GPRE Documentation

Release 1.0

Jappe Franke, Allard de Wit, Wouter Meijninger

Sep 04, 2023

CONTENTS

The Golden Paddy Recommendation Engine (GPRE) has been developed as part of Smart Agriculture Myanmar project (SAM) which is funded by G4AW facility of the Dutch ministry of foreign affairs.

GPRE uses information on weather, location, crop type and sowing date for providing effective, time- and location- and crop stage specific advisory services to smallholder farmers in Myanmar. It is aimed at improving agricultural productivity and farmer income and at improving the management of weather related emergencies. Currently GPRE provides several services for Myanmar:

- Weather related services aimed at understanding current and forecasted weather conditions.
- Disease related services (rice only) for understanding impact of disease pressure on rice cultivation
- Prediction of crop stages based on location, weather and sowing date.

The Golden Paddy Recommendation Engine has been developed by Wageningen Environmental Research (WENR) together with with ImpactTerra. Embedding of results and visualization in an interface has been implemented by Satelligence.

CHAPTER

ONE

REFERENCE GUIDE

1.1 GPRE overview

1.1.1 Service components

The overall goal of the Golden Paddy Recommendation Engine (GPRE) is to support farmers and analysts in understanding the the impact of weather conditions on crop development, disease and extreme weather. GPRE provides several agronomic services for this purpose:

- **Weather related services**
  - maps of forecasted weather variables for all regions in Myanmar based on the forecast from DarkSky (7 days ahead)
  - charts of weather variables for all regions in Myanmar. Charts include the data from the current year, previous year, the climatology and the weather forecast.
- **Disease related services (rice only):**
  - Maps of susceptibility for 5 common rice diseases plus an indicator showing current conditions compared to the climatology (alerts)
  - Charts of susceptibility for 5 common rice diseases including the current year, forecast and the climatology.
- **Prediction of crop stages:**
  - Phenological stages are predicted according to the BBCH scale for four crops and several varieties (maize, sugar cane, mungbean and rice) based on location, weather and sowing date. If crop management messages are provided for the given crop and variety, the system can also provide the dates on which those messages could be sent to the farmer given the relevant crop stage.

