
GeoHealthCheck Documentation

Release 0.4-dev

Tom Kralidis

Mar 21, 2018

Contents

1	Overview	1
2	Features	3
3	Links	5
3.1	Installation	5
3.2	Configuration	7
3.3	Adminstration	10
3.4	Architecture	13
3.5	Plugins	13
3.6	License	33
3.7	Contact	33
4	Indices and tables	35
	Python Module Index	37

CHAPTER 1

Overview

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

It can be used to monitor overall health of OGC services like WMS, WFS, WCS, WMTS, SOS, CSW and more, but also standard web(-API) URLs.

CHAPTER 2

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 notifications

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

This document applies to GHC version 0.4-dev and was generated on Mar 21, 2018. 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

Easiest and quickest install/run 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 microframework and uses Flask-SQLAlchemy for database interaction and Flask-Login for authorization. Flask-Migrate with Alembic and Flask-Script support is used for database upgrades.

OWSLib is used to interact with OGC Web Services.

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

3.1.3 Install

Note: it is strongly recommended to install 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.

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

```
virtualenv ghc && cd 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_RUN_FREQUENCY
# - GHC_SELF_REGISTER
# - GHC_NOTIFICATIONS
# - GHC_NOTIFICATIONS_VERBOSITY
# - GHC_ADMIN_EMAIL
# - GHC_NOTIFICATIONS_EMAIL
# - GHC_SITE_TITLE
# - GHC_SITE_URL
# - GHC_SMTP # if GHC_NOTIFICATIONS is enabled
# - GHC_MAP # or use default settings

# init database
python GeoHealthCheck/models.py create

# start server
python GeoHealthCheck/app.py # http://localhost:8000/
```

Schedule the cronjobs.

```
# edit local paths to scripts
vi jobs.cron

# enable cron
crontab jobs.cron
```

3.1.4 Upgrade

An existing GHC database installation can be upgraded with:

```
# In the top directory (e.g. the toplevel 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==2.0.3*, 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 already uptodate
- 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.

3.1.5 Running

Start using the built-in `mod_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
```

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

3.2 Configuration

Core configuration is set by GeoHealthCheck in `GeoHealthCheck/config_main.py`. You can override these settings in `instance/config_site.py`:

- **SQLALCHEMY_DATABASE_URI**: the database configuration. See the SQLAlchemy documentation for more info
- **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_RUN_FREQUENCY**: cron keyword used to indicate frequency of runs (i.e. `hourly`, `daily`, `monthly`)
- **GHC_SELF_REGISTER**: allow registrations from users on the website
- **GHC_NOTIFICATIONS**: turn on email 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_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**: 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_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

3.2.1 Enabling or disabling languages

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 a new language (e.g. this needs a new translation file called `*.po`) is to be added, make a copy of one of the folders in `GeoHealthCheck/translations/`; rename the folder to the desired language (e.g. ‘de’ for german); start editing the file in `LC_MESSAGES/messages.po` and add 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"
```

3.2.2 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`.

3.2.3 Configuring notifications

GeoHealthCheck can send notifications to various channels, depending on resource. Notifications can be configured in edit form:

The screenshot shows a configuration form with two main sections:

- Notify emails:** A text input field containing the email addresses `test@test.com, other@test.com`.
- Notify webhooks:** A text input field containing `http://localhost/`, `field=value`, and `other_field=value`.

Below the webhooks section, there is a text area with a placeholder: "Enter url for webhook. Optionally, enter payload as list of key=value lines or JSON object." The text area contains the following content:

```
http://url/path
field_name=value
or
{"field_name": "value"}
```

A plus sign (+) is visible at the bottom of the text area, indicating that more webhooks can be added.

Fig. 3.1: Figure - GHC notifications configuration

There are two channels implemented:

Email

Notifications can be send to designated emails. If set in config, GeoHealthCheck will send notifications for all re-sources 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 resource is created, owner's email is added to the list. User can add any email address, even for users that are not registered in GeoHealthCheck instance. When editing emails list for a resource, user will get address suggestions based on emails added for other resources by that user. Multiple emails should be separated with comma (,) char.

Webhook

Notifications can be also send as webhooks through *POST* request. Resource can have arbitrary number of webhooks configured.

In edit form, user can add webhook configuration. Each webhook should be entered in separate field. Each webhook should contain at least URL to which *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

Configuration can hold additional form payload that will be send along with GHC fields. Syntax for configuration:

- first line should be url to which webhook will be send
- second line should be empty
- third line (and subsequent) are used to store custom payload, and should contain either: * each pair of field and value in separate lines (*field=value*) * JSONified object, which properties will be used as form fields

Configuration samples:

- just an url

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

- url with fields as field-value pairs

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

foo=bar
otherfield=someothervalue
```

- url and payload as JSON:

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

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

3.3 Adminstration

3.3.1 Database

For database administration the following commands are available.

NB, although SQLite works fine for most installations it is recommended for performance and reliability to use PostgreSQL.

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

NB for detailed reporting data only JSON is supported.

3.3.2 User Management

On setup a single *admin* user is created interactively.

Via the **GHC_SELF_REGISTER** config setting, you allow/disallow registrations from users on the website.

3.3.3 Adding Resources

When being logged in, click the Add+ button for adding new resources.

The folowing resource types are available:

- File Transfer Protocol (FTP)
- Web Map Service (WMS)
- Web Address (URL)
- Catalogue Service (CSW)
- Web Map Tile Service (WMTS)
- Web Processing Service (WPS)
- Web Coverage Service (WCS)

- Web Feature Service (WFS)
- Tile Map Service (TMS)
- Web Accessible Folder (WAF)
- Sensor Observation Service (SOS)
- [SensorThings API \(STA\)](#)
- GeoNode autodiscovery

3.3.4 Deleting Resources

Open the resource details by clicking its name in the resources list at the Dashboard page. Under the resource title is a red Delete button.

3.3.5 Editing Resources

Open the resource details by clicking its name in the resources list at the Dashboard page. Under the resource title is a blue Edit button.

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

- Resource name
- Resource Tags
- Notification recipients
- Resource Probes, select from “Probes Available”
- For each Probe: Probe parameters
- For each Probe: Probe Checks, select from “Checks Available”
- For each Check: Checks parameters

By default, when resource is created, owner’s email will be added to notifications, however, resource can have arbitrary number or emails to notify.

3.3.6 Scheduling Runs

- Permanent Jobs

Edit the file `jobs.cron` 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_SCRIPTS/windows/python /path/to/GeoHealthCheck/GeoHealthCheck/models.py run` to the commandline to test if they work successfully. 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`.

- interactive TBF

3.3.7 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 build locally.

3.3.8 GeoNode Resource type notes

GeoNode Resource is a virtual resource. It represents one GeoNode instance, but underneath it's trying autodiscovery of OWS endpoints available in that instance. Note, that OWS autodiscovery feature is optional, and you should check if GeoNode instance has this feature enabled.

When adding *GeoNode instance* Resource, you have to enter url to instance's home page. GeoHealthCheck will construct url to target OWS endpoints listing and create relevant Resources (WMS, WFS, WMTS, OWC Resources). It will check all endpoints provided by GeoNode API, and will reject those which responded with error.

All resources added in this way will have at least one tag, which is constructed with 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 Architecture

(To be extended)

3.4.1 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

3.4.2 Data Model

3.5 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.

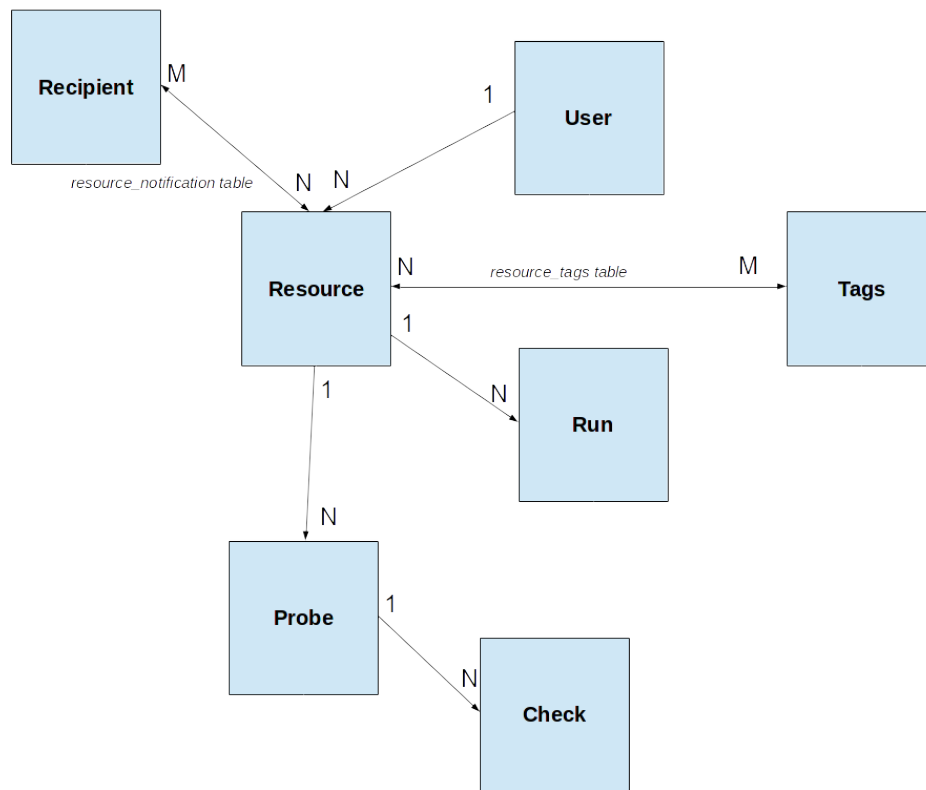


Fig. 3.2: Figure - GHC Data Model

3.5.1 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 sequentially 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*.

3.5.2 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

3.5.3 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`:

```
1 from GeoHealthCheck.probe import Probe
2
3
4 class HttpGet(Probe):
5     """
```

```

6   Do HTTP GET Request, to poll/ping any Resource bare url.
7   """
8
9   NAME = 'HTTP GET Resource URL'
10  DESCRIPTION = 'Simple HTTP GET on Resource URL'
11  RESOURCE_TYPE = '*:*'
12  REQUEST_METHOD = 'GET'
13
14  CHECKS_AVAIL = {
15      'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
16          'default': True
17      },
18      'GeoHealthCheck.plugins.check.checks.ContainsStrings': {},
19      'GeoHealthCheck.plugins.check.checks.NotContainsStrings': {},
20      'GeoHealthCheck.plugins.check.checks.HttpHasContentType': {}
21  }
22  """Checks avail"""
23

```

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`:

```

1  import sys
2  from owslib.etree import etree
3  from GeoHealthCheck.plugin import Plugin
4  from GeoHealthCheck.check import Check
5  try:
6      from html import escape # python 3.x
7  except ImportError:
8      from cgi import escape # python 2.x
9
10
11  """ Contains basic Check classes for a Probe object. """
12
13
14  class HttpStatusNoError(Check):
15      """
16      Checks if HTTP status code is not in the 400- or 500-range.
17      """

```

```

18 NAME = 'HTTP status should not be errored'
19 DESCRIPTION = 'Response should not contain a HTTP 400 or 500 range Error'
20
21
22 def __init__(self):
23     Check.__init__(self)
24
25 def perform(self):
26     """Default check: Resource should at least give no error"""
27     status = self.probe.response.status_code
28     overall_status = status / 100
29     if overall_status in [4, 5]:
30         self.set_result(False, 'HTTP Error status=%d' % status)
31
32
33 class HttpHasHeaderValue(Check):
34     """

```

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:

```

1 from GeoHealthCheck.plugin import Plugin
2 from GeoHealthCheck.probe import Probe
3
4
5 class OwsGetCaps(Probe):
6     """
7     Fetch OWS capabilities doc
8     """
9
10    AUTHOR = 'GHC Team'
11    NAME = 'OWS GetCapabilities'
12    DESCRIPTION = 'Perform GetCapabilities Operation and check validity'
13    # Abstract Base Class for OGC OWS GetCaps Probes
14    # Needs specification in subclasses
15    # RESOURCE_TYPE = 'OGC:ABC'
16
17    REQUEST_METHOD = 'GET'
18    REQUEST_TEMPLATE = \
19        '?SERVICE={service}&VERSION={version}&REQUEST=GetCapabilities'
20
21    PARAM_DEFS = {
22        'service': {
23            'type': 'string',
24            'description': 'The OWS service within resource endpoint',
25            'default': None,
26            'required': True
27        },
28        'version': {
29            'type': 'string',

```

```

30         'description': 'The OWS service version within resource endpoint',
31         'default': None,
32         'required': True,
33         'range': None
34     }
35 }
36 """Param defs, to be specified in subclasses"""
37
38 CHECKS_AVAIL = {
39     'GeoHealthCheck.plugins.check.checks.XmlParse': {
40         'default': True
41     },
42     'GeoHealthCheck.plugins.check.checks.NotContainsOwsException': {
43         'default': True
44     },
45     'GeoHealthCheck.plugins.check.checks.ContainsStrings': {
46         'set_params': {
47             'strings': {
48                 'name': 'Contains Title Element',
49                 'value': ['Title>']
50             }
51         },
52         'default': True
53     },
54 }
55 """
56 Checks avail for all specific Caps checks.
57 Optionally override Check PARAM_DEFS using set_params
58 e.g. with specific `value`.
59 """
60
61
62 class WmsGetCaps(OwsGetCaps):
63     """Fetch WMS capabilities doc"""
64
65     NAME = 'WMS GetCapabilities'
66     RESOURCE_TYPE = 'OGC:WMS'
67
68     PARAM_DEFS = Plugin.merge(OwsGetCaps.PARAM_DEFS, {
69
70         'service': {
71             'value': 'WMS'
72         },
73         'version': {
74             'default': '1.1.1',
75             'range': ['1.1.1', '1.3.0']
76         }
77     })
78     """Param defs"""
79
80
81 class WfsGetCaps(OwsGetCaps):
82     """WFS GetCapabilities Probe"""
83
84     NAME = 'WFS GetCapabilities'
85     RESOURCE_TYPE = 'OGC:WFS'
86
87     def __init__(self):

```

```

88     OwsGetCaps.__init__(self)
89
90     PARAM_DEFS = Plugin.merge(OwsGetCaps.PARAM_DEFS, {
91         'service': {
92             'value': 'WFS'
93         },
94         'version': {
95             'default': '1.1.0',
96             'range': ['1.0.0', '1.1.0', '2.0.2']
97         }
98     })
99     """Param defs"""
100
101
102 class WcsGetCaps(OwsGetCaps):
103     """WCS GetCapabilities Probe"""
104
105     NAME = 'WCS GetCapabilities'
106     RESOURCE_TYPE = 'OGC:WCS'
107
108     PARAM_DEFS = Plugin.merge(OwsGetCaps.PARAM_DEFS, {

```

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:

```
1 from GeoHealthCheck.probe import Probe
2
3
4 class StaCaps(Probe):
5     """Probe for SensorThings API main endpoint url"""
6
7     NAME = 'STA Capabilities'
8     DESCRIPTION = 'Perform STA Capabilities Operation and check validity'
9     RESOURCE_TYPE = 'OGC:STA'
10
11     REQUEST_METHOD = 'GET'
12
13     def __init__(self):
14         Probe.__init__(self)
15
16     CHECKS_AVAIL = {
17         'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
18             'default': True
19         },
20         'GeoHealthCheck.plugins.check.checks.JsonParse': {
21             'default': True
22         },
23         'GeoHealthCheck.plugins.check.checks.ContainsStrings': {
24             'default': True,
25             'set_params': {
26                 'strings': {
27                     'name': 'Must contain STA Entity names',
28                     'value': ['Things', 'Datastreams', 'Observations',
```

```

29         'FeaturesOfInterest', 'Locations']
30     }
31 }
32 },
33 }
34 """
35 Checks avail for all specific Caps checks.
36 Optionally override Check.PARAM_DEFS using set_params
37 e.g. with specific `value` or even `name`.
38 """
39
40
41 class StaGetEntities(Probe):
42     """Fetch STA entities of type and check result"""
43
44     NAME = 'STA GetEntities'
45     DESCRIPTION = 'Fetch all STA Entities of given type'
46     RESOURCE_TYPE = 'OGC:STA'
47
48     REQUEST_METHOD = 'GET'
49
50     # e.g. http://52.26.56.239:8080/OGCSensorThings/v1.0/Things
51     REQUEST_TEMPLATE = '/{entities}'
52
53     def __init__(self):
54         Probe.__init__(self)
55
56     PARAM_DEFS = {
57         'entities': {
58             'type': 'string',
59             'description': 'The STA Entity collection type',
60             'default': 'Things',
61             'required': True,
62             'range': ['Things', 'DataStreams', 'Observations',
63                     'Locations', 'Sensors', 'FeaturesOfInterest',
64                     'ObservedProperties', 'HistoricalLocations']
65         }
66     }
67     """Param defs"""
68
69     CHECKS_AVAIL = {
70         'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {
71             'default': True
72         },
73         'GeoHealthCheck.plugins.check.checks.JsonParse': {
74             'default': True
75         }
76     }
77     """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.

```

1 import random
2
3 from GeoHealthCheck.probe import Probe
4 from GeoHealthCheck.result import Result
5 from owslib.wms import WebMapService
6
7
8 class WmsDrilldown(Probe):
9     """
10     Probe for WMS endpoint "drilldown": starting
11     with GetCapabilities doc: get Layers and do
12     GetMap on them etc. Using OWSLib.WebMapService.
13
14     TODO: needs finalization.
15     """
16
17     NAME = 'WMS Drilldown'
18     DESCRIPTION = 'Traverses a WMS endpoint by drilling down from Capabilities'
19     RESOURCE_TYPE = 'OGC:WMS'
20
21     REQUEST_METHOD = 'GET'
22
23     PARAM_DEFS = {
24         'drilldown_level': {
25             'type': 'string',
26             'description': 'How heavy the drilldown should be.',
27             'default': 'minor',
28             'required': True,
29             'range': ['minor', 'moderate', 'full']
30         }
31     }
32     """Param defs"""
33
34     def __init__(self):
35         Probe.__init__(self)
36
37     def perform_request(self):
38         """
39         Perform the drilldown.
40         See https://github.com/geopython/OWSLib/blob/master/tests/doctests/wms\_GeoServerCapabilities.txt
41         """
42
43         wms = None
44
45         # 1. Test capabilities doc, parses
46         result = Result(True, 'Test Capabilities')
47         result.start()
48         try:
49             wms = WebMapService(self._resource.url)
50             title = wms.identification.title
51             self.log('response: title=%s' % title)
52         except Exception as err:
53             result.set(False, str(err))
54
55         result.stop()
56         self.result.add_result(result)
57
58         # 2. Test layers

```

```

59     # TODO: use parameters to work on less/more drilling
60     # "full" could be all layers.
61     result = Result(True, 'Test Layers')
62     result.start()
63     try:
64         # Pick a random layer
65         layer_name = random.sample(wms.contents.keys(), 1)[0]
66         layer = wms[layer_name]
67
68         # TODO Only use EPSG:4326, later random CRS
69         if 'EPSG:4326' in layer.crsOptions \
70             and layer.boundingBoxWGS84:

```

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!

3.5.4 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',

    # 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': {
        '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'
    },
    '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.

3.5.5 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**.

3.5.6 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 are 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: *GeoHealthCheck.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 requests 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.

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

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_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: *GeoHealthCheck.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:

Results are helper-classes whose instances 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.HttpStatusNoError': {'default':
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, 'required': True, 'type': 'string', 'desc':
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, 'required': True, 'type': 'string', 'descr
    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 = {'version': {'default': '2.0.2', 'range': ['2.0.2'], 'required': True
    Param defs

class GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps
  Bases: GeoHealthCheck.probe.Probe

  Fetch OWS capabilities doc

  CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.ContainsStrings': {'default': T
    Checks avail for all specific Caps checks. Optionally override Check PARAM_DEFS using set_params
    e.g. with specific value.

  PARAM_DEFS = {'version': {'default': None, 'range': None, 'required': True, 'type'
    Param defs, to be specified in subclasses

class GeoHealthCheck.plugins.probe.owsgetcaps.SosGetCaps
  Bases: GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps

  SOS GetCapabilities Probe

  PARAM_DEFS = {'version': {'default': '1.0.0', 'range': ['1.0.0', '2.0.0'], 'require
    Param defs

class GeoHealthCheck.plugins.probe.owsgetcaps.WcsGetCaps
  Bases: GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps

  WCS GetCapabilities Probe

  PARAM_DEFS = {'version': {'default': '1.1.0', 'range': ['1.1.0', '1.1.1', '2.0.1'],
    Param defs

class GeoHealthCheck.plugins.probe.owsgetcaps.WfsGetCaps
  Bases: GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps

  WFS GetCapabilities Probe

  PARAM_DEFS = {'version': {'default': '1.1.0', 'range': ['1.0.0', '1.1.0', '2.0.2'],
    Param defs

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

  Fetch WMS capabilities doc

  PARAM_DEFS = {'version': {'default': '1.1.1', 'range': ['1.1.1', '1.3.0'], 'require
    Param defs

```

```
class GeoHealthCheck.plugins.probe.owsgetcaps.WmtsGetCaps
```

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

```
    WMTS GetCapabilities Probe
```

```
    PARAM_DEFS = {'version': {'default': '1.0.0', 'range': ['1.0.0'], 'required': True}
                  Param defs
```

```
class GeoHealthCheck.plugins.probe.owsgetcaps.WpsGetCaps
```

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

```
    WPS GetCapabilities Probe
```

```
    PARAM_DEFS = {'version': {'default': '1.0.0', 'range': ['1.0.0', '2.0.0'], 'required': True}
                  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.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 = {'layers': {'default': [], 'range': None, 'required': True, 'type': 'list'}
                  Param defs
```

```
    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
```

```
    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', 'range': ['minor', 'moderate']}
                  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}
                    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':
    Check for TMS GetTile

  PARAM_DEFS = {'y': {'default': '0', 'range': None, 'required': True, 'type': 'str'}
    Param defs

  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

  perform_request()
    Perform actual request to service, overridden from base class

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

  Probe for SensorThings API main endpoint url

  CHECKS_AVAIL = {'GeoHealthCheck.plugins.check.checks.HttpStatusNoError': {'default':
    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':
    Check for STA Get entity Collection

  PARAM_DEFS = {'entities': {'default': 'Things', 'range': ['Things', 'DataStreams',
    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':
    Checks for WFS GetFeature Response available. Optionally override Check PARAM_DEFS using
    set_params e.g. with specific value or even name.

  PARAM_DEFS = {'type_name': {'default': None, 'range': None, 'required': True, 'type':
    Param defs

  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

perform_request ()
Perform actual request to service, overridden from base class

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, 'range': None, 'required': True, 'type':
Param defs

class GeoHealthCheck.plugins.check.checks.HttpHasContentType

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

Checks if HTTP response has content type.

PARAM_DEFS = {'header_value': {'default': None, 'range': None, 'required': True, '
Params defs for header content type.

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_value': {'default': None, 'range': None, 'required': True, '
Param defs

class GeoHealthCheck.plugins.check.checks.HttpHasImageContentType

Bases: *GeoHealthCheck.check.Check*

Checks if HTTP response has image content type.

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.

class GeoHealthCheck.plugins.check.checks.NotContainsOwsException

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

Checks if HTTP response NOT contains given OWS Exceptions.

PARAM_DEFS = {'strings': {'description': 'The string text(s) that should be contained
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, 'range': None, 'required': True, 'type'  
    Param defs
```

```
class GeoHealthCheck.plugins.check.checks.XmlParse
```

```
    Bases: GeoHealthCheck.check.Check
```

Checks if HTTP response is valid XML.

3.6 License

The MIT License (MIT)

Copyright (c) 2014-2015 Tom Kralidis

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3.7 Contact

The website geohealthcheck.org is the main entry point.

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

3.7.1 Links

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

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

g

GeoHealthCheck.check, 28
GeoHealthCheck.plugin, 26
GeoHealthCheck.plugins.check.checks, 32
GeoHealthCheck.plugins.probe.http, 28
GeoHealthCheck.plugins.probe.owsgetcaps,
29
GeoHealthCheck.plugins.probe.sta, 31
GeoHealthCheck.plugins.probe.tms, 30
GeoHealthCheck.plugins.probe.wfs, 31
GeoHealthCheck.plugins.probe.wms, 30
GeoHealthCheck.plugins.probe.wmsdrilldown,
30
GeoHealthCheck.probe, 26
GeoHealthCheck.result, 28

A

after_request() (GeoHealthCheck.probe.Probe method), 27

AUTHOR (GeoHealthCheck.plugin.Plugin attribute), 26

B

before_request() (GeoHealthCheck.plugins.probe.tms.TmsGetTileAll method), 31

before_request() (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBboxAll method), 31

before_request() (GeoHealthCheck.plugins.probe.wms.WmsGetMapV1All method), 30

before_request() (GeoHealthCheck.probe.Probe method), 27

C

calc_result() (GeoHealthCheck.probe.Probe method), 27

Check (class in GeoHealthCheck.check), 28

CheckResult (class in GeoHealthCheck.result), 28

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.http.HttpGet attribute), 28

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps attribute), 29

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.sta.StaCaps attribute), 31

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.sta.StaGetEntities attribute), 31

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.tms.TmsCaps attribute), 30

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.tms.TmsGetTile attribute), 31

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox attribute), 31

CHECKS_AVAIL (GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 attribute), 30

CHECKS_AVAIL (GeoHealthCheck.probe.Probe attribute), 27

ContainsStrings (class in GeoHealthCheck.plugins.check.checks), 32

copy() (GeoHealthCheck.plugin.Plugin static method), 26

CopyGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps), 29

CopyGetMapV1All

D

DESCRIPTION (GeoHealthCheck.plugin.Plugin attribute), 26

E

expand_params() (GeoHealthCheck.probe.Probe method), 27

G

GeoHealthCheck.check (module), 28

GeoHealthCheck.plugin (module), 26

GeoHealthCheck.plugins.check.checks (module), 32

GeoHealthCheck.plugins.probe.http (module), 28

GeoHealthCheck.plugins.probe.owsgetcaps (module), 29

GeoHealthCheck.plugins.probe.sta (module), 31

GeoHealthCheck.plugins.probe.tms (module), 30

GeoHealthCheck.plugins.probe.wfs (module), 31

GeoHealthCheck.plugins.probe.wms (module), 30

GeoHealthCheck.plugins.probe.wmsdrilldown (module), 30

GeoHealthCheck.probe (module), 26

GeoHealthCheck.result (module), 28

get_default_parameter_values() (GeoHealthCheck.plugin.Plugin method), 26

get_metadata() (GeoHealthCheck.plugins.probe.tms.TmsGetTile method), 31

get_metadata() (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox method), 31

get_metadata() (GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 method), 30
 get_metadata() (GeoHealthCheck.probe.Probe method), 27
 get_metadata_cached() (GeoHealthCheck.probe.Probe method), 27
 get_param() (GeoHealthCheck.plugin.Plugin method), 26
 get_param_defs() (GeoHealthCheck.plugin.Plugin method), 26
 get_plugin_vars() (GeoHealthCheck.plugin.Plugin method), 26
 get_plugins() (GeoHealthCheck.plugin.Plugin static method), 26
 get_request_headers() (GeoHealthCheck.plugins.probe.http.HttpPost method), 29
 get_var_names() (GeoHealthCheck.plugin.Plugin method), 26

H

HttpGet (class in GeoHealthCheck.plugins.probe.http), 28
 HttpGetQuery (class in GeoHealthCheck.plugins.probe.http), 28
 HttpHasContentType (class in GeoHealthCheck.plugins.check.checks), 32
 HttpHasHeaderValue (class in GeoHealthCheck.plugins.check.checks), 32
 HttpHasImageContentType (class in GeoHealthCheck.plugins.check.checks), 32
 HttpPost (class in GeoHealthCheck.plugins.probe.http), 28
 HttpStatusNoError (class in GeoHealthCheck.plugins.check.checks), 32

I

init() (GeoHealthCheck.check.Check method), 28
 init() (GeoHealthCheck.probe.Probe method), 27

J

JsonParse (class in GeoHealthCheck.plugins.check.checks), 32

M

merge() (GeoHealthCheck.plugin.Plugin static method), 26
 METADATA_CACHE (GeoHealthCheck.probe.Probe attribute), 27

N

NAME (GeoHealthCheck.plugin.Plugin attribute), 26
 NotContainsOwsException (class in GeoHealthCheck.plugins.check.checks), 32

NotContainsStrings (class in GeoHealthCheck.plugins.check.checks), 32

O

OwsGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps), 29

P

PARAM_DEFS (GeoHealthCheck.plugin.Plugin attribute), 26
 PARAM_DEFS (GeoHealthCheck.plugins.check.checks.ContainsStrings attribute), 32
 PARAM_DEFS (GeoHealthCheck.plugins.check.checks.HttpHasContentType attribute), 32
 PARAM_DEFS (GeoHealthCheck.plugins.check.checks.HttpHasHeaderValue attribute), 32
 PARAM_DEFS (GeoHealthCheck.plugins.check.checks.NotContainsOwsException attribute), 32
 PARAM_DEFS (GeoHealthCheck.plugins.check.checks.NotContainsString attribute), 33
 PARAM_DEFS (GeoHealthCheck.plugins.probe.http.HttpGetQuery attribute), 28
 PARAM_DEFS (GeoHealthCheck.plugins.probe.http.HttpPost attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.CswGetCaps attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.OwsGetCaps attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.SosGetCaps attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.WcsGetCaps attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.WfsGetCaps attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.WmsGetCaps attribute), 29
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.WmtsGetCaps attribute), 30
 PARAM_DEFS (GeoHealthCheck.plugins.probe.owsgetcaps.WpsGetCaps attribute), 30
 PARAM_DEFS (GeoHealthCheck.plugins.probe.sta.StaGetEntities attribute), 31
 PARAM_DEFS (GeoHealthCheck.plugins.probe.tms.TmsGetTile attribute), 31
 PARAM_DEFS (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox attribute), 31
 PARAM_DEFS (GeoHealthCheck.plugins.probe.wms.WmsGetMapV1 attribute), 30
 PARAM_DEFS (GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown attribute), 30
 PARAM_DEFS (GeoHealthCheck.probe.Probe attribute), 27
 perform() (GeoHealthCheck.check.Check method), 28

- perform() (GeoHealthCheck.plugins.check.checks.HttpStatusNoError method), 29
- perform_request() (GeoHealthCheck.plugins.probe.owsgetcaps, HealthCheck.plugins.probe.tms.TmsGetTileAll method), 31
- perform_request() (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBbox method), 32
- perform_request() (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBboxAll method), 31
- perform_request() (GeoHealthCheck.plugins.probe.wfs.WfsGetFeatureBboxAll method), 31
- perform_request() (GeoHealthCheck.plugins.probe.wms.WmsGetMapV1All method), 30
- perform_request() (GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown method), 30
- perform_request() (GeoHealthCheck.plugins.probe.wmsdrilldown.WmsDrilldown method), 29
- perform_request() (GeoHealthCheck.probe.Probe method), 27
- Plugin (class in GeoHealthCheck.plugin), 26
- Probe (class in GeoHealthCheck.probe), 26
- ProbeResult (class in GeoHealthCheck.result), 28
- ## R
- REQUEST_HEADERS (GeoHealthCheck.probe.Probe attribute), 27
- REQUEST_METHOD (GeoHealthCheck.probe.Probe attribute), 27
- REQUEST_TEMPLATE (GeoHealthCheck.probe.Probe attribute), 27
- RESOURCE_TYPE (GeoHealthCheck.probe.Probe attribute), 27
- ResourceResult (class in GeoHealthCheck.result), 28
- Result (class in GeoHealthCheck.result), 28
- run() (GeoHealthCheck.probe.Probe static method), 27
- run_checks() (GeoHealthCheck.probe.Probe method), 27
- run_request() (GeoHealthCheck.probe.Probe method), 28
- ## S
- SosGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps), 29
- StaCaps (class in GeoHealthCheck.plugins.probe.sta), 31
- StaGetEntities (class in GeoHealthCheck.plugins.probe.sta), 31
- ## T
- TmsCaps (class in GeoHealthCheck.plugins.probe.tms), 30
- TmsGetTile (class in GeoHealthCheck.plugins.probe.tms), 30
- TmsGetTileAll (class in GeoHealthCheck.plugins.probe.tms), 31
- ## W
- WcsGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps),
- WfsGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps),
- WfsGetFeatureBbox (class in GeoHealthCheck.plugins.probe.wfs), 31
- WfsGetFeatureBboxAll (class in GeoHealthCheck.plugins.probe.wfs), 31
- WmsDrilldown (class in GeoHealthCheck.plugins.probe.wmsdrilldown), 30
- WmsGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps), 29
- WmsGetMapV1 (class in GeoHealthCheck.plugins.probe.wms), 30
- WmsGetMapV1All (class in GeoHealthCheck.plugins.probe.wms), 30
- WmtsGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps), 29
- WpsGetCaps (class in GeoHealthCheck.plugins.probe.owsgetcaps), 30
- ## X
- XmlParse (class in GeoHealthCheck.plugins.check.checks), 33