical example why you would need GeoHealthCheck: GetCapabilities always succeeds on the service endpoint, but more detailed *GetMap* requests fail!

The last run within the history is again success, so apparently the underlying issues have been repaired and the WMS is healthy again! For the last Run (green dot in graph), the email below is received.

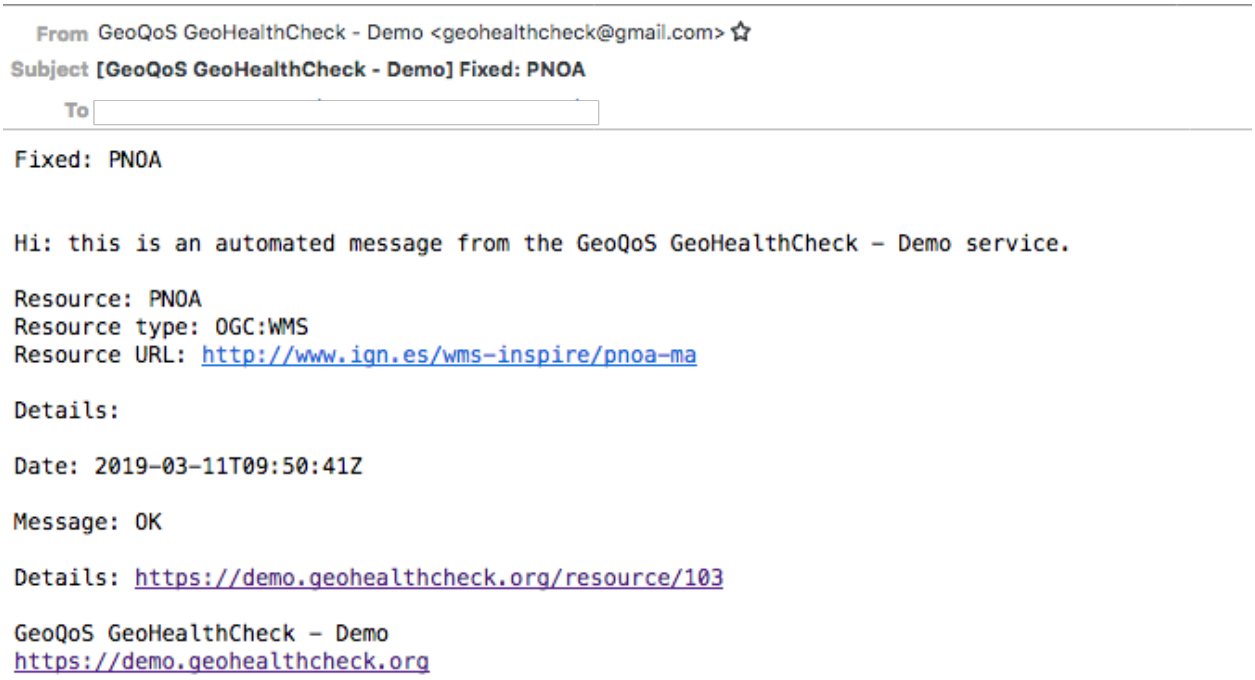


Fig. 13: *Email Notification - Resource Ok Again*

This kind of email is received when the Resource is healthy (Ok, True) again.

### 3.4.9 Per-Resource Notifications

Notifications for each Resource can be configured in the Resource edit form:

Note: if left empty, the global (email-)notification settings will apply.

Two notification channel-types are currently available:

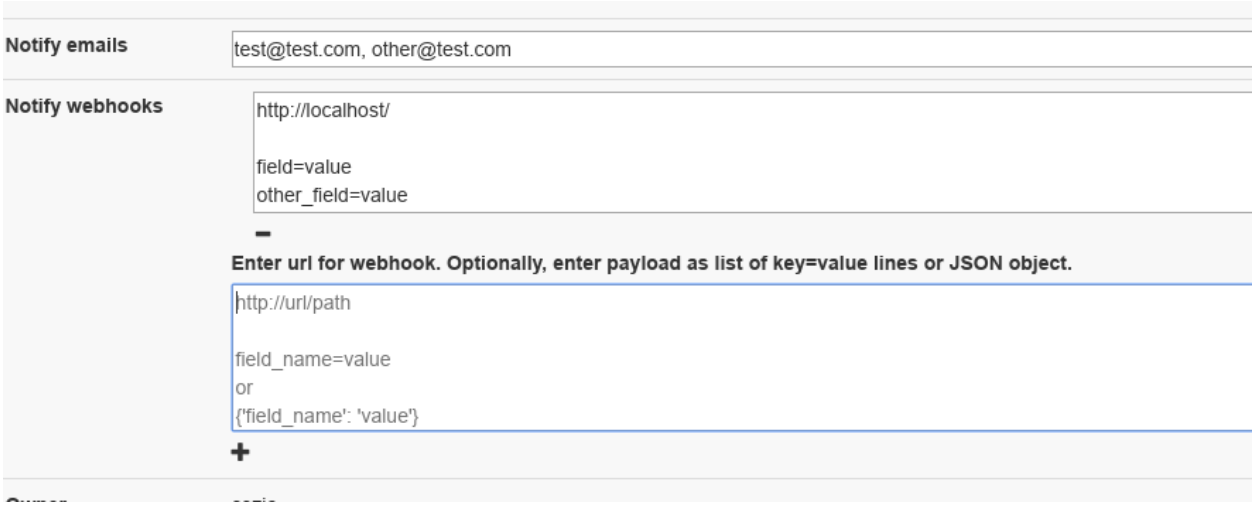


Fig. 14: *GHC notifications configuration*

### Email

Notifications can be sent to designated emails. If set in the config, GeoHealthCheck will send notifications for all resources to emails defined in **GHC_NOTIFICATIONS_EMAIL**. Additionally, each resource can have arbitrary list of emails (filled in **Notify emails** field in edit-form). By default, when a Resource is created, the owner's email is added to the list. The editing User can add any email address, even for Users not registered in the GeoHealthCheck instance. When editing an email-list for a resource, the user will get address suggestions based on emails added for other Resources by that User. Multiple emails should be separated with comma (,) chars.

### Webhook

Notifications can be also sent as webhooks (through *POST* requests). A Resource can have an arbitrary number of webhooks configured.

In the edit form, the User can configure webhooks. Each webhook should be entered in a separate field. Each webhook should contain at least a URL to which the *POST* request will be send. GeoHealthCheck will send following fields with that request:

| Form field | Field type | Description |
|---|---|---|
| ghc.result | string | Descriptive result of failed test |
| ghc.resource.url | URL | Resource's url |
| ghc.resource.title | string | Resource's title |
| ghc.resource.type | string | Resource's type name |
| ghc.resource.view | URL | URL to resource data in GeoHealthCheck |

A webhook configuration can hold additional form payload that will be sent along with GHC fields. Syntax for configuration:

- first line should be URL to which webhook will be sent
- second line should be empty
- third line (and subsequent) are used to store the custom payload, and should contain either: * pairs of field and value in separate lines (*field*=*value*) * a JSONified object, whose properties will be used as form fields

Configuration samples:

- URL-only:

```
http://server/webhook/endpoint
```

- URL with fields as field-value pairs:

```
http://server/webhook/endpoint

foo=bar
otherfield=someothervalue
```

- URL with payload as JSON:

```
http://server/webhook/endpoint

{"foo":"bar","otherfield":"someothervalue"}
```

### 3.4.10 GeoNode Resource Type

*GeoNode* Resource is a virtual Resource. It represents one GeoNode instance, but underneath auto-discovery is applied of OWS endpoints available in that instance. Note, that the OWS auto-discovery feature is optional, and you should check if your GeoNode instance has this feature enabled.

When a adding *GeoNode instance* Resource, you have to enter the URL to the GN instance's home page. GeoHealthCheck will construct the URLs to fetch the list of OWS endpoints and create relevant Resources (WMS, WFS, WMTS, and other OWS Resources). It will check all endpoints provided by the GeoNode API, and will reject those which responded with an error.

All Resources added in this way will have at least one tag, which is constructed with the template: *GeoNode _hostname_*, where *_hostname_* is a host name from url provided. For example, let's assume you add GeoNode instance that is served from *demo.geonode.org*. All resources created in this way will have *GeoNode demo.geonode.org* tag.

### 3.4.11 GeoHealthCheck Reporter Type

The *GeoHealthCheck Reporter (GHC-R)* Resource type allows users to receive a regular status summary report by email for the Resources in any local or remote GHC instance. Typically this is used for the local GHC instance. To setup:

- in top-menu select *Add | GeoHealthCheck Reporter (GHC-R)*
- in *Add Resource* screen add the site URL of the target GHC instance

Then in *Resource Edit* screen

- if the target GHC instance requires authentication: in *Authentication* form field select *Basic* and fill in username and password
- set *Run Every* field to a high value, typically 1440 minutes (every 24 hour)
- click *Edit* button for the assigned *GHC Email Reporter*
- set *email* field in *Probe Parameters* to one or more email adresses (comma-separated)

**Warning:** The Resource form-field *"Notify emails"* is **not** the target for the Email Report! It is used to report any possible errors for report assembly and email delivery.

**Warning:** Summary email reports may in cases be marked as spam by your email provider. In those cases you should greenlist (mark as non-spam) the sender email address.

**Note:** Tip: The *GeoHealthCheck Reporter Probe* uses the */api/v1.0/summary* API call. You can always get the last status report message as text via the URL *<GHC Instance URL>/api/v1.0/summary.txt* for example https://demo.geohealthcheck.org/api/v1.0/summary.txt

### 3.4.12 Resource Authentication

Resource authentication allows a user to optionally add credentials to access a secured *Resource* endpoint. Currently two (HTTP) authentication methods are supported:

- *Basic Authentication*: "classic" username and password based
- *Bearer Token*: single token based

The default is *None*, i.e. no authentication.

Within the *Resource* Edit screen, whenever a user selects an authentication method, the related input form-fields are shown. Any credentials added are stored encrypted.

Resource Authentication has been implemented using GHC Plugins, thus may be extended at will.

## 3.5 Architecture

GeoHealthCheck (GHC) consists of three cooperating parts as depicted in the figure below.

The **GHC Webapp** provides the Dashboard where users configure web services (Resources) for (scheduled) health-checks and view the status of these checks. The **GHC Runner** performs the actual health-checks and notifications, based on what the user configured via the GHC Webapp.

The third part is the **Database** that stores all information like users, resources, checks, schedules, results etc.

The **GHC Webapp** is run as a standard Python (Flask) webapp. The **GHC Runner** runs as a daemon process using an internal scheduler to invoke the actual healthchecks.

**GHC Webapp** and **GHC Runner** can run as separate processes (preferred) or both within the **GHC Webapp** process. This depends on a configuration option. If **GHC_RUNNER_IN_WEBAPP** is set to True (the default) then the **GHC Runner** is started within the **GHC Webapp**.

A third option is to only run the **GHC Webapp** and have the **GHC Runner** scheduled via *cron*. This was the (only) GHC option before v0.5.0 and will be phased out as starting with v0.5.0, per-Resource scheduling was introduced and *cron* support is highly platform-dependent (e.g. hard to use with Docker-based technologies).

Dependent on the database-type (Postgres or SQLite) the **Database** is run within the above processes (SQLite) or as a separate process (Postgres).

So in the most minimal setup, i.e. **GHC Webapp** and **GHC Runner** running within a single process and using SQLite as the database, only a single GHC process instance is required.

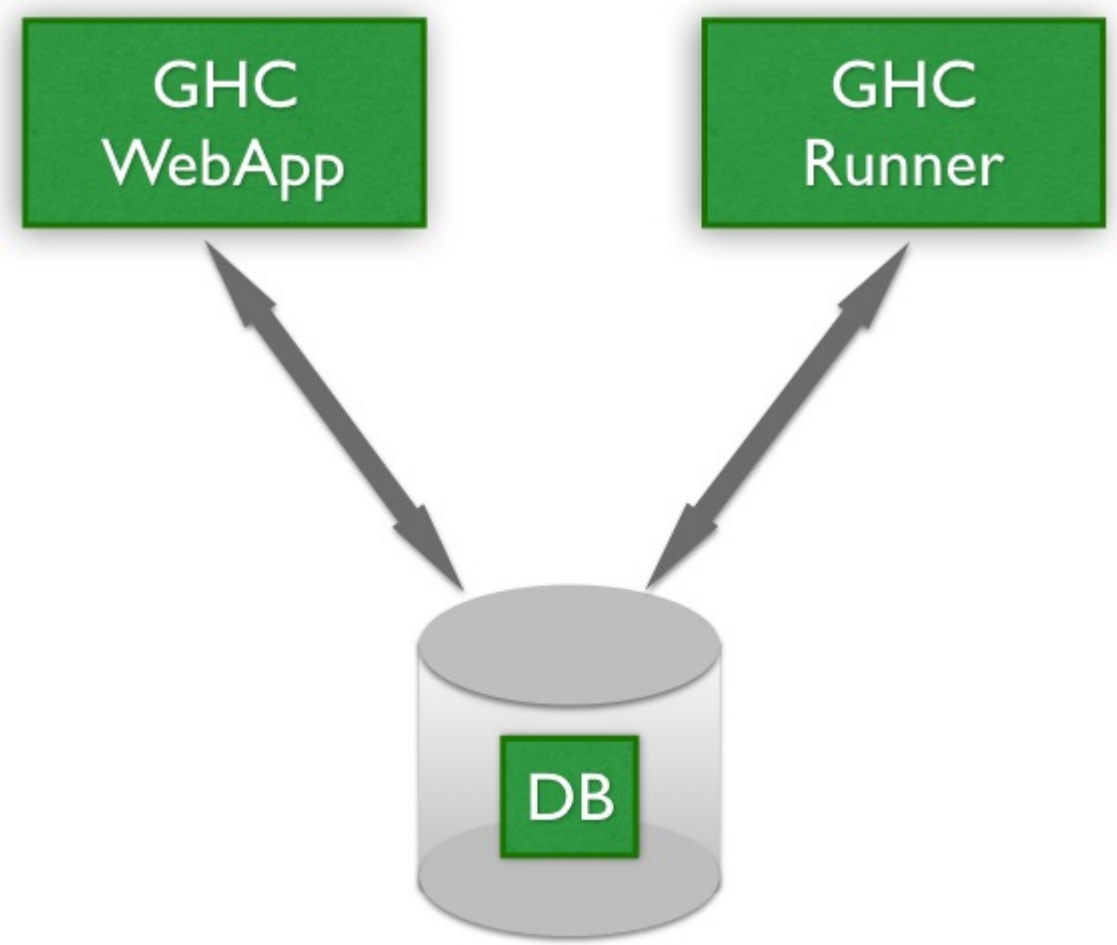


Fig. 15: *Figure - GHC Parts*

### 3.5.1 Core Concepts

GeoHealthCheck is built with the following concepts in mind:

- *Resource*: a single, unique endpoint, like an OGC WMS, FTP URL, or plain old web link. A GeoHealthCheck deployment typically monitors numerous *Resources*.
- *Run*: the execution and scoring of a test against a *Resource*. A *Resource* may have multiple *Runs*
- Each *User* owns one or more *Resources*
- Each *Resource* is tested, "probed", via one or more *Probes*
- Each *Probe* typically runs one or more requests on a *Resource* URL
- Each *Probe* invokes one or more *Checks* to determine *Run* result
- *Probes* and *Checks* are extensible *Plugins* via respective *Probe* and *Check* classes
- One or more *Tags* can be associated with a *Resource* to support grouping
- One or more *Recipient* can be associated with a *Resource*. Each *Recipient* describes:
  - communication channel
  - target identifier

### 3.5.2 Data Model

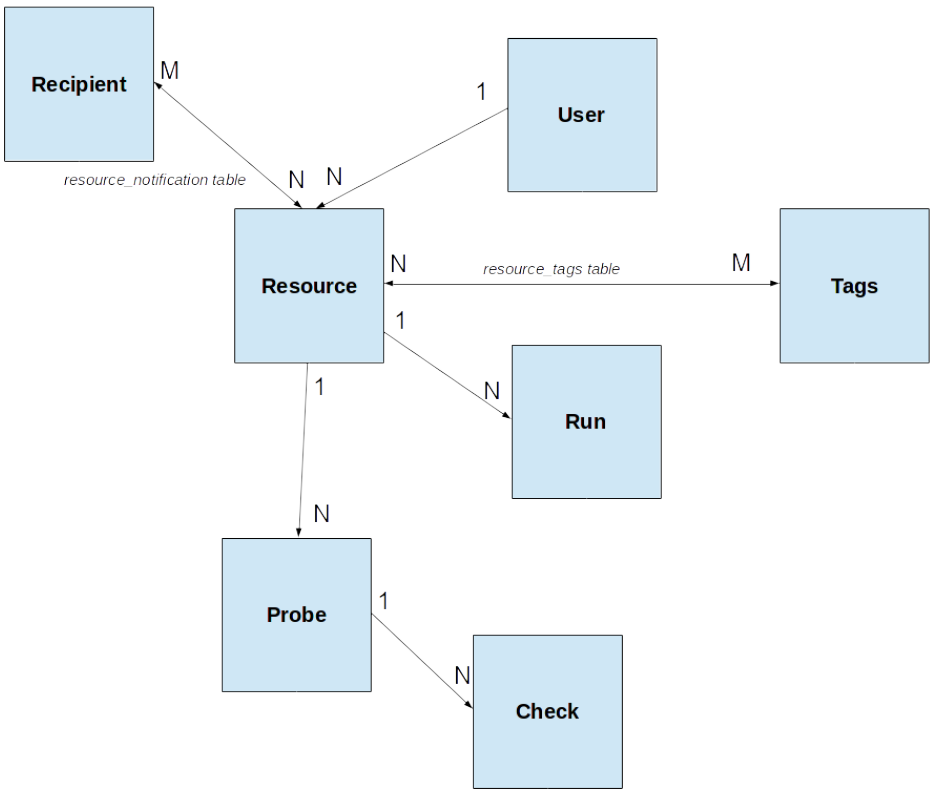


Fig. 16: *Figure - GHC Data Model*

### 3.5.3 GHC Webapp Design

The **GHC Webapp** is realized as a standard *Flask* web-application using *SQLAlchemy* for ORM database support. It is the user-visible part of GHC as it runs via the browser. Its main two functionalities are to allow users to:

- manage (create, update, delete) Resources, their attributes and their Probes and Checks, and
- view results and statistics of Resources (Dashboard function)

Deployment can be realized using the various standard Flask deployment methods: standalone, within a WSGI server etc.

As an option (via configuration, see above) the **GHC Runner** may run within the **GHC Webapp**. Note that in case that when the **GHC Webapp** runs as multiple processes and/or threads "Resource Locking" (see below) will prevent inconsistencies.

### 3.5.4 GHC Runner Design

The **GHC Runner** in its core is a job scheduler based on the Python library APScheduler. Each job scheduled is a healthcheck runner for a single *Resource* that runs all the *Probes* for that *Resource*. The run-frequency follows the per-Resource run frequency (since v0.5.0).

The GHC Runner is thus responsible for running the *Probes* for each *Resource*, storing the *Results* and doing notifications when needed.

The **GHC Runner** can run as a separate (Python) process, or within the **GHC WebApp** (see above). Separate processes is the preferred mode of running.

#### Job Runner Synchronization

As multiple instances of the job scheduler (i.e. APScheduler) may run in different processes and even threads within processes, the database is used to synchronize and assure only one job will run for a single *Resource*.

This is achieved by having one lock per *Resource* via the table *ResourceLock*. Only the process/thread that acquires its related *ResourceLock* record runs the job. As to avoid permanent "lockouts", each *ResourceLock* has a lifetime, namely the timespan until the next Run as configured for/per *Resource*. This gives all job runners a chance to obtain a lock once "time's up" for the *ResourceLock*.

Additional lock-liveliness is realized by using a unique **UUID per job runner**. Once the lock is obtained, the UUID-field of the lock record is set and committed to the DB. If we then try to obtain the lock again (by reading from DB) but the UUID is different this means another job runner instance did the same but was just before us. The lock-lifetime (see above) guards that a particular UUID keeps the lock forever, e.g. on sudden application shutdown.

To further increase liveliness, mainly to avoid all Jobs running at the same time when scheduled to run at the same frequency, each Job is started with a random time-offset on GHC Runner startup.

The locking mechanism described above is supported for SQLite, but it is strongly advised to use PostgreSQL in production deployments, also for better robustness and performance in general.

## 3.6 Plugins

GHC can be extended for Resource-specific healthchecks via Plugins. GHC already comes with a set of standard plugins that may suffice most installations. However, there is no limit to detailed healthchecks one may want to perform. Hence developers can extend or even replace the GHC standard Plugins with custom implementations.

Two Plugin types exist that can be extended: the *Probe* and *Check* class. In v0.7.0 also plugins for Resource Authentication, *ResourceAuth*, were added and in v0.9.0 the geocoder plugin was introduced.

### 3.6.1 Concepts

GHC versions after May 1, 2017 perform healthchecks exclusively via Plugins (see *Upgrade* how to upgrade from older versions). The basic concept is simple: each *Resource* (typically an OWS endpoint) has one or more *Probes*. During a GHC run (via *cron* or manually), GHC sequentually invokes the *Probes* for each *Resource* to determine the health (QoS) of the *Resource*.

A *Probe* typically implements a single request like a *WMS GetMap*. A *Probe* contains and applies one or more *Checks* (the other Plugin class). A *Check* implements typically a single check on the HTTP Response object of its parent *Probe*, for example if the HTTP response has no errors or if a *WMS GetMap* actually returns an image (content-type check). Each *Check* will supply a *CheckResult* to its parent *Probe*. The list of *CheckResults* will then ultimately determine the *ProbeResult*. The *Probe* will in turn supply the *ProbeResult* to its parent *ResourceResult*. The GHC healthchecker will then determine the final outcome of the *Run* (fail/success) for the *Resource*, adding the list of Probe/CheckResults to the historic Run-data in the DB. This data can later be used for reporting and determining which *Check(s)* were failing.

So in summary: a *Resource* has one or more *Probes*, each *Probe* one or more *Checks*. On a GHC run these together provide a *Result*.

Probes and Checks available to the GHC instance are configured in *config_site.py*, the GHC instance config file. Also configured there is the default *Probe* class to assign to a Resource-type when it is added. Assignment and configuration/parameterization of *Probes* and *Checks* is via de UI on the Resource-edit page and stored in the database (tables: *probe_vars* and *check_vars*). That way the GHC healthcheck runner can read (from the DB) the list of Probes/Checks and their config for each Resource.

### 3.6.2 Implementation

*Probes* and *Checks* plugins are implemented as Python classes derived from `GeoHealthCheck.probe.Probe` and `GeoHealthCheck.check.Check` respectively. These classes inherit from the GHC abstract base class `GeoHealthCheck.plugin.Plugin`. This class mainly provides default attributes (in capitals) and introspection methods needed for UI configuration. *Class-attributes* (in capitals) are the most important concept of GHC Plugins in general. These provide metadata for various GHC functions (internal, UI etc). General class-attributes that Plugin authors should provide for derived *Probes* or *Checks* are:

- *AUTHOR*: Plugin author or team.
- *NAME*: Short name of Plugin.
- *DESCRIPTION*: Longer description of Plugin.
- *PARAM_DEFS*: Plugin Parameter definitions (see next)

*PARAM_DEFS*, a Python *dict* defines the parameter definitions for the *Probe* or *Check* that a user can configure via the UI. Each parameter (name) is itself a *dict* entry key that with the following key/value pairs:

- *type*: the parameter type, value: 'string', 'stringlist' (comma-separated strings) or 'bbox' (lowerX, lowerY, upperX, upperY),
- *description*: description of the parameter,

- *default*: parameter default value,
- *required*: is parameter required?,
- *range*: range of possible parameter values (array of strings), results in UI dropdown selector

A *Probe* should supply these additional class-attributes:

- *RESOURCE_TYPE* : GHC Resource type this Probe applies to, e.g. *OGC:WMS*, *:* (any Resource Type), see *enums.py* for range
- *REQUEST_METHOD* : HTTP request method capitalized, 'GET' (default) or 'POST'.
- *REQUEST_HEADERS* : *dict* of optional HTTP request headers
- *REQUEST_TEMPLATE*: template in standard Python *str.format(*args)* to be substituted with actual parameters from *PARAM_DEFS*
- *CHECKS_AVAIL* : available Check (classes) for this Probe.

Note: *CHECKS_AVAIL* denotes all possible *Checks* that can be assigned, by default or via UI, to an instance of this *Probe*.

A *Check* has no additional class-attributes.

In many cases writing a *Probe* is a matter of just defining the above class-attributes. The GHC healthchecker `GeoHealthCheck.healthcheck.run_test_resource()` will call lifecycle methods of the *GeoHealthCheck.probe.Probe* base class, using the class-attributes and actualized parameters (stored in *probe_vars* table) as defined in *PARAM_DEFS* plus a list of the actual and parameterized Checks (stored in *check_vars* table) for its Probe instance.

More advanced *Probes* can override base-class methods of *Probe* in particular *GeoHealthCheck.probe.Probe.perform_request()*. In that case the Probe-author should add one or more *GeoHealthCheck.result.Result* objects to *self.result* via *self.result.add_result(result)*

Writing a *Check* class requires providing the Plugin class-attributes (see above) including optional *PARAM_DEFS*. The actual check is implemented by overriding the *Check* base class method *GeoHealthCheck.check.Check.perform()*, setting the check-result via `GeoHealthCheck.check.Check.set_result()`.

Finally your Probes and Checks need to be made available to your GHC instance via *config_site.py* and need to be found on the Python-PATH of your app.

The above may seem daunting at first. Examples below will hopefully make things clear as writing new *Probes* and *Checks* may sometimes be a matter of minutes!

*TODO: may need VERSION variable class-attr to support upgrades*

### 3.6.3 Examples

GHC includes Probes and Checks that on first setup are made available in *config_site.py*. By studying the the GHC standard Probes and Checks under the subdir *GeoHealthCheck/plugins*, Plugin-authors may get a feel how implementation can be effected.

There are broadly two ways to write a *Probe*:

- using a *REQUEST_** class-attributes, i.e. letting GHC do the Probe's HTTP requests and checks
- overriding *GeoHealthCheck.probe.Probe.perform_request()*: making your own requests

An example for each is provided, including the *Checks* used.

The simplest Probe is one that does:

- an HTTP GET on a *Resource* URL
- checks if the HTTP Response is not errored, i.e. a 404 or 500 status

- optionally checks if the HTTP Response (not) contains expected strings

Below is the implementation of the class *GeoHealthCheck.plugins.probe.http.HttpGet*:

```
from GeoHealthCheck.probe import Probe


class HttpGet(Probe):
    """
    Do HTTP GET Request, to poll/ping any Resource bare url.
    """

    NAME = 'HTTP GET Resource URL'
    DESCRIPTION = 'Simple HTTP GET on Resource URL'
    RESOURCE_TYPE = '*:*'
    REQUEST_METHOD = 'GET'

    CHECKS_AVAIL = {
        'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.ContainsStrings': {},
        'GeoHealthCheck.plugins.check.checks.NotContainsStrings': {},
        'GeoHealthCheck.plugins.check.checks.HttpHasContentType': {}
    }
    """Checks avail"""

```

Yes, this is the entire implementation of *GeoHealthCheck.plugins.probe.http.HttpGet*! Only class-attributes are needed:

- standard Plugin attributes: *AUTHOR* ('GHC Team' by default) *NAME*, *DESCRIPTION*
- *RESOURCE_TYPE* = *'*:*'* denotes that any Resource may use this Probe (UI lists this Probe under "Probes Available" for Resource)
- *REQUEST_METHOD* = *'GET'* : GHC should use the HTTP GET request method
- *CHECKS_AVAIL* : all Check classes that can be applied to this Probe (UI lists these under "Checks Available" for Probe)

By setting:

```
'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
   'default': True
},
```

that Check is automatically assigned to this Probe when created. The other Checks may be added and configured via the UI.

Next look at the Checks, the class *GeoHealthCheck.plugins.check.checks.HttpStatusNoError*:

```
import sys
from owslib.etree import etree
from GeoHealthCheck.util import CONFIG
from GeoHealthCheck.plugin import Plugin
from GeoHealthCheck.check import Check
```

(continues on next page)

(continued from previous page)

```
from html import escape


""" Contains basic Check classes for a Probe object."""


class HttpStatusNoError(Check):
    """
    Checks if HTTP status code is not in the 400- or 500-range.
    """

    NAME = 'HTTP status should not be errored'
    DESCRIPTION = 'Response should not contain a HTTP 400 or 500 range Error'

    def __init__(self):
        Check.__init__(self)

    def perform(self):
        """Default check: Resource should at least give no error"""
        status = self.probe.response.status_code
        overall_status = status // 100
        if overall_status in [4, 5]:
            self.set_result(False, 'HTTP Error status=%d' % status)


class HttpHasHeaderValue(Check):
    """
    Checks if header exists and has given header value.
    See http://docs.python-requests.org/en/master/user/quickstart
```

Also this class is quite simple: providing class-attributes *NAME*, *DESCRIPTION* and implementing the base-class method *GeoHealthCheck.check.Check.perform()*. Via *self.probe* a Check always has a reference to its parent Probe instance and the HTTP Response object via *self.probe.response*. The check itself is a test if the HTTP status code is in the 400 or 500-range. The *CheckResult* is implicitly created by setting: *self.set_result(False, 'HTTP Error status=%d' % status)* in case of errors. *self.set_result()* only needs to be called when a Check fails. By default the Result is succes (*True*).

According to this pattern more advanced Probes are implemented for *OWS GetCapabilities*, the most basic test for OWS-es like WMS and WFS. Below the implementation of the class *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps* and its derived classes for specific OWS-es:

```
from GeoHealthCheck.plugin import Plugin
from GeoHealthCheck.probe import Probe


class OwsGetCaps(Probe):
    """
    Fetch OWS capabilities doc
    """

    AUTHOR = 'GHC Team'
    NAME = 'OWS GetCapabilities'
    DESCRIPTION = 'Perform GetCapabilities Operation and check validity'
```

(continues on next page)

(continued from previous page)

```
    # Abstract Base Class for OGC OWS GetCaps Probes
    # Needs specification in subclasses
    # RESOURCE_TYPE = 'OGC:ABC'

    REQUEST_METHOD = 'GET'
    REQUEST_TEMPLATE = \
        '?SERVICE={service}&VERSION={version}&REQUEST=GetCapabilities'

    PARAM_DEFS = {
        'service': {
            'type': 'string',
            'description': 'The OWS service within resource endpoint',
            'default': None,
            'required': True
        },
        'version': {
            'type': 'string',
            'description': 'The OWS service version within resource endpoint',
            'default': None,
            'required': True,
            'range': None
        }
    }
    """Param defs, to be specified in subclasses"""

    CHECKS_AVAIL = {
        'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.XmlParse': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.ContainsStrings': {
            'set_params': {
                'strings': {
                    'name': 'Contains Title Element',
                    'value': ['Title>']
                }
            },
            'default': True
        },
    }
    """
    Checks avail for all specific Caps checks.
    Optionally override Check PARAM_DEFS using set_params
    e.g. with specific `value`.
    """
```

(continues on next page)

(continued from previous page)

```
class WmsGetCaps(OwsGetCaps):
    """Fetch WMS capabilities doc"""

    NAME = 'WMS GetCapabilities'
    RESOURCE_TYPE = 'OGC:WMS'

    PARAM_DEFS = Plugin.merge(OwsGetCaps.PARAM_DEFS, {

        'service': {
            'value': 'WMS'
        },
        'version': {
            'default': '1.3.0',
            'range': ['1.1.1', '1.3.0']
        }
    })
    """Param defs"""


class WfsGetCaps(OwsGetCaps):
    """WFS GetCapabilities Probe"""

    NAME = 'WFS GetCapabilities'
    RESOURCE_TYPE = 'OGC:WFS'

    def __init__(self):
        OwsGetCaps.__init__(self)

    PARAM_DEFS = Plugin.merge(OwsGetCaps.PARAM_DEFS, {
        'service': {
            'value': 'WFS'
        },
        'version': {
            'default': '1.1.0',
            'range': ['1.0.0', '1.1.0', '2.0.2']
        }
    })
    """Param defs"""


class WcsGetCaps(OwsGetCaps):
    """WCS GetCapabilities Probe"""

    NAME = 'WCS GetCapabilities'
```

More elaborate but still only class-attributes are used! Compared to *GeoHealthCheck.plugins.probe.http.HttpGet*, two additional class-attributes are used in *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps* :

- *REQUEST_TEMPLATE ='?SERVICE={service}&VERSION={version}&REQUEST=GetCapabilities'*
- *PARAM_DEFS* for the *REQUEST_TEMPLATE*

GHC will recognize a *REQUEST_TEMPLATE* (for GET or POST) and use *PARAM_DEFS* to substitute configured or

default values, here defined in subclasses. This string is then appended to the Resource URL.

Three *Checks* are available, all included by default. Also see the construct:

```
'GeoHealthCheck.plugins.check.checks.ContainsStrings': {
   'set_params': {
       'strings': {
           'name': 'Contains Title Element',
           'value': ['Title>']
       }
   },
   'default': True
},
```

This not only assigns this Check automatically on creation, but also provides it with parameters, in this case a *Capabilities* response document should always contain a <*Title*> XML element. The class *GeoHealthCheck.plugins.check.checks.ContainsStrings* checks if a response doc contains all of a list (array) of configured strings. So the full checklist on the response doc is:

- is it XML-parsable: *GeoHealthCheck.plugins.check.checks.XmlParse*
- does not contain an Exception: *GeoHealthCheck.plugins.check.checks.NotContainsOwsException*
- does it have a <*Title*> element: *GeoHealthCheck.plugins.check.checks.ContainsStrings*

These Checks are performed in that order. If any fails, the Probe Run is in error.

We can now look at classes derived from *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*, in particular *GeoHealthCheck.plugins.probe.owsgetcaps.WmsGetCaps* and *GeoHealthCheck.plugins.probe.owsgetcaps.WfsGetCaps*. These only need to set their *RESOURCE_TYPE* e.g. *OGC:WMS* and override/merge *PARAM_DEFS*. For example for WMS:

```
PARAM_DEFS = Plugin.merge(OwsGetCaps.PARAM_DEFS, {

  'service': {
      'value': 'WMS'
  },
  'version': {
      'default': '1.1.1',
      'range': ['1.1.1', '1.3.0']
  }
})
```

This sets a fixed *value* for *service*, later becoming *service=WMS* in the URL request string. For *version* it sets both a *range* of values a user can choose from, plus a default value *1.1.1*. *Plugin.merge* needs to be used to merge-in new values. Alternatively *PARAM_DEFS* can be completely redefined, but in this case we only need to make per-OWS specific settings.

Also new in this example is parameterization of Checks for the class *GeoHealthCheck.plugins.check.checks.ContainsStrings*. This is a generic HTTP response checker for a list of strings that each need to be present in the response. Alternatively *GeoHealthCheck.plugins.check.checks.NotContainsStrings* has the reverse test. Both are extremely useful and for example available to our first example *GeoHealthCheck.plugins.probe.http.HttpGet*. The concept of *PARAM_DEFS* is the same for Probes and Checks.

In fact a Probe for any REST API could be defined in the above matter. For example, later in the project a Probe was added for the SensorThings API (STA), a recent OGC-standard for managing Sensor data via a JSON REST API. See the listing below:

```python
from GeoHealthCheck.probe import Probe


class StaCaps(Probe):
    """Probe for SensorThings API main endpoint url"""

    NAME = 'STA Capabilities'
    DESCRIPTION = 'Perform STA Capabilities Operation and check validity'
    RESOURCE_TYPE = 'OGC:STA'

    REQUEST_METHOD = 'GET'

    def __init__(self):
        Probe.__init__(self)

    CHECKS_AVAIL = {
        'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.JsonParse': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.ContainsStrings': {
            'default': True,
            'set_params': {
                'strings': {
                    'name': 'Must contain STA Entity names',
                    'value': ['Things', 'Datastreams', 'Observations',
                              'FeaturesOfInterest', 'Locations']
                }
            }
        },
    }
    """
    Checks avail for all specific Caps checks.
    Optionally override Check.PARAM_DEFS using set_params
    e.g. with specific `value` or even `name`.
    """


class StaGetEntities(Probe):
    """Fetch STA entities of type and check result"""

    NAME = 'STA GetEntities'
    DESCRIPTION = 'Fetch all STA Entities of given type'
    RESOURCE_TYPE = 'OGC:STA'

    REQUEST_METHOD = 'GET'

    # e.g. http://52.26.56.239:8080/OGCSensorThings/v1.0/Things
    REQUEST_TEMPLATE = '/{entities}'

    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        Probe.__init__(self)

    PARAM_DEFS = {
        'entities': {
            'type': 'string',
            'description': 'The STA Entity collection type',
            'default': 'Things',
            'required': True,
            'range': ['Things', 'DataStreams', 'Observations',
                      'Locations', 'Sensors', 'FeaturesOfInterest',
                      'ObservedProperties', 'HistoricalLocations']
        }
    }
    """Param defs"""

    CHECKS_AVAIL = {
        'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
            'default': True
        },
        'GeoHealthCheck.plugins.check.checks.JsonParse': {
            'default': True
        }
    }
    """Check for STA Get entity Collection"""
```

Up to now all Probes were defined using and overriding class-attributes. Next is a more elaborate example where the Probe overrides the Probe baseclass method *GeoHealthCheck.probe.Probe.perform_request()*. The example is more of a showcase: *GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown* literally drills-down through WMS-entities: starting with the *GetCapabilities* doc it fetches the list of *Layers* and does a *GetMap* on random layers etc. It uses *OWSLib.WebMapService*.

We show the first 70 lines here.

```
import random

from GeoHealthCheck.probe import Probe
from GeoHealthCheck.result import Result
from owslib.wms import WebMapService


class WmsDrilldown(Probe):
    """
    Probe for WMS endpoint "drilldown": starting
    with GetCapabilities doc: get Layers and do
    GetMap on them etc. Using OWSLib.WebMapService.

    TODO: needs finalization.
    """

    NAME = 'WMS Drilldown'
    DESCRIPTION = 'Traverses a WMS endpoint by drilling down from Capabilities'
    RESOURCE_TYPE = 'OGC:WMS'

```

(continues on next page)

(continued from previous page)

```
    REQUEST_METHOD = 'GET'

    PARAM_DEFS = {
        'drilldown_level': {
            'type': 'string',
            'description': 'How heavy the drilldown should be.',
            'default': 'minor',
            'required': True,
            'range': ['minor', 'moderate', 'full']
        }
    }
    """Param defs"""

    def __init__(self):
        Probe.__init__(self)

    def perform_request(self):
        """
        Perform the drilldown.
        See https://github.com/geopython/OWSLib/blob/
        master/tests/doctests/wms_GeoServerCapabilities.txt
        """
        wms = None

        # 1. Test capabilities doc, parses
        result = Result(True, 'Test Capabilities')
        result.start()
        try:
            wms = WebMapService(self._resource.url,
                                headers=self.get_request_headers())
            title = wms.identification.title
            self.log('response: title=%s' % title)
        except Exception as err:
            result.set(False, str(err))

        result.stop()
        self.result.add_result(result)

        # 2. Test layers
        # TODO: use parameters to work on less/more drilling
        # "full" could be all layers.
        result = Result(True, 'Test Layers')
        result.start()
        try:
            # Pick a random layer
            layer_name = random.sample(wms.contents.keys(), 1)[0]
            layer = wms[layer_name]

            # TODO Only use EPSG:4326, later random CRS
            if 'EPSG:4326' in layer.crsOptions \
```

This shows that any kind of simple or elaborate healthchecks can be implemented using single or multiple HTTP requests. As long as Result objects are set via *self.result.add_result(result)*. It is optional to also define *Checks* in this

case. In the example *GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown* example no Checks are used.

One can imagine custom Probes for many use-cases:

- drill-downs for OWS-es
- checking both the service and its metadata (CSW links in Capabilities doc e.g.)
- gaps in timeseries data (SOS, STA)
- even checking resources like a remote GHC itself!

Writing custom Probes is only limited by your imagination!

### 3.6.4 Configuration

Plugins available to a GHC installation are configured via *config_main.py* and overridden in *config_site.py*. By default all built-in Plugins are available.

- **GHC_PLUGINS**: *list* of built-in/core Plugin classes and/or modules available on installation
- **GHC_PROBE_DEFAULTS**: Default *Probe* class to assign on "add" per Resource-type
- **GHC_USER_PLUGINS**: *list* of your Plugin classes and/or modules available on installation

To add your Plugins, you need to configure **GHC_USER_PLUGINS**. In most cases you don't need to bother with **GHC_PLUGINS** and **GHC_PROBE_DEFAULTS**.

See an example for both below from *config_main.py* for **GHC_PLUGINS** and **GHC_PROBE_DEFAULTS**:

```
GHC_PLUGINS = [
    # Probes
    'GeoHealthCheck.plugins.probe.owsgetcaps',
    'GeoHealthCheck.plugins.probe.wms',
    'GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox',
    'GeoHealthCheck.plugins.probe.tms',
    'GeoHealthCheck.plugins.probe.http',
    'GeoHealthCheck.plugins.probe.sta',
    'GeoHealthCheck.plugins.probe.wmsdrilldown',
    'GeoHealthCheck.plugins.probe.ogcfeat',

    # Checks
    'GeoHealthCheck.plugins.check.checks',
]

# Default Probe to assign on "add" per Resource-type
GHC_PROBE_DEFAULTS = {
    'OGC:WMS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.WmsGetCaps'
    },
    'OGC:WMTS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.WmtsGetCaps'
    },
    'OSGeo:TMS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.tms.TmsCaps'
    },
    'OGC:WFS': {
```

(continues on next page)

(continued from previous page)

```
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.WfsGetCaps'
    },
    'OGC:WCS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.WcsGetCaps'
    },
    'OGC:WPS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.WpsGetCaps'
    },
    'OGC:CSW': {
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.CswGetCaps'
    },
    'OGC:SOS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.owsgetcaps.SosGetCaps'
    },
    'OGC:STA': {
        'probe_class': 'GeoHealthCheck.plugins.probe.sta.StaCaps'
    },
    'OGCFeat': {
        'probe_class': 'GeoHealthCheck.plugins.probe.ogcfeat.OGCFeatDrilldown'
    },
    'ESRI:FS': {
        'probe_class': 'GeoHealthCheck.plugins.probe.esrifs.ESRIFSDrilldown'
    },
    'urn:geoss:waf': {
        'probe_class': 'GeoHealthCheck.plugins.probe.http.HttpGet'
    },
    'WWW:LINK': {
        'probe_class': 'GeoHealthCheck.plugins.probe.http.HttpGet'
    },
    'FTP': {
        'probe_class': None
    }
}
```

To add your User Plugins these steps are needed:

- place your Plugin in any directory
- specify your Plugin in *config_site.py* in **GHC_USER_PLUGINS** var
- your Plugin module needs to be available in the *PYTHONPATH* of the GHC app

Let's say your Plugin is in file */plugins/ext/myplugin.py*. Example *config_site.py*

```
GHC_USER_PLUGINS='ext.myplugin'
```

Then you need to add the path */plugins* to the *PYTHONPATH* such that your Plugin is found.

### 3.6.5 User Plugins via Docker

The easiest way to add your Plugins (and running GHC in general!) is by using GHC Docker. See more info in the GHC Docker Plugins README.

### 3.6.6 Plugin API Docs

For GHC extension via Plugins the following classes apply.

Most Plugins have *PARAM_DEFS* parameter definitions. These are variables that should be filled in by the user in the GUI unless a fixed *value* applies.

#### Plugins - Base Classes

These are the base classes for GHC Plugins. Developers will mainly extend *Probe* and *Check*.

**class** GeoHealthCheck.plugin.**Plugin**

Bases: `object`

Abstract Base class for all GHC Plugins. Derived classes should fill in all class variables that are UPPER_CASE, unless they ar fine with default-values from superclass(es).

**AUTHOR = 'GHC Team'**
Plugin author or team.

**DESCRIPTION = 'Description missing in DESCRIPTION class var'**
Longer description of Plugin. TODO: optional i18n e.g. DESCRIPTION_de_DE ?

**NAME = 'Name missing in NAME class var'**
Short name of Plugin. TODO: i18n e.g. NAME_nl_NL ?

**PARAM_DEFS = {}**
Plugin Parameter definitions.

**static copy**(*obj*)
Deep copy of usually *dict* object.

**get_default_parameter_values**()
Get all default parameter values

**get_param**(*param_name*)
Get actual parameter value. *param_name* should be defined in *PARAM_DEFS*.

**get_param_defs**()
Get all PARAM_DEFS as dict.

**get_plugin_vars**()
Get all (uppercase) class variables of a class as a dict

**static get_plugins**(*baseclass='GeoHealthCheck.plugin.Plugin'*, *filters=None*)
Class method to get list of Plugins of particular baseclass (optional), default is all Plugins. filters is a list of tuples to filter out Plugins with class var values: (class var, value), e.g. *filters=[('RESOURCE_TYPE', 'OGC:*'), ('RESOURCE_TYPE', 'OGC:WMS')]*.

**get_var_names**()
Get all Plugin variable names as a dict

**static merge**(*dict1*, *dict2*)
Recursive merge of two *dict*, mainly used for PARAM_DEFS, CHECKS_AVAIL overriding. :param dict1: base dict :param dict2: dict to merge into dict1 :return: deep copy of dict2 merged into dict1

class GeoHealthCheck.probe.**Probe**

Bases: plugin.Plugin

Base class for specific implementations to run a Probe with Checks. Most Probes can be implemented using REQUEST_TEMPLATES parameterized via actualized PARAM_DEFS but specialized Probes may implement their own Requests and Checks, for example by "drilling down" through OWS services on an OGC OWS endpoint starting at the Capabilities level or for specific WWW:LINK-based REST APIs.

**CHECKS_AVAIL = {}**

Available Check (classes) for this Probe in *dict* format. Key is a Check class (string), values are optional (default *{}*). In the (constant) value 'parameters' and other attributes for Check.PARAM_DEFS can be specified, including *default* if this Check should be added to Probe on creation.

**METADATA_CACHE = {}**

Cache for metadata, like capabilities documents or OWSLib Service instances. Saves doing multiple requests/responses. In particular for endpoints with 50+ Layers.

**PARAM_DEFS = {}**

Parameter definitions mostly for *REQUEST_TEMPLATE* but potential other uses in specific Probe implementations. Format is *dict* where each key is a parameter name and the value a *dict* of: *type*, *description*, *required*, *default*, *range* (value range) and optional *value* item. If *value* specified, this value becomes fixed (non-editable) unless overridden in subclass.

**REQUEST_HEADERS = {}**

*dict* of optional HTTP request headers.

**REQUEST_METHOD = 'GET'**

HTTP request method capitalized, GET (default) or POST.

**REQUEST_TEMPLATE = ''**

Template in standard Python *str.format(*args)*. The variables like {service} and {version} within a template are filled from actual values for parameters defined in PARAM_DEFS and substituted from values or constant values specified by user in GUI and stored in DB.

**RESOURCE_TYPE = 'Not Applicable'**

Type of GHC Resource e.g. 'OGC:WMS', default not applicable.

**STANDARD_REQUEST_HEADERS = {'Accept-Encoding': 'deflate, gzip;q=1.0, *;q=0.5', 'User-Agent': 'GeoHealthCheck 0.9.0 (https://geohealthcheck.org)'}**

*dict* of HTTP headers to add to each HTTP request.

**after_request**()

After running actual request to service

**before_request**()

Before running actual request to service

**calc_result**()

Calculate overall result from the Result object

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**get_metadata**(*resource*, *version='any'*)

Get metadata, specific per Resource type. :param resource: :param version: :return: Metadata object

**get_metadata_cached**(*resource*, *version='any'*)

Get metadata, specific per Resource type, get from cache if cached. :param resource: :param version: :return: Metadata object

**get_plugin_vars**()
Get all (uppercase) class variables of a class as a dict

**get_var_names**()
Get all Plugin variable names as a dict

**init**(*resource*, *probe_vars*)
Probe contains the actual Probe parameters (from Models/DB) for requests and a list of response Checks with their functions and parameters :param resource: :param probe_vars: :return: None

**perform_get_request**(*url*)
Perform actual HTTP GET request to service

**perform_post_request**(*url_base*, *request_string*)
Perform actual HTTP POST request to service

**perform_request**()
Perform actual request to service

**static run**(*resource*, *probe_vars*)
Class method to create and run a single Probe instance. Follows strict sequence of method calls. Each method can be overridden in subclass.

**run_checks**()
Do the checks on the response from request

**run_request**()
Run actual request to service

**class** GeoHealthCheck.check.**Check**
Bases: plugin.Plugin

Base class for specific Plugin implementations to perform a check on results from a Probe.

**init**(*probe*, *check_vars*)
Initialize Checker with parent Probe and parameters dict. :return:

**perform**()
Perform this Check's specific check. TODO: return Result object. :return:

**class** GeoHealthCheck.resourceauth.**ResourceAuth**
Bases: plugin.Plugin

Base class for specific Plugin implementations to perform authentication on a Resource. Subclasses provide specific auth methods like Basic Auth, Bearer Token etc.

**static decode**(*encoded*)
Decode/decrypt encrypted string into auth dict. :return: encoded auth dict

**encode**()
Encode/encrypt auth dict structure. :return: encoded string

**static get_auth_defs**()
Get available ResourceAuth definitions. :return: dict keyed by NAME with object instance values

**get_auth_header**()
Get encoded authorization header value from config data. Authorization scheme-specific. :return: None or dict with http auth header

**init**(*auth_dict=None*)
Initialize ResourceAuth with related Resource and auth dict. :return:

*Results* are helper-classes whose intances are generated by both *Probe* and *Check* classes. They form the ultimate outcome when running a *Probe*. A *ResourceResult* contains *ProbeResults*, the latter contains *CheckResults*.

class GeoHealthCheck.result.**CheckResult**(*check*, *check_vars*, *success=True*, *message='OK'*)
Bases: *GeoHealthCheck.result.Result*

Holds result data from a single Check.

class GeoHealthCheck.result.**ProbeResult**(*probe*, *probe_vars*)
Bases: *GeoHealthCheck.result.Result*

Holds result data from a single Probe: one Probe, N Checks.

class GeoHealthCheck.result.**ResourceResult**(*resource*)
Bases: *GeoHealthCheck.result.Result*

Holds result data from a single Resource: one Resource, N Probe(Results). Provides Run data.

class GeoHealthCheck.result.**Result**(*success=True*, *message='OK'*)
Bases: object

Base class for results for Resource or Probe.

### Plugins - Probes

*Probes* apply to a single *Resource* instance. They are responsible for running requests against the Resource URL endpoint. Most *Probes* are implemented mainly via configuring class variables in particular *PARAM_DEFS* and *CHECKS_AVAIL*, but one is free to override any of the *Probe* baseclass methods.

class GeoHealthCheck.plugins.probe.http.**HttpGet**
Bases: *GeoHealthCheck.probe.Probe*

Do HTTP GET Request, to poll/ping any Resource bare url.

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {}, 'GeoHealthCheck.plugins.check.checks.HttpHasContentType': {}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.NotContainsStrings': {}}**
Checks avail

class GeoHealthCheck.plugins.probe.http.**HttpGetQuery**
Bases: *GeoHealthCheck.plugins.probe.http.HttpGet*

Do HTTP GET Request, to poll/ping any Resource bare url with query string.

**PARAM_DEFS = {'query': {'default': None, 'description': 'The query string to add to request (without ?)', 'required': True, 'type': 'string'}}**
Param defs

class GeoHealthCheck.plugins.probe.http.**HttpPost**
Bases: *GeoHealthCheck.plugins.probe.http.HttpGet*

Do HTTP POST Request, to send POST request to Resource bare url with POST body.

**PARAM_DEFS = {'body': {'default': None, 'description': 'The post body to send', 'required': True, 'type': 'string'}, 'content_type': {'default': 'text/xml;charset=UTF-8', 'description': 'The post content type to send', 'required': True, 'type': 'string'}}**
Param defs

**get_request_headers**()
Overridden from Probe: construct request_headers via parameter substitution from content_type Parameter.

class GeoHealthCheck.plugins.probe.owsgetcaps.**CswGetCaps**
Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

CSW GetCapabilities Probe

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'CSW'}, 'version': {'default': '2.0.2', 'description': 'The OWS service version within resource endpoint', 'range': ['2.0.2'], 'required': True, 'type': 'string'}}**

Param defs

**class** GeoHealthCheck.plugins.probe.owsgetcaps.**OwsGetCaps**

Bases: *GeoHealthCheck.probe.Probe*

Fetch OWS capabilities doc

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {'default': True, 'set_params': {'strings': {'name': 'Contains Title Element', 'value': ['Title>']}}}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {'default': True}, 'GeoHealthCheck.plugins.check.checks.XmlParse': {'default': True}}**

Checks avail for all specific Caps checks. Optionally override Check PARAM_DEFS using set_params e.g. with specific *value*.

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string'}, 'version': {'default': None, 'description': 'The OWS service version within resource endpoint', 'range': None, 'required': True, 'type': 'string'}}**

Param defs, to be specified in subclasses

**class** GeoHealthCheck.plugins.probe.owsgetcaps.**SosGetCaps**

Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

SOS GetCapabilities Probe

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'SOS'}, 'version': {'default': '1.0.0', 'description': 'The OWS service version within resource endpoint', 'range': ['1.0.0', '2.0.0'], 'required': True, 'type': 'string'}}**

Param defs

**class** GeoHealthCheck.plugins.probe.owsgetcaps.**WcsGetCaps**

Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

WCS GetCapabilities Probe

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'WCS'}, 'version': {'default': '1.1.0', 'description': 'The OWS service version within resource endpoint', 'range': ['1.1.0', '1.1.1', '2.0.1'], 'required': True, 'type': 'string'}}**

Param defs

**class** GeoHealthCheck.plugins.probe.owsgetcaps.**WfsGetCaps**

Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

WFS GetCapabilities Probe

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'WFS'}, 'version': {'default': '1.1.0', 'description': 'The OWS service version within resource endpoint', 'range': ['1.0.0', '1.1.0', '2.0.2'], 'required': True, 'type': 'string'}}**

Param defs

class GeoHealthCheck.plugins.probe.owsgetcaps.**WmsGetCaps**
Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

Fetch WMS capabilities doc

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'WMS'}, 'version': {'default': '1.3.0', 'description': 'The OWS service version within resource endpoint', 'range': ['1.1.1', '1.3.0'], 'required': True, 'type': 'string'}}**
Param defs

class GeoHealthCheck.plugins.probe.owsgetcaps.**WmtsGetCaps**
Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

WMTS GetCapabilities Probe

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'WMTS'}, 'version': {'default': '1.0.0', 'description': 'The OWS service version within resource endpoint', 'range': ['1.0.0'], 'required': True, 'type': 'string'}}**
Param defs

**after_request**()
After running actual request to service

**before_request**()
Before running actual request to service

class GeoHealthCheck.plugins.probe.owsgetcaps.**WpsGetCaps**
Bases: *GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps*

WPS GetCapabilities Probe

**PARAM_DEFS = {'service': {'default': None, 'description': 'The OWS service within resource endpoint', 'required': True, 'type': 'string', 'value': 'WPS'}, 'version': {'default': '1.0.0', 'description': 'The OWS service version within resource endpoint', 'range': ['1.0.0', '2.0.0'], 'required': True, 'type': 'string'}}**
Param defs

class GeoHealthCheck.plugins.probe.wms.**WmsGetMapV1**
Bases: *GeoHealthCheck.probe.Probe*

Get WMS map image using the OGC WMS GetMap v1.1.1 Operation for single Layer.

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpHasImageContentType': {'default': True}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {'default': True}}**
Checks for WMS GetMap Response available. Optionally override Check PARAM_DEFS using set_params e.g. with specific *value* or even *name*.

**PARAM_DEFS = {'bbox': {'default': ['-180', '-90', '180', '90'], 'description': 'The WMS bounding box', 'range': None, 'required': True, 'type': 'bbox'}, 'exceptions': {'default': 'application/vnd.ogc.se_xml', 'description': 'The Exception format to use', 'range': None, 'required': True, 'type': 'string'}, 'format': {'default': 'image/png', 'description': 'The image format', 'range': None, 'required': True, 'type': 'string'}, 'height': {'default': '256', 'description': 'The image height', 'required': True, 'type': 'string'}, 'layers': {'default': [], 'description': 'The WMS Layer, select one', 'range': None, 'required': True, 'type': 'stringlist'}, 'srs': {'default': 'EPSG:4326', 'description': 'The SRS as EPSG: code', 'range': None, 'required': True, 'type': 'string'}, 'styles': {'default': None, 'description': 'The Styles to apply', 'required': False, 'type': 'string'}, 'width': {'default': '256', 'description': 'The image width', 'required': True, 'type': 'string'}}**

Param defs

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**get_metadata**(*resource*, *version='1.1.1'*)

Get metadata, specific per Resource type. :param resource: :param version: :return: Metadata object

**class** GeoHealthCheck.plugins.probe.wms.**WmsGetMapV1All**

Bases: *GeoHealthCheck.plugins.probe.wms.WmsGetMapV1*

Get WMS map image for each Layer using the WMS GetMap operation.

**before_request**()

Before request to service, overridden from base class

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**perform_request**()

Perform actual request to service, overridden from base class

**class** GeoHealthCheck.plugins.probe.wmsdrilldown.**WmsDrilldown**

Bases: *GeoHealthCheck.probe.Probe*

Probe for WMS endpoint "drilldown": starting with GetCapabilities doc: get Layers and do GetMap on them etc. Using OWSLib.WebMapService.

TODO: needs finalization.

**PARAM_DEFS = {'drilldown_level': {'default': 'minor', 'description': 'How heavy the drilldown should be.', 'range': ['minor', 'moderate', 'full'], 'required': True, 'type': 'string'}}**

Param defs

**perform_request**()

Perform the drilldown. See https://github.com/geopython/OWSLib/blob/master/tests/doctests/wms_GeoServerCapabilities.txt

**class** GeoHealthCheck.plugins.probe.tms.**TmsCaps**

Bases: *GeoHealthCheck.probe.Probe*

Probe for TMS main endpoint url

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {'default': True, 'set_params': {'strings': {'name': 'Must contain TileMap Element', 'value': ['TileMap']}}}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.XmlParse': {'default': True}}**

Checks avail for all specific Caps checks. Optionally override Check.PARAM_DEFS using set_params e.g. with specific *value* or even *name*.

**class** GeoHealthCheck.plugins.probe.tms.**TmsGetTile**

Bases: *GeoHealthCheck.probe.Probe*

Fetch TMS tile and check result

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpHasImageContentType': {'default': True}}**

Check for TMS GetTile

**PARAM_DEFS = {'extension': {'default': 'png', 'description': 'The tile image extension', 'range': None, 'required': True, 'type': 'string'}, 'layer': {'default': None, 'description': 'The TMS Layer within resource endpoint', 'range': None, 'required': True, 'type': 'string'}, 'x': {'default': '0', 'description': 'The tile x offset', 'range': None, 'required': True, 'type': 'string'}, 'y': {'default': '0', 'description': 'The tile y offset', 'range': None, 'required': True, 'type': 'string'}, 'zoom': {'default': '0', 'description': 'The tile pyramid zoomlevel', 'range': None, 'required': True, 'type': 'string'}}**

Param defs

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**get_metadata**(*resource*, *version='1.0.0'*)

Get metadata, specific per Resource type. :param resource: :param version: :return: Metadata object

**class** GeoHealthCheck.plugins.probe.tms.**TmsGetTileAll**

Bases: *GeoHealthCheck.plugins.probe.tms.TmsGetTile*

Get TMS map image for each Layer using the TMS GetTile operation.

**before_request**()

Before request to service, overridden from base class

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**perform_request**()

Perform actual request to service, overridden from base class

**class** GeoHealthCheck.plugins.probe.wmts.**WmtsGetTile**

Bases: *GeoHealthCheck.probe.Probe*

Get WMTS map tile for specific layers. There are 2 possible request templates to support both KVP and REST

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpHasImageContentType': {'default': True}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {'default': True}}**

Checks for WMTS GetTile Response available. Optionally override Check PARAM_DEFS using set_params e.g. with specific *value* or even *name*.

PARAM_DEFS = {'exceptions': {'description': 'The Exception format to use', 'type': 'string', 'value': 'application/vnd.ogc.se_xml'}, 'format': {'default': 'sample', 'description': 'The image format', 'range': ['sample'], 'type': 'string'}, 'kvprest': {'description': 'Request the endpoint through KVP or REST', 'range': [], 'required': True, 'type': 'string'}, 'latitude_4326': {'description': 'latitude of tile to request', 'type': 'string'}, 'layers': {'default': [], 'description': 'The WMTS Layer, select one', 'range': None, 'required': True, 'type': 'stringlist'}, 'longitude_4326': {'description': 'longitude of tile to request', 'type': 'string'}, 'style': {'description': 'The Styles to apply', 'type': 'string', 'value': 'default'}, 'tilematrix': {'default': 'sample', 'description': 'Zoom level index', 'range': ['all', 'sample'], 'type': 'string'}, 'tilematrixset': {'default': 'all', 'description': 'Projection with its own set of zoom level indices', 'range': ['all', 'sample'], 'type': 'string'}}

Param defs

actual_request()

Perform actual request to service

before_request()

Before request to service, overridden from base class

calculate_center_tile(*center_coord*, *tilematrix*, *crs*)

Determine center tile row and column indexes based on topleft coordinate, scale, center coordinate and tilewidth/height

check_capabilities(*url*)

Check for exception in GetCapabilities response

expand_params(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

get_metadata(*resource*, *version='1.0.0'*)

Get metadata, specific per Resource type. :param resource: :param version: :return: Metadata object

perform_request()

Perform actual request to service, overridden from base class

test_kvp_rest()

Make requests on some variations of the url to test if KVP and/or REST is possible.

class GeoHealthCheck.plugins.probe.wmts.**WmtsGetTileAll**

Bases: *GeoHealthCheck.plugins.probe.wmts.WmtsGetTile*

Get WMTS GetTile for all layers.

PARAM_DEFS = {'exceptions': {'description': 'The Exception format to use', 'type': 'string', 'value': 'application/vnd.ogc.se_xml'}, 'format': {'default': 'sample', 'description': 'The image format', 'range': ['sample'], 'type': 'string'}, 'kvprest': {'description': 'Request the endpoint through KVP or REST', 'range': [], 'required': True, 'type': 'string'}, 'latitude_4326': {'description': 'latitude of tile to request', 'type': 'string'}, 'layers': {'default': [], 'description': 'The WMTS Layer, select one', 'range': None, 'required': True, 'type': 'stringlist'}, 'longitude_4326': {'description': 'longitude of tile to request', 'type': 'string'}, 'style': {'description': 'The Styles to apply', 'type': 'string', 'value': 'default'}, 'tilematrix': {'default': 'sample', 'description': 'Zoom level index', 'range': ['all', 'sample'], 'type': 'string'}, 'tilematrixset': {'default': 'all', 'description': 'Projection with its own set of zoom level indices', 'range': ['all', 'sample'], 'type': 'string'}}

Param defs

**before_request**()
Before request to service, overridden from base class

**expand_params**(*resource*)
Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**class** GeoHealthCheck.plugins.probe.sta.**StaCaps**
Bases: *GeoHealthCheck.probe.Probe*

Probe for SensorThings API main endpoint url

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {'default': True, 'set_params': {'strings': {'name': 'Must contain STA Entity names', 'value': ['Things', 'Datastreams', 'Observations', 'FeaturesOfInterest', 'Locations']}}}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.JsonParse': {'default': True}}**
Checks avail for all specific Caps checks. Optionally override Check.PARAM_DEFS using set_params e.g. with specific *value* or even *name*.

**class** GeoHealthCheck.plugins.probe.sta.**StaGetEntities**
Bases: *GeoHealthCheck.probe.Probe*

Fetch STA entities of type and check result

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.JsonParse': {'default': True}}**
Check for STA Get entity Collection

**PARAM_DEFS = {'entities': {'default': 'Things', 'description': 'The STA Entity collection type', 'range': ['Things', 'DataStreams', 'Observations', 'Locations', 'Sensors', 'FeaturesOfInterest', 'ObservedProperties', 'HistoricalLocations'], 'required': True, 'type': 'string'}}**
Param defs

**class** GeoHealthCheck.plugins.probe.wfs.**WfsGetFeatureBbox**
Bases: *GeoHealthCheck.probe.Probe*

do WFS GetFeature in BBOX

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {'default': True, 'set_params': {'strings': {'description': '\n Has FeatureCollection element in response doc\n ', 'name': 'Must contain FeatureCollection Element', 'value': ['FeatureCollection']}}}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {'default': True}, 'GeoHealthCheck.plugins.check.checks.XmlParse': {'default': True}}**
Checks for WFS GetFeature Response available. Optionally override Check PARAM_DEFS using set_params e.g. with specific *value* or even *name*.

**PARAM_DEFS = {'bbox': {'default': ['-180', '-90', '180', '90'], 'description': 'The tile image extension', 'range': None, 'required': True, 'type': 'bbox'}, 'geom_property_name': {'default': None, 'description': 'Name of the geometry property within FeatureType', 'range': None, 'required': True, 'type': 'string', 'value': 'Not Required'}, 'srs': {'default': 'EPSG:4326', 'description': 'The SRS as EPSG: code', 'range': None, 'required': True, 'type': 'string'}, 'type_name': {'default': None, 'description': 'The WFS FeatureType name', 'range': None, 'required': True, 'type': 'string'}, 'type_ns_prefix': {'default': None, 'description': 'The WFS FeatureType namespace prefix', 'range': None, 'required': True, 'type': 'string'}, 'type_ns_uri': {'default': '0', 'description': 'The WFS FeatureType namespace URI', 'range': None, 'required': True, 'type': 'string'}}**

Param defs

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**get_metadata**(*resource*, *version='1.1.0'*)

Get metadata, specific per Resource type. :param resource: :param version: :return: Metadata object

**class** GeoHealthCheck.plugins.probe.wfs.**WfsGetFeatureBboxAll**

Bases: *GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox*

Do WFS GetFeature for each FeatureType in WFS.

**before_request**()

Before request to service, overridden from base class

**expand_params**(*resource*)

Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**perform_request**()

Perform actual request to service, overridden from base class

**class** GeoHealthCheck.plugins.probe.wcs.**WcsGetCoverage**

Bases: *GeoHealthCheck.probe.Probe*

Get WCS coverage image using the OGC WCS GetCoverage v2.0.1 Operation.

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpHasImageContentType': {'default': True}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {'default': True}}**

Checks for WCS GetCoverage Response available. Optionally override Check PARAM_DEFS using set_params e.g. with specific *value* or even *name*.

**PARAM_DEFS = {'format': {'default': [], 'description': 'Image outputformat', 'range': None, 'required': True, 'type': 'string'}, 'height': {'default': '10', 'description': 'The image height', 'required': True, 'type': 'string'}, 'layers': {'default': [], 'description': 'The WCS Layer ID, select one', 'range': None, 'required': True, 'type': 'stringlist'}, 'subset': {'default': ['-180', '-90', '180', '90'], 'description': 'The WCS subset of x and y axis', 'range': None, 'required': True, 'type': 'bbox'}, 'subsetting_crs': {'default': '', 'description': 'The crs of SUBSET and also OUTPUTCRS', 'range': None, 'required': True, 'type': 'string'}, 'width': {'default': '10', 'description': 'The image width', 'required': True, 'type': 'string'}}**

Param defs

**expand_params**(*resource*)
Called after creation. Use to expand PARAM_DEFS, e.g. from Resource metadata like WMS Capabilities. See e.g. WmsGetMapV1 class. :param resource: :return: None

**get_metadata**(*resource*, *version='2.0.1'*)
Get metadata, specific per Resource type. :param resource: :param version: :return: Metadata object

**class** GeoHealthCheck.plugins.probe.ogcfeat.**OGCFeatCaps**
Bases: *GeoHealthCheck.probe.Probe*

Probe for OGC API - Features endpoint url

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {'default': True, 'set_params': {'strings': {'name': 'Contains required strings', 'value': ['/conformance', '/collections', 'service', 'links']}}}, 'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}, 'GeoHealthCheck.plugins.check.checks.JsonParse': {'default': True}}**
Validate OGC API Features (OAFeat) endpoint landing page

**class** GeoHealthCheck.plugins.probe.ogcfeat.**OGCFeatDrilldown**
Bases: *GeoHealthCheck.probe.Probe*

Probe for OGC API Features (OAFeat) endpoint "drilldown" or "crawl": starting with top endpoint: get Collections and fetch Features on them etc. Uses the OWSLib owslib.ogcapi package.

**PARAM_DEFS = {'drilldown_level': {'default': 'basic', 'description': 'How thorough the drilldown should be. basic: test presence endpoints, full: go through collections, fetch Features', 'range': ['basic', 'full'], 'required': True, 'type': 'string'}}**
Param defs

**perform_request**()
Perform the drilldown. See https://github.com/geopython/OWSLib/blob/master/tests/doctests/wfs3_GeoServerCapabilities.txt

**class** GeoHealthCheck.plugins.probe.ogcfeat.**OGCFeatOpenAPIValidator**
Bases: *GeoHealthCheck.probe.Probe*

Probe for OGC API Features (OAFeat) OpenAPI Document Validation. Uses https://pypi.org/project/openapi-spec-validator/.

**REQUEST_METHOD = 'GET'**
Param defs

**perform_request**()
Perform the validation. Uses https://github.com/p1c2u/openapi-spec-validator on the specfile (dict) returned from the OpenAPI endpoint.

**class** GeoHealthCheck.plugins.probe.mapbox.**TileJSON**
Bases: *GeoHealthCheck.probe.Probe*

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}}**
Checks avail

**perform_request**()
Perform actual request to service

**class** GeoHealthCheck.plugins.probe.ogc3dtiles.**OGC3DTiles**
Bases: *GeoHealthCheck.probe.Probe*

**CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default': True}}**
Checks avail

**perform_request**()
Perform actual request to service

class GeoHealthCheck.plugins.probe.esrifs.**ESRIFSDrilldown**
Bases: *GeoHealthCheck.probe.Probe*

Probe for ESRI FeatureServer endpoint "drilldown": starting with top /FeatureServer endpoint: get Layers and get Features on these. Test e.g. from https://sampleserver6.arcgisonline.com/arcgis/rest/services (at least sampleserver6 is ArcGIS 10.6.1 supporting Paging).

**PARAM_DEFS = {'drilldown_level': {'default': 'basic', 'description': 'How heavy the drilldown should be. basic: test presence of Capabilities, full: go through Layers, get Features', 'range': ['basic', 'full'], 'required': True, 'type': 'string'}}**
Param defs

**perform_request**()
Perform the drilldown.

class GeoHealthCheck.plugins.probe.ghcreport.**GHCEmailReporter**
Bases: *GeoHealthCheck.probe.Probe*

Probe for GeoHealthCheck endpoint recurring status Reporter. When invoked it will get the overall status of the GHC Endpoint and email a summary, with links to more detailed reports.

**PARAM_DEFS = {'email': {'default': None, 'description': 'A comma-separated list of email addresses to send status report to', 'required': True, 'type': 'string'}}**
Param defs

**perform_request**()
Perform the reporting.

### Plugins - Checks

*Checks* apply to a single *Probe* instance. They are responsible for checking request results from their *Probe*.

class GeoHealthCheck.plugins.check.checks.**ContainsStrings**
Bases: *GeoHealthCheck.check.Check*

Checks if HTTP response contains given strings (keywords).

**PARAM_DEFS = {'strings': {'default': None, 'description': 'The string text(s) that should be contained in response (comma-separated)', 'range': None, 'required': True, 'type': 'stringlist'}}**
Param defs

**perform**()
Perform this Check's specific check. TODO: return Result object. :return:

class GeoHealthCheck.plugins.check.checks.**HttpHasContentType**
Bases: *GeoHealthCheck.plugins.check.checks.HttpHasHeaderValue*

Checks if HTTP response has content type.

**PARAM_DEFS = {'header_name': {'default': None, 'description': 'The HTTP header name', 'range': None, 'required': True, 'type': 'string', 'value': 'content-type'}, 'header_value': {'default': None, 'description': 'The HTTP header value', 'range': None, 'required': True, 'type': 'string'}}**

Params defs for header content type.

**perform**()

Perform this Check's specific check. TODO: return Result object. :return:

**class** GeoHealthCheck.plugins.check.checks.**HttpHasHeaderValue**

Bases: *GeoHealthCheck.check.Check*

Checks if header exists and has given header value. See http://docs.python-requests.org/en/master/user/quickstart

**PARAM_DEFS = {'header_name': {'default': None, 'description': 'The HTTP header name', 'range': None, 'required': True, 'type': 'string'}, 'header_value': {'default': None, 'description': 'The HTTP header value', 'range': None, 'required': True, 'type': 'string'}}**

Param defs

**perform**()

Perform this Check's specific check. TODO: return Result object. :return:

**class** GeoHealthCheck.plugins.check.checks.**HttpHasImageContentType**

Bases: *GeoHealthCheck.check.Check*

Checks if HTTP response has image content type.

**perform**()

Perform this Check's specific check. TODO: return Result object. :return:

**class** GeoHealthCheck.plugins.check.checks.**HttpStatusNoError**

Bases: *GeoHealthCheck.check.Check*

Checks if HTTP status code is not in the 400- or 500-range.

**perform**()

Default check: Resource should at least give no error

**class** GeoHealthCheck.plugins.check.checks.**JsonParse**

Bases: *GeoHealthCheck.check.Check*

Checks if HTTP response is valid JSON.

**perform**()

Perform this Check's specific check. TODO: return Result object. :return:

**class** GeoHealthCheck.plugins.check.checks.**NotContainsOwsException**

Bases: *GeoHealthCheck.plugins.check.checks.NotContainsStrings*

Checks if HTTP response NOT contains given OWS Exceptions.

**PARAM_DEFS = {'strings': {'default': None, 'description': 'The string text(s) that should be contained in response (comma-separated)', 'range': None, 'required': True, 'type': 'stringlist', 'value': ['ExceptionReport>', 'ServiceException>']}}**

Param defs

**class** GeoHealthCheck.plugins.check.checks.**NotContainsStrings**

Bases: *GeoHealthCheck.plugins.check.checks.ContainsStrings*

Checks if HTTP response NOT contains given strings (keywords).

**PARAM_DEFS = {'strings': {'default': None, 'description': 'The string text(s) that should NOT be\n contained in response (comma-separated)', 'range': None, 'required': True, 'type': 'stringlist'}}**
Param defs

**perform()**
Perform this Check's specific check. TODO: return Result object. :return:

**class** GeoHealthCheck.plugins.check.checks.**XmlParse**
Bases: *GeoHealthCheck.check.Check*

Checks if HTTP response is valid XML.

**perform()**
Perform this Check's specific check. TODO: return Result object. :return:

### Plugins - Resource Auth

*ResourceAuth* apply to optional authentication for a *Resource* instance. They are responsible for handling any (UI) configuration, encoding and execution of specific HTTP authentication methods for the *Resource* endpoint.

**class** GeoHealthCheck.plugins.resourceauth.resourceauths.**BasicAuth**
Bases: *GeoHealthCheck.resourceauth.ResourceAuth*

Basic authentication.

**PARAM_DEFS = {'password': {'default': None, 'description': 'Password', 'range': None, 'required': True, 'type': 'password'}, 'username': {'default': None, 'description': 'Username', 'range': None, 'required': True, 'type': 'string'}}**
Param defs

**encode_auth_header_val()**
Get encoded authorization header value from config data. Authorization scheme-specific.

```
{
  'type': 'Basic',
  'data': {
    'username': 'the_user',
    'password': 'the_password'
  }
}
```

**Returns** None or http Basic auth header value

**class** GeoHealthCheck.plugins.resourceauth.resourceauths.**BearerTokenAuth**
Bases: *GeoHealthCheck.resourceauth.ResourceAuth*

Bearer token auth

**PARAM_DEFS = {'token': {'default': None, 'description': 'Token string', 'range': None, 'required': True, 'type': 'password'}}**
Param defs

**encode_auth_header_val()**
Get encoded authorization header value from config data. Authorization scheme-specific.

```
{
  'type': 'Bearer Token',
  'data': {
      'token': 'the_token'
   }
}
```

**Returns** None or http auth header value

**class** GeoHealthCheck.plugins.resourceauth.resourceauths.**NoAuth**
Bases: *GeoHealthCheck.resourceauth.ResourceAuth*

Checks if header exists and has given header value. See http://docs.python-requests.org/en/master/user/quickstart

**PARAM_DEFS = {}**
Param defs

**encode**()
Encode/encrypt auth dict structure. :return: encoded string

### Plugins - Geocoder

*Geocoder* apply to geocoder services. They are responsible for geolocating a server on a map.

**class** GeoHealthCheck.plugins.geocode.fixedlocation.**FixedLocation**
Bases: GeoHealthCheck.geocoder.Geocoder

Spoof getting a geolocation for a server by provinding a fixed lat, lon result. The lat, lon can be specified in the initialisation parameters. When omitted: default to 0, 0.

**LATITUDE = 0**
Parameter with the default latitude position. This is overruled when the latitude option is provided in the init step.

**LONGITUDE = 0**
Parameter with the default longitude position. This is overruled when the longitude option is provided in the init step.

**init**(*geocode_vars={}*)
Initialise the geocoder service with an optional dictionary.

When the dictionary contains the element *lat* and/or *lon*, then these values are used to position the server.

**locate**(*_=None*)
Perform a geocoding to locate a server. In this case it will render a fixed position, so provinding the adress of the server is optional.

**class** GeoHealthCheck.plugins.geocode.webgeocoder.**HttpGeocoder**
Bases: GeoHealthCheck.geocoder.Geocoder

A base class for geocoders on the web.

It is intended to use a *subclass* of this class and implement the *make_call* method.

**after_request**()
After running actual request to service

**before_request**()
Before running actual request to service

**locate**(*ip*)
Class method to create and run a single Probe instance. Follows strict sequence of method calls. Each method can be overridden in subclass.

**run_request**(*ip*)
Prepare actual request to service

**class** GeoHealthCheck.plugins.geocode.webgeocoder.**HttpGetGeocoder**
Bases: *GeoHealthCheck.plugins.geocode.webgeocoder.HttpGeocoder*

A geocoder plugin using a http GET request.

Use the *init* method (**not** the dunder methode) to initialise the geocoder. Provide a dict with keys: *geocoder_url*, *lat_field*, *lon_field*, and optional *template* and *parameters*. The *geocoder_url* parameter should include *{hostname}* where the *locate* function will substitute the server name that needs to be located. The *lat_field* and *lon_field* parameters specify the field names of the lat/lon in the json response.

**class** GeoHealthCheck.plugins.geocode.webgeocoder.**HttpPostGeocoder**
Bases: *GeoHealthCheck.plugins.geocode.webgeocoder.HttpGeocoder*

A geocoder plugin using a http POST request.

Use the *init* method (**not** the dunder methode) to initialise the geocoder. Provide a dict with keys: *geocoder_url*, *lat_field*, *lon_field*, and optional *template* and *parameters*. The *geocoder_url* parameter should include *{hostname}* where the *locate* function will substitute the server name that needs to be located. The *lat_field* and *lon_field* parameters specify the field names of the lat/lon in the json response.

## 3.7 License

## 3.8 Contact

The website geohealthcheck.org is the main entry point.

All development is done via GitHub: see https://github.com/geopython/geohealthcheck.

### 3.8.1 Links

- website: http://geohealthcheck.org
- GitHub: https://github.com/geopython/geohealthcheck
- Demo: https://demo.geohealthcheck.org
- Presentation: http://geohealthcheck.org/presentation
- Gitter Chat: https://gitter.im/geopython/GeoHealthCheck

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## g


GeoHealthCheck.check, 52
GeoHealthCheck.plugin, 50
GeoHealthCheck.plugins.check.checks, 62
GeoHealthCheck.plugins.geocode.fixedlocation, 65
GeoHealthCheck.plugins.geocode.webgeocoder, 65
GeoHealthCheck.plugins.probe.esrifs, 62
GeoHealthCheck.plugins.probe.ghcreport, 62
GeoHealthCheck.plugins.probe.http, 53
GeoHealthCheck.plugins.probe.mapbox, 61
GeoHealthCheck.plugins.probe.ogc3dtiles, 61
GeoHealthCheck.plugins.probe.ogcfeat, 61
GeoHealthCheck.plugins.probe.owsgetcaps, 53
GeoHealthCheck.plugins.probe.sta, 59
GeoHealthCheck.plugins.probe.tms, 56
GeoHealthCheck.plugins.probe.wcs, 60
GeoHealthCheck.plugins.probe.wfs, 59
GeoHealthCheck.plugins.probe.wms, 55
GeoHealthCheck.plugins.probe.wmsdrilldown, 56
GeoHealthCheck.plugins.probe.wmts, 57
GeoHealthCheck.plugins.resourceauth.resourceauths, 64
GeoHealthCheck.probe, 50
GeoHealthCheck.resourceauth, 52
GeoHealthCheck.result, 52

# INDEX


## A

actual_request() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
after_request() (*GeoHealthCheck.plugins.geocode.webgeocoder.HttpGeocoder method*), 65
after_request() (*GeoHealthCheck.plugins.probe.owsgetcaps.WmtsGetCaps method*), 55
after_request() (*GeoHealthCheck.probe.Probe method*), 51
AUTHOR (*GeoHealthCheck.plugin.Plugin attribute*), 50

## B

BasicAuth (*class in GeoHealthCheck.plugins.resourceauth.resourceauths*), 64
BearerTokenAuth (*class in GeoHealthCheck.plugins.resourceauth.resourceauths*), 64
before_request() (*GeoHealthCheck.plugins.geocode.webgeocoder.HttpGeocoder method*), 65
before_request() (*GeoHealthCheck.plugins.probe.owsgetcaps.WmtsGetCaps method*), 55
before_request() (*GeoHealthCheck.plugins.probe.tms.TmsGetTileAll method*), 57
before_request() (*GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBboxAll method*), 60
before_request() (*GeoHealthCheck.plugins.probe.wms.WmsGetMapV1All method*), 56
before_request() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
before_request() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTileAll method*), 59
before_request() (*GeoHealthCheck.probe.Probe method*), 51

## C

calc_result() (*GeoHealthCheck.probe.Probe method*), 51
calculate_center_tile() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
Check (*class in GeoHealthCheck.check*), 52
check_capabilities() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
CheckResult (*class in GeoHealthCheck.result*), 52
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.http.HttpGet attribute*), 53
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.mapbox.TileJSON attribute*), 61
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.ogc3dtiles.OGC3DTiles attribute*), 61
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.ogcfeat.OGCFeatCaps attribute*), 61
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps attribute*), 54
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.sta.StaCaps attribute*), 59
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.sta.StaGetEntities attribute*), 59
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.tms.TmsCaps attribute*), 56
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.tms.TmsGetTile attribute*), 57
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.wcs.WcsGetCoverage attribute*), 60
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox attribute*), 59
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 attribute*), 55
CHECKS_AVAIL (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile attribute*), 57
CHECKS_AVAIL (*GeoHealthCheck.probe.Probe attribute*), 51

ContainsStrings (*class in GeoHealthCheck.plugins.check.checks*), 62
copy() (*GeoHealthCheck.plugin.Plugin static method*), 50
CswGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 53

## D

decode() (*GeoHealthCheck.resourceauth.ResourceAuth static method*), 52
DESCRIPTION (*GeoHealthCheck.plugin.Plugin attribute*), 50

## E

encode() (*GeoHealthCheck.plugins.resourceauth.resourceauths.GetTokenAuth method*), 65
encode() (*GeoHealthCheck.resourceauth.ResourceAuth method*), 52
encode_auth_header_val() (*GeoHealthCheck.plugins.resourceauth.resourceauths.BasicAuth method*), 64
encode_auth_header_val() (*GeoHealthCheck.plugins.resourceauth.resourceauths.GetTokenAuth method*), 64
ESRIFSDrilldown (*class in GeoHealthCheck.plugins.probe.esrifs*), 62
expand_params() (*GeoHealthCheck.plugins.probe.tms.TmsGetTile method*), 57
expand_params() (*GeoHealthCheck.plugins.probe.tms.TmsGetTileAll method*), 57
expand_params() (*GeoHealthCheck.plugins.probe.wcs.WcsGetCoverage method*), 60
expand_params() (*GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox method*), 60
expand_params() (*GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBboxAll method*), 60
expand_params() (*GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 method*), 56
expand_params() (*GeoHealthCheck.plugins.probe.wms.WmsGetMapV1All method*), 56
expand_params() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
expand_params() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTileAll method*), 59
expand_params() (*GeoHealthCheck.probe.Probe method*), 51

## F

FixedLocation (*class in GeoHealthCheck.plugins.geocode.fixedlocation*), 65

## G

GeoHealthCheck.check
    module, 52
GeoHealthCheck.plugin
    module, 50
GeoHealthCheck.plugins.check.checks
    module, 62
GeoHealthCheck.plugins.geocode.fixedlocation
    module, 65
GeoHealthCheck.plugins.geocode.webgeocoder
    module, 65
GeoHealthCheck.plugins.probe.esrifs
    module, 62
GeoHealthCheck.plugins.probe.ghcreport
    module, 62
GeoHealthCheck.plugins.probe.http
    module, 53
GeoHealthCheck.plugins.probe.mapbox
    module, 61
GeoHealthCheck.plugins.probe.ogc3dtiles
    module, 61
GeoHealthCheck.plugins.probe.ogcfeat
    module, 61
GeoHealthCheck.plugins.probe.owsgetcaps
    module, 53
GeoHealthCheck.plugins.probe.sta
    module, 59
GeoHealthCheck.plugins.probe.tms
    module, 56
GeoHealthCheck.plugins.probe.wcs
    module, 60
GeoHealthCheck.plugins.probe.wfs
    module, 59
GeoHealthCheck.plugins.probe.wms
    module, 55
GeoHealthCheck.plugins.probe.wmsdrilldown
    module, 56
GeoHealthCheck.plugins.probe.wmts
    module, 57
GeoHealthCheck.plugins.resourceauth.resourceauths
    module, 64
GeoHealthCheck.probe
    module, 50
GeoHealthCheck.resourceauth
    module, 52
GeoHealthCheck.result

module, 52
get_auth_defs() (GeoHealthCheck.resourceauth.ResourceAuth static method), 52
get_auth_header() (GeoHealthCheck.resourceauth.ResourceAuth method), 52
get_default_parameter_values() (GeoHealthCheck.plugin.Plugin method), 50
get_metadata() (GeoHealthCheck.plugins.probe.tms.TmsGetTile method), 57
get_metadata() (GeoHealthCheck.plugins.probe.wcs.WcsGetCoverage method), 61
get_metadata() (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox method), 60
get_metadata() (GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 method), 56
get_metadata() (GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method), 58
get_metadata() (GeoHealthCheck.probe.Probe method), 51
get_metadata_cached() (GeoHealthCheck.probe.Probe method), 51
get_param() (GeoHealthCheck.plugin.Plugin method), 50
get_param_defs() (GeoHealthCheck.plugin.Plugin method), 50
get_plugin_vars() (GeoHealthCheck.plugin.Plugin method), 50
get_plugin_vars() (GeoHealthCheck.probe.Probe method), 51
get_plugins() (GeoHealthCheck.plugin.Plugin static method), 50
get_request_headers() (GeoHealthCheck.plugins.probe.http.HttpPost method), 53
get_var_names() (GeoHealthCheck.plugin.Plugin method), 50
get_var_names() (GeoHealthCheck.probe.Probe method), 52
GHCEmailReporter (class in GeoHealthCheck.plugins.probe.ghcreport), 62

## H

HttpGeocoder (class in GeoHealthCheck.plugins.geocode.webgeocoder), 65
HttpGet (class in GeoHealthCheck.plugins.probe.http), 53
HttpGetGeocoder (class in GeoHealthCheck.plugins.geocode.webgeocoder), 66
HttpGetQuery (class in GeoHealthCheck.plugins.probe.http), 53
HttpHasContentType (class in GeoHealthCheck.plugins.check.checks), 62
HttpHasHeaderValue (class in GeoHealthCheck.plugins.check.checks), 63
HttpHasImageContentType (class in GeoHealthCheck.plugins.check.checks), 63
HttpPost (class in GeoHealthCheck.plugins.probe.http), 53
HttpPostGeocoder (class in GeoHealthCheck.plugins.geocode.webgeocoder), 66
HttpStatusNoError (class in GeoHealthCheck.plugins.check.checks), 63

## I

init() (GeoHealthCheck.check.Check method), 52
init() (GeoHealthCheck.plugins.geocode.fixedlocation.FixedLocation method), 65
init() (GeoHealthCheck.probe.Probe method), 52
init() (GeoHealthCheck.resourceauth.ResourceAuth method), 52

## J

JsonParse (class in GeoHealthCheck.plugins.check.checks), 63

## L

LATITUDE (GeoHealthCheck.plugins.geocode.fixedlocation.FixedLocation attribute), 65
locate() (GeoHealthCheck.plugins.geocode.fixedlocation.FixedLocation method), 65
locate() (GeoHealthCheck.plugins.geocode.webgeocoder.HttpGeocoder method), 66
LONGITUDE (GeoHealthCheck.plugins.geocode.fixedlocation.FixedLocation attribute), 65

## M

merge() (GeoHealthCheck.plugin.Plugin static method), 50
METADATA_CACHE (GeoHealthCheck.probe.Probe attribute), 51
module
    GeoHealthCheck.check, 52
    GeoHealthCheck.plugin, 50
    GeoHealthCheck.plugins.check.checks, 62
    GeoHealthCheck.plugins.geocode.fixedlocation, 65
    GeoHealthCheck.plugins.geocode.webgeocoder, 65

GeoHealthCheck.plugins.probe.esrifs, 62
GeoHealthCheck.plugins.probe.ghcreport, 62
GeoHealthCheck.plugins.probe.http, 53
GeoHealthCheck.plugins.probe.mapbox, 61
GeoHealthCheck.plugins.probe.ogc3dtiles, 61
GeoHealthCheck.plugins.probe.ogcfeat, 61
GeoHealthCheck.plugins.probe.owsgetcaps, 53
GeoHealthCheck.plugins.probe.sta, 59
GeoHealthCheck.plugins.probe.tms, 56
GeoHealthCheck.plugins.probe.wcs, 60
GeoHealthCheck.plugins.probe.wfs, 59
GeoHealthCheck.plugins.probe.wms, 55
GeoHealthCheck.plugins.probe.wmsdrilldown, 56
GeoHealthCheck.plugins.probe.wmts, 57
GeoHealthCheck.plugins.resourceauth.resourceauths, 64
GeoHealthCheck.probe, 50
GeoHealthCheck.resourceauth, 52
GeoHealthCheck.result, 52

## N

NAME (*GeoHealthCheck.plugin.Plugin attribute*), 50
NoAuth (*class in GeoHealthCheck.plugins.resourceauth.resourceauths*), 65
NotContainsOwsException (*class in GeoHealthCheck.plugins.check.checks*), 63
NotContainsStrings (*class in GeoHealthCheck.plugins.check.checks*), 63

## O

OGC3DTiles (*class in GeoHealthCheck.plugins.probe.ogc3dtiles*), 61
OGCFeatCaps (*class in GeoHealthCheck.plugins.probe.ogcfeat*), 61
OGCFeatDrilldown (*class in GeoHealthCheck.plugins.probe.ogcfeat*), 61
OGCFeatOpenAPIValidator (*class in GeoHealthCheck.plugins.probe.ogcfeat*), 61
OwsGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 54

## P

PARAM_DEFS (*GeoHealthCheck.plugin.Plugin attribute*), 50
PARAM_DEFS (*GeoHealthCheck.plugins.check.checks.ContainsStrings attribute*), 62
PARAM_DEFS (*GeoHealthCheck.plugins.check.checks.HttpHasContentType attribute*), 62
PARAM_DEFS (*GeoHealthCheck.plugins.check.checks.HttpHasHeaderValue attribute*), 63
PARAM_DEFS (*GeoHealthCheck.plugins.check.checks.NotContainsOwsExcep attribute*), 63
PARAM_DEFS (*GeoHealthCheck.plugins.check.checks.NotContainsStrings attribute*), 63
PARAM_DEFS (*GeoHealthCheck.plugins.probe.esrifs.ESRIFSDrilldown attribute*), 62
PARAM_DEFS (*GeoHealthCheck.plugins.probe.ghcreport.GHCEmailReporte attribute*), 62
PARAM_DEFS (*GeoHealthCheck.plugins.probe.http.HttpGetQuery attribute*), 53
PARAM_DEFS (*GeoHealthCheck.plugins.probe.http.HttpPost attribute*), 53
PARAM_DEFS (*GeoHealthCheck.plugins.probe.ogcfeat.OGCFeatDrilldown attribute*), 61
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.CswGetCaps attribute*), 54
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps attribute*), 54
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.SosGetCaps attribute*), 54
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.WcsGetCaps attribute*), 54
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.WfsGetCaps attribute*), 54
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.WmsGetCaps attribute*), 55
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.WmtsGetCaps attribute*), 55
PARAM_DEFS (*GeoHealthCheck.plugins.probe.owsgetcaps.WpsGetCaps attribute*), 55
PARAM_DEFS (*GeoHealthCheck.plugins.probe.sta.StaGetEntities attribute*), 59
PARAM_DEFS (*GeoHealthCheck.plugins.probe.tms.TmsGetTile attribute*), 57
PARAM_DEFS (*GeoHealthCheck.plugins.probe.wcs.WcsGetCoverage attribute*), 60
PARAM_DEFS (*GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox attribute*), 59
PARAM_DEFS (*GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 attribute*), 55
PARAM_DEFS (*GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown attribute*), 56
PARAM_DEFS (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile attribute*), 57
PARAM_DEFS (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTileAll attribute*), 58
PARAM_DEFS (*GeoHealthCheck.plugins.resourceauth.resourceauths.BasicA attribute*), 64
PARAM_DEFS (*GeoHealthCheck.plugins.resourceauth.resourceauths.Bearer attribute*), 64
PARAM_DEFS (*GeoHealthCheck.plugins.resourceauth.resourceauths.NoAuth attribute*), 65

PARAM_DEFS (*GeoHealthCheck.probe.Probe attribute*), 51
perform() (*GeoHealthCheck.check.Check method*), 52
perform() (*GeoHealthCheck.plugins.check.checks.ContainsStrings method*), 62
perform() (*GeoHealthCheck.plugins.check.checks.HttpHasCode method*), 63
perform() (*GeoHealthCheck.plugins.check.checks.HttpHasHeaderValue method*), 63
perform() (*GeoHealthCheck.plugins.check.checks.HttpHasImageContentType method*), 63
perform() (*GeoHealthCheck.plugins.check.checks.HttpStatusNoError method*), 63
perform() (*GeoHealthCheck.plugins.check.checks.JsonParse method*), 63
perform() (*GeoHealthCheck.plugins.check.checks.NotContainsStrings method*), 64
perform() (*GeoHealthCheck.plugins.check.checks.XmlParse method*), 64
perform_get_request() (*GeoHealthCheck.probe.Probe method*), 52
perform_post_request() (*GeoHealthCheck.probe.Probe method*), 52
perform_request() (*GeoHealthCheck.plugins.probe.esrifs.ESRIFSDrilldown method*), 62
perform_request() (*GeoHealthCheck.plugins.probe.ghcreport.GHCEmailReporter method*), 62
perform_request() (*GeoHealthCheck.plugins.probe.mapbox.TileJSON method*), 61
perform_request() (*GeoHealthCheck.plugins.probe.ogc3dtiles.OGC3DTiles method*), 62
perform_request() (*GeoHealthCheck.plugins.probe.ogcfeat.OGCFeatDrilldown method*), 61
perform_request() (*GeoHealthCheck.plugins.probe.ogcfeat.OGCFeatOpenAPIValidator method*), 61
perform_request() (*GeoHealthCheck.plugins.probe.tms.TmsGetTileAll method*), 57
perform_request() (*GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBboxAll method*), 60
perform_request() (*GeoHealthCheck.plugins.probe.wms.WmsGetMapV1All method*), 56
perform_request() (*GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown method*), 56
perform_request() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
perform_request() (*GeoHealthCheck.probe.Probe method*), 52
Plugin (*class in GeoHealthCheck.plugin*), 50
Probe (*class in GeoHealthCheck.probe*), 50
ProbeResult (*class in GeoHealthCheck.result*), 53

## R

REQUEST_HEADERS (*GeoHealthCheck.probe.Probe attribute*), 51
REQUEST_METHOD (*GeoHealthCheck.plugins.probe.ogcfeat.OGCFeatOpenAPIValidator attribute*), 61
REQUEST_METHOD (*GeoHealthCheck.probe.Probe attribute*), 51
REQUEST_TEMPLATE (*GeoHealthCheck.probe.Probe attribute*), 51
RESOURCE_TYPE (*GeoHealthCheck.probe.Probe attribute*), 51
ResourceAuth (*class in GeoHealthCheck.resourceauth*), 52
ResourceResult (*class in GeoHealthCheck.result*), 53
Result (*class in GeoHealthCheck.result*), 53
run() (*GeoHealthCheck.probe.Probe static method*), 52
run_checks() (*GeoHealthCheck.probe.Probe method*), 52
run_request() (*GeoHealthCheck.plugins.geocode.webgeocoder.HttpGeoc method*), 66
run_request() (*GeoHealthCheck.probe.Probe method*), 52

## S

SosGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 54
StaCaps (*class in GeoHealthCheck.plugins.probe.sta*), 59
StaGetEntities (*class in GeoHealthCheck.plugins.probe.sta*), 59
STANDARD_REQUEST_HEADERS (*GeoHealthCheck.probe.Probe attribute*), 51

## T

test_kvp_rest() (*GeoHealthCheck.plugins.probe.wmts.WmtsGetTile method*), 58
TileJSON (*class in GeoHealthCheck.plugins.probe.mapbox*), 61
TmsCaps (*class in GeoHealthCheck.plugins.probe.tms*), 56
TmsGetTile (*class in GeoHealthCheck.plugins.probe.tms*), 57

TmsGetTileAll (*class in GeoHealthCheck.plugins.probe.tms*), 57

## W

WcsGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 54
WcsGetCoverage (*class in GeoHealthCheck.plugins.probe.wcs*), 60
WfsGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 54
WfsGetFeatureBbox (*class in GeoHealthCheck.plugins.probe.wfs*), 59
WfsGetFeatureBboxAll (*class in GeoHealthCheck.plugins.probe.wfs*), 60
WmsDrilldown (*class in GeoHealthCheck.plugins.probe.wmsdrilldown*), 56
WmsGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 55
WmsGetMapV1 (*class in GeoHealthCheck.plugins.probe.wms*), 55
WmsGetMapV1All (*class in GeoHealthCheck.plugins.probe.wms*), 56
WmtsGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 55
WmtsGetTile (*class in GeoHealthCheck.plugins.probe.wmts*), 57
WmtsGetTileAll (*class in GeoHealthCheck.plugins.probe.wmts*), 58
WpsGetCaps (*class in GeoHealthCheck.plugins.probe.owsgetcaps*), 55

## X

XmlParse (*class in GeoHealthCheck.plugins.check.checks*), 64