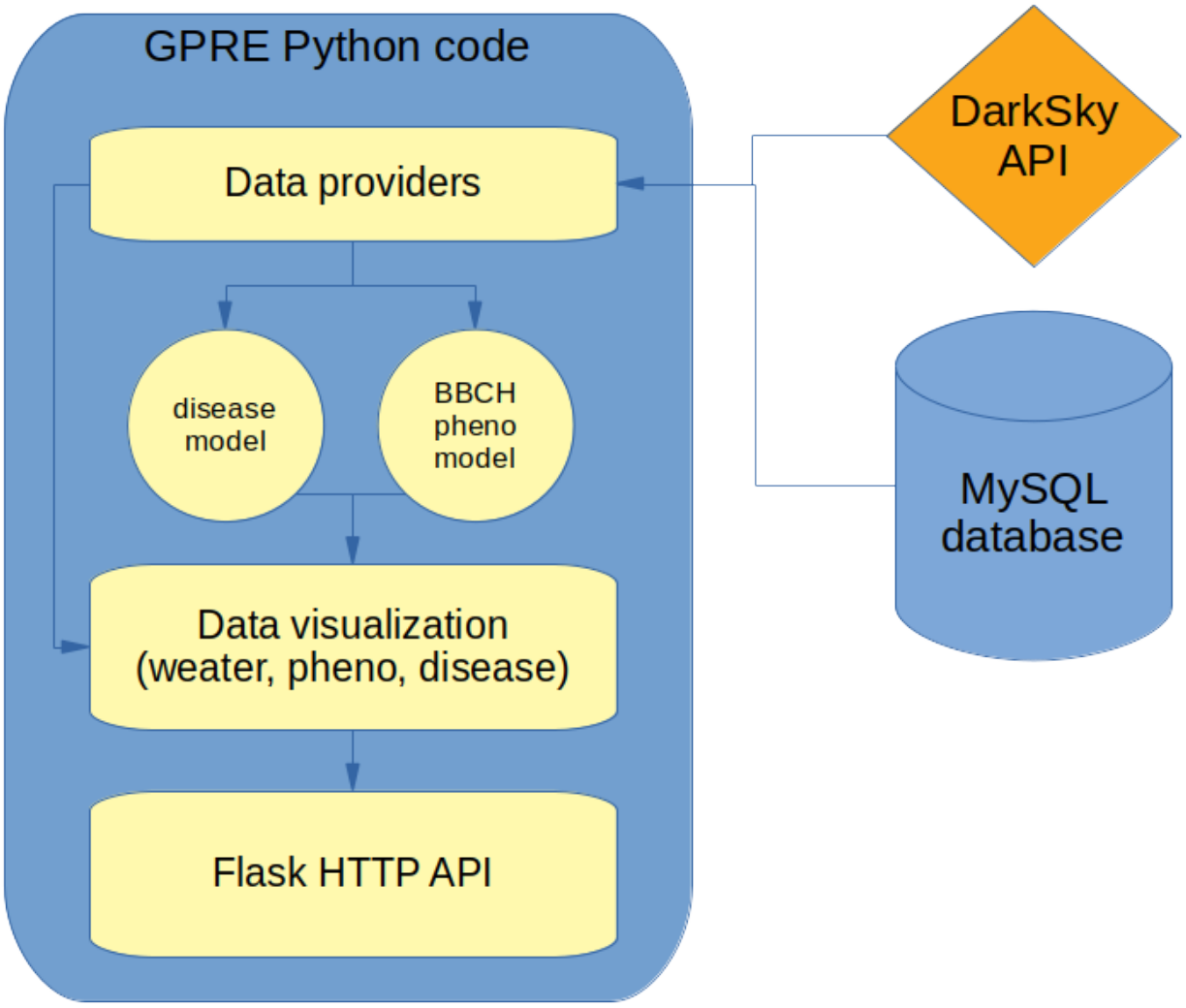
This documentation provides an overview of the current implementation of GPRE including the data sources used, modelling approach for phenology and disease and technical information related to deployment, debugging and extending GPRE with new crops.

1.1.2 Technical components

The main components of GPRE are:

- Modules written in python, which provide
  - Data providers for weather data (DarkSky forecast as well as historical weather data and the climatology)
  - Data providers for crop related data.
  - Visualization of results in HTML charts.
  - HTTP API's based on Flask that be called with parameters and return the results as JSON.
  - Execution of tasks (computing of cached results for the weather maps and disease maps).
- A MySQL database which contains:
  - Crop and variety specific characteristics/parameters required for the crop model
  - Messages related to crop management.
  - Historical weather data and the corresponding climatology based on ERA-INTERIM
  - Cached results for weather and disease maps
- Simulation models written in python which provide:
  - Prediction of crop stages and related messages
  - Prediction of disease susceptibility

The figure below provides a graphical overview of the main GPRE components.



1.2 Data sources

1.2.1 Weather

DarkSky API

GPRE uses the API provided by DarkSky to retrieve the weather forecast for the upcoming 7 days consisting of daily weather variables. Most of the variables provided by the DarkSky API are not relevant for agronomic applications (moonphase, windbearing, etc) therefore only a small subset of the variables is used:

- dewpointTemperature in degrees C;
- temperatureHigh in degrees C;
- temperatureLow in degrees C;
- windSpeed in m/sec;
- precipIntensity in mm/hour;

Examples of the API call can be found in the table below. Extensive information is available from the documentation for the DarkSky API

Next to a forecast API, DarkSky also provides a TimeMachine API which can be used to retrieve the weather variables for a day in the past. During development of GPRE several problems were encountered with the TimeMachine API:

- Each TimeMachine call only provides the data for one day. This makes querying a time-series relatively expensive and slow.
- For Myanmar the TimeMachine API provides weather data only for one year in the past which is insufficient for model calibration and computation of the climatology.

Therefore GPRE does not rely on DarkSky for historical data but instead uses the ERA-INTERIM archive.

API type	units	example
Forecast API	SI	ForecastApi
TimeMachine API	SI	TimeMachineAPI1
TimeMachine API + timezone and ISO time string	SI	TimeMachineAPI2

Note that datetimes in the DarkSky JSON response are presented as UNIX time stamps in the local timezone. This can be converted to timezone aware date/time in python using:

```
>>> import datetime as dt
>>> import pytz
>>> DarkSky_timestamp = 1592988827
>>> timezone = pytz.timezone("Asia/Yangon")
>>> tz_aware_datetime = dt.datetime.fromtimestamp(DarkSky_timestamp, timezone)
>>> print(tz_aware_datetime)
datetime.datetime(2020, 6, 24, 15, 23, 47, tzinfo=<DstTzInfo 'Asia/Yangon' +0630+6:30:00␣
↪STD>)
```

ECMWF ERA-INTERIM

ERA-INTERIM is a reanalysis of the global atmosphere since 1989, continuing in real time (Berrisford et al. 2009). The ERA-INTERIM atmospheric model and reanalysis system has a spatial resolution of 0.7°×0.7° and 60 atmospheric layers. Due to an improved reanalysis system, performance of ERA-INTERIM has improved compared to previous reanalysis data sets such as ERA-40 (ECMWF 2007).

Currently ERA-INTERIM is superseded by the latest reanalysis "ERA5" and its agriculturally enhanced version AgERA5. ERA5 and AgERA5 were not yet available at the start of development of GPRE, but hopefully the ERA-INTERIM database currently used for GPRE can be replaced by AgERA5 before the end of the project.

The ERA-INTERIM version used for GPRE has been downscaled to a 0.25x0.25 (~25km) degree grid and is available in the GPRE database since 2000. Also the climatology used by GPRE is based upon the ERA-INTERIM archive.

Combining weather data sources

Combining time-series of weather variables from different data sources can sometimes be tricky due to systematic differences between the data sources. Particularly for weather variables derived from weather models differences in temperature may exist due to differences in the estimated elevation of the land surface at a given point. Also differences in other variables can exist due to different model physics or boundary conditions.

Some differences can be corrected for, particularly temperature differences can often be corrected using so-called "lapse-rate" corrections. However, this implies that the target elevation of both data source at that particular location is known. However, DarkSky does not provide an elevation for its target location for both its forecast response and its TimeMachine response. Therefore, such corrections are not possible using data from DarkSky.

When looking at graphs or maps that combine weather data from the archive and the forecast it is therefore useful to bear in mind that systematic offsets may be present. This can be different for different location depending on how well the datasets are consistent for that location

1.2.2 Phenology modelling

For setting up the phenological model two main sources of inputs are required:

- Information on the phenological response of the crop to temperature and possibly day length. These parameters are assumed to be relatively invariant across varieties for a given crop type. Examples are the base and cutoff temperatures for phenological development.
- Information on the duration of the different growth stages of a particular crop. This information is assumed to be highly variable among varieties of a given crop.

For deriving these inputs different sources have been used. For the first category, the parameter values have been derived from existing crop simulation models and literature review on phenological development. For example the parameter files for the WOFOST crop simulation model provide information about the temperature response function of maize, rice, mungbean and sugarcane.

For calibrating the duration of the phenological stages for different crops and varieties, field data has been collected on the typical duration of crop varieties in Myanmar. Based on that information the duration of crop stages has been calibrated interactively using a workflow implemented in Jupyter notebooks.

1.2.3 Disease modelling

The disease modelling is based on the disease environmental response functions for age, temperature and humidity from the EpiRice model. Note that the disease modelling does not implement the entire EpiRice model, but only the environmental response functions which are used to compute the susceptibility of the plant for the disease given the conditions. Those environmental response functions are not described in the paper but can be obtained from the R implementation of EpiRice.

1.3 Service Components

1.3.1 Weather Monitoring service

The purpose of the weather monitoring service is to provide insight into the current season weather and the weather forecast for the coming week. Based on that information, an analyst can decide on actions that farmers need to take to succesfully cultivate their fields. Such actions can be sowing, weeding, fertilizer application, irrigation, harvesting and nearly any other crop management action that is determined by the weather conditions.

For many of such advisories on crop management it is often insufficient to only know the weather forecast. It is just as important to know the conditions over the past weeks as well as an impression of how "normal" the current season is. Therefore, the weather service combines data from three different sources:

- A weather forecast for the coming 7 days;
- The historical weather since the start of the current season;
- the long term average weather conditions (e.g. the 'climatology')

Based on this information charts can be generated for any location in Myanmar that put the current year weather conditions and the weather forecast into a climatological perspective. Further, the weather monitoring service also provides output for creating maps of the forecasted rainfall, temperature and wind for entire Myanmar as well as maps showing these values relative to the climatology (the so-called *z-score*). The latter can be used for alerting analysts on possible extreme conditions that could endanger the farmer and his/her crops.

Weather charts

An example of a weather chart for maximum temperature is provided below. It shows the current season weather as the red line and the forecast for the coming 7 days as the red line with the circular markers. Further it also provides the maximum temperature for the previous years for this location. Often weather can only be properly understood with reference to the long term average and the extremes. Therefore the historical mean is also provide as the black line, the mean +/- one standard deviation as the dark grey areas and the minimum and maximum values recorded in the past as the light grey areas.

Using this approach it is easy to see that there was a strong drop in the maximum temperature at the end of april with values that are lower than the minimum from the climatology. Fortunately, this drop in temperature is not as severe in the minimum temperature and is still well with the normal range. Therefore damage to crops are not to be expected from this event.

Example of a weather chart for maximum temperature.

Example of a weather chart for minimum temperature.

The chart for rainfall are slightly different from the charts for temperature and windspeed due to the chaotic nature of rainfall and the skewed distribution. Therefore, the charts do not show the standard deviations as a background layer but only the current year, the forecast, the historical mean and the minimum and maximum values in the climatology.

Example of a weather chart for rainfall.

Note: The weather charts have been implemented as HTML figures and therefore allow zooming and panning as well as exporting to a PNG file for use in reports and presentations. Further, data points are shown when hovering above the chart. For the figures on the temperature and wind these include the current year, the forecast, the previous year and the long term average. The chart for rainfall will only show the the current year, the forecast and the long term average. The hover labels for the current year and the forecast will also show the day-of-year next to the value itself.

Weather maps

The GPRE services do not generate weather maps themselves but rather provide the data to generate those maps at the regional level. The weather maps are based on the DarkSky forecast for the coming 7 days (daily values) which are provided for each level 3 region in Myanmar as stored in the *regions* table in the MySQL database. Values are returned based on the *GID_3* code for which the first 5 regions are listed in the table below.

GID_3	NAME_3	TYPE_3	longitude	latitude
MMR.1.1.1_1	Bassein West	Village Township	94.635	16.904
MMR.1.1.2_1	Kyaunggon	Village Township	95.119	17.101
MMR.1.1.3_1	Kyonpyaw	Village Township	95.169	17.298
MMR.1.1.4_1	Ngaputaw	Village Township	94.513	16.415
MMR.1.1.5_1	Thabaung	Village Township	94.746	17.174

For each region a call to DarkSky is made based on the given longitude/latitude. The forecast retrieved from DarkSky is processed in order to derive the indicators in the table below for each region.

Variable name	Explanation
rainfall	Rainfall expected in mm
rainfall_class	Classification of rainfall into class labels: "No rainfall [0-1]", "Light rainfall [1-5]", "Medium rainfall [5-10]", "Heavy rainfall [10-25]", "Very heavy rainfall [25-50]", "Extreme rainfall [>50]"
minimum_temp	Daily minimum temperature in degrees C
minimum_temp	Minimum temperature scored according to the historical distribution: 0 means equal or lower then the minimum historically reported for this variable. 1 means equal or larger then the the maximum historically reported for this variable. Values between 0 and 1 indicate a the position within the historical distribution.
maximum_temp	Daily maximum temperature in degrees C
maximum_temp	See minimum temperature
windspeed	Daily mean windspeed in meter/sec.
windgust	Maximum daily windgust in meter/sec.
windgust_class	Classification of wind gust into 5 classes: "Low wind", "medium wind", "strong wind gusts (some difficulties)", "Heavy wind gusts (caution)", "Extreme wind gusts (dangerous)"
vapour_pre	Daily mean vapour pressure

1.3.2 Crop stage prediction service

Phenology models predict time of events in an organism's development. Development of many organisms which cannot internally regulate their own temperature, is dependent on temperatures to which they are exposed in the environment. Plants and invertebrates, including insects and nematodes, require a certain amount of heat to develop from one point in their life-cycle to another, e.g., from eggs to adults. Because of yearly variations in weather, calendar dates are not a good basis for making management decisions. Measuring the amount of heat accumulated over time provides a physiological time scale that is biologically more accurate than calendar days.

The amount of heat needed by an organism to develop is known as physiological time. The amount of heat required to complete a given organism's development does not vary a lot– the combination of temperature (between thresholds) and time will always be similar. Physiological time is often expressed in units called degree-days. For instance: if a

species has a lower developmental threshold of 10° C, and the daily average temperature is at 13°C (or 3° above the lower developmental threshold), 3 degree-day is accumulated.

Each stage of an organism's development has its own total heat requirement. Development can be estimated by accumulating degree-days between temperature thresholds throughout the season. The accumulation of degree-days from a starting point can help predict when a developmental stage will be reached. Since many agro management actions that farmers have to take are connected to phenological stages of crop, monitoring and predicting phenological development through growing degree-days can support farmers when to take certain actions and helps planning such activities.

The GPRE crop stage prediction service uses a phenological model to simulate the crop development stages since sowing. The applied model is:

- able to estimate the timing of different phenological stages (BBCH), such as emergence, vining and flowering, and the duration of phenological phases, such as the grain filling phase.
- is based on the response of plant to surrounding temperature (air temperature), where each crop has a specific temperature range represented by a minimum, maximum, and optimum.

BBCH

The BBCH-scale is used to identify the phenological development stages of plants[1]. BBCH-scales have been developed for a range of crop species where similar growth stages of each plant are given the same code. The two figures below show the BBCH phenology scale for wheat and potato. See wikipedia page on the BBCH-scale for a complete list of crops and their corresponding phenology BBCH scale.

(see figures below).

Plant response

How a crop responds to the surrounding temperatures varies per crop. Temperature is the most important among all factors that influence rate of plant development. For the GPRE crop stage prediction service we refer to temperature as the daily average temperature. In general the response curve can be represented by minimum, maximum and optimum temperatures. Where the minimum (or base) and maximum temperatures that define limits of development, and the optimum temperature at which the development rate is maximal. All together these temperatures are also known as the cardinal temperatures. The figure below shows different types of response curves.

When the temperature response function (TRF) is zero, development does not take place, which occurs when temperature is below the minimum or above the maximum temperature. Development takes place at the maximum rate if the response function is one, which occurs when temperature is between the optimal cardinal temperature. The relative development rate is computed using linear interpolation along the temperature response function using the daily mean temperature as its input value on the X axis.

The daily accumulation of growing degree-days (*GDD*) is thus computed as:

$$GDD = GDD_{max} \cdot TRF(T_{avg})$$

where the maximum growing degree-days (GDD_{max}) is computed as the optimum cardinal temperature (T_{opt1}) minus the minimum or base temperature (T_{min})

The development stage DVS_t of the crop at a given day *t* can now be computed by accumulating *GDD*:

$$DVS_t = DVS_{t-1} + GDD_t$$

[1] BBCH (Biologische Bundesanstalt, Bundessortenamt und CHemische Industrie) - scale: is a scale used to identify the phenological development stages of a plant. A series of BBCH-scales have been developed for a range of crop species. Phenological development stages of plants are used in a number of scientific disciplines (crop physiology, phytopathology, entomology and plant breeding) and in the agriculture industry (timing of pesticide application, fertization, agricultural insurance). The BBCH-scale uses a decimal code system, which is divided into principal and secondary growth stages, and is based on the cereal code system (Zadoks scale) developed by Zadoks.

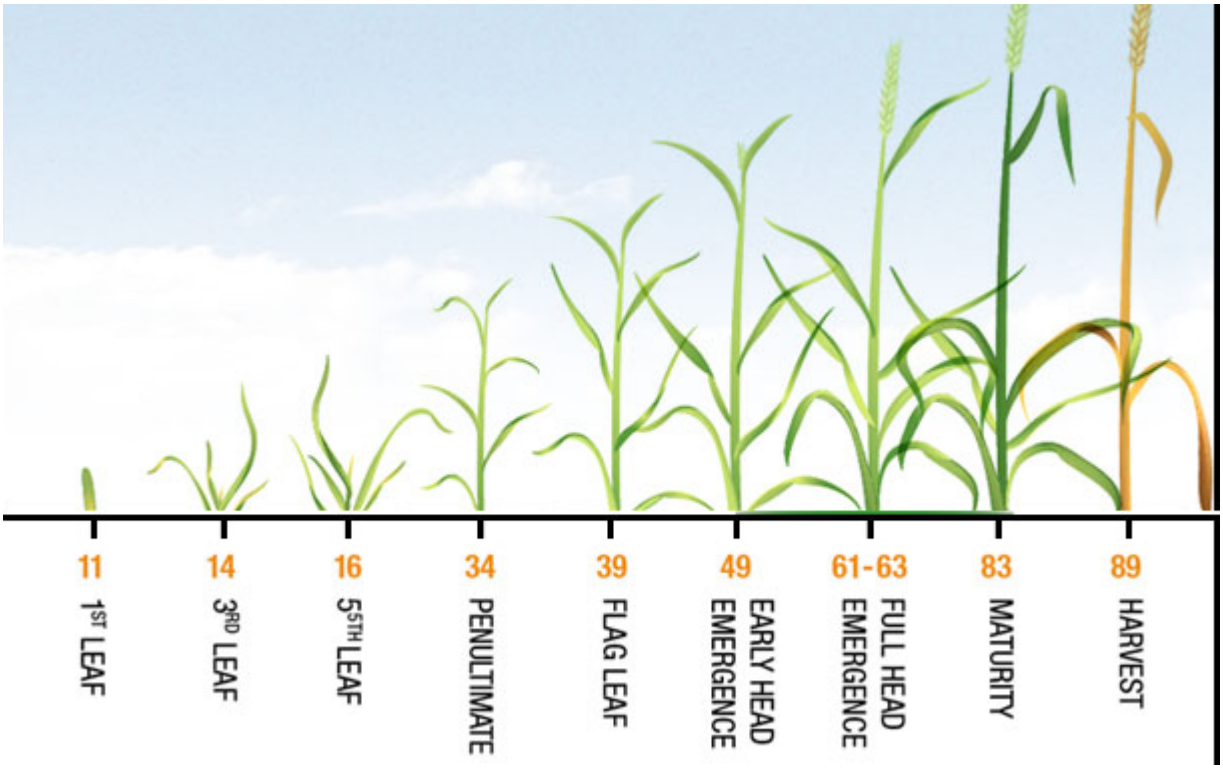


Fig. 1: BBCH stages for wheat.

The specific BBCH stages for a crop can now be connected to specific DVS values and the date on which this stages occurs can be predicted using meteorological data using either historical records, a weather forecast or a climatology.

The crop stage prediction service is implemented in a python module and uses the simulation engine provided by PCSE to make the actual computations. The service is exposed through a web interface and delivers a JSON response containing the predicted crop growth plus alerts for crop treatments and possible weather induced stress:

More information on PCSE can be found @ http://pcse.readthedocs.io/en/stable

1.3.3 Disease monitoring service

The disease monitoring service is currently only operational for rice. It is based on the EpiRice model which defines environmental response functions for a number of important rice disease based on temperature and relative humidity. The disease monitoring service does not fully implement the epirice model nor does it predict the actual disease infestation on a particular location. It merely predicts the susceptibility of the crop to a particular disease given the prevailing weather conditions.

The current implementation provides both charts of the time-course of disease susceptibility for a given location as well as data for generating maps of disease susceptibility . Not that the maps are generated based on the environmental response curves for the whole of Myanmar but they take no notice of the actual rice growing areas. Therefore, alerts could be generated for areas that are not important for rice cultivation.

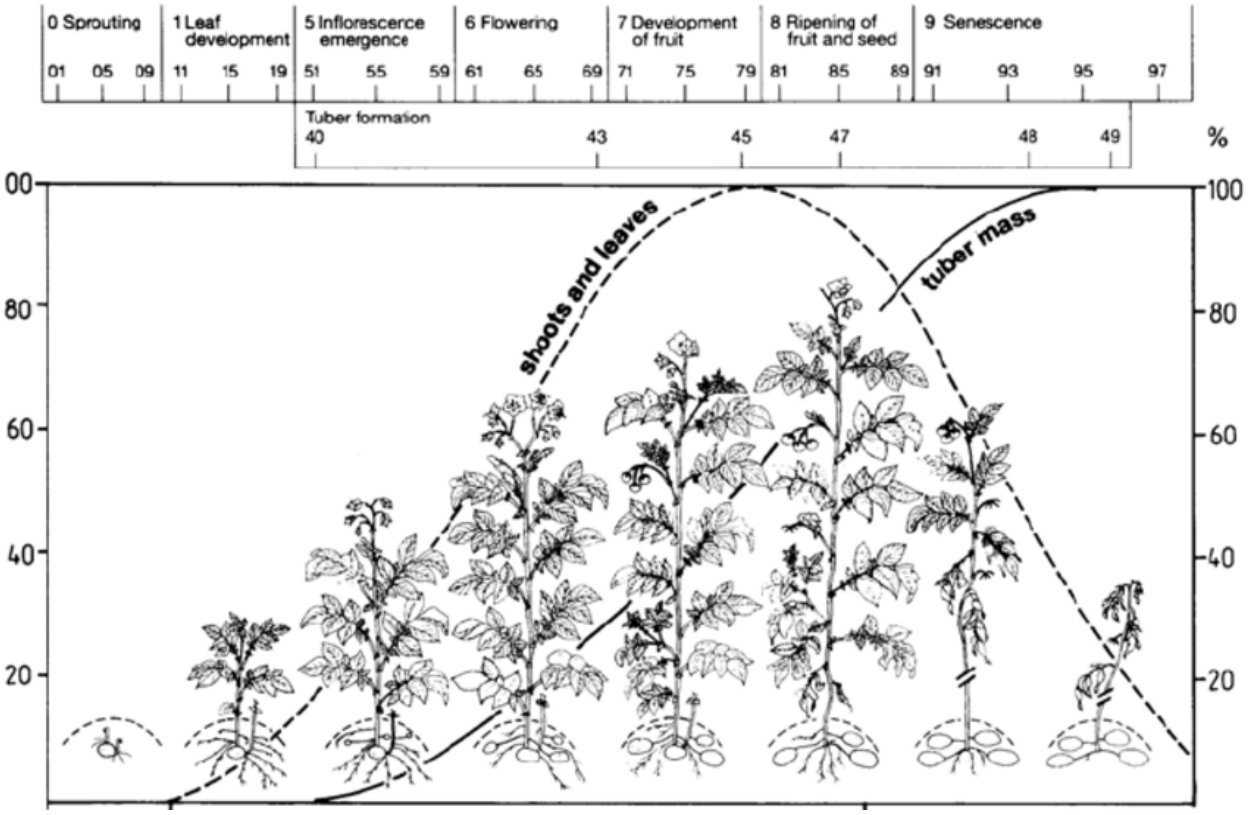


Fig. 2: BBCH stages for potato.

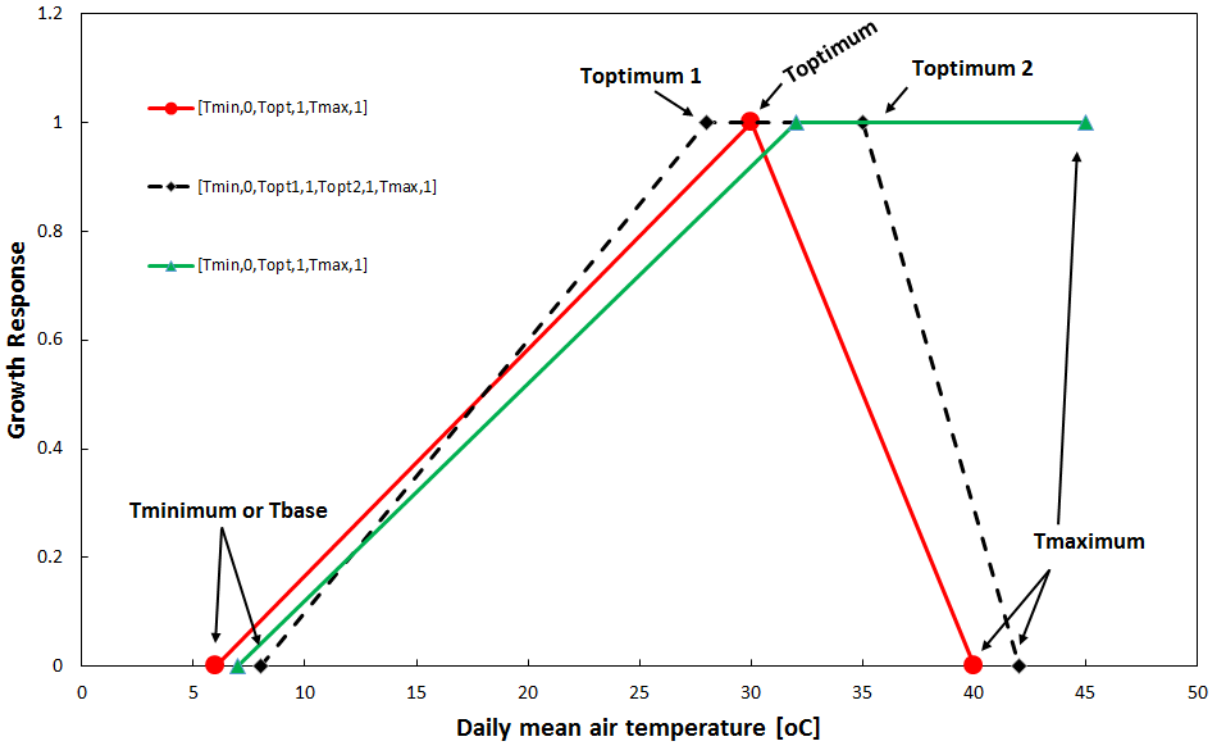


Fig. 3: Temperature response function - various forms.

Disease charts

The charts for disease susceptibility are very similar to the weather monitoring. They provide insight into the course of disease susceptibility for the current year (the period from sowing date up till today), the forecast (the next 7 days) and the variability according to the historical archive. To show the variability, the plot includes, the minimum/maximum, the 10th/90th percentile and the median of the historical data.

Using this approach it demonstrates that disease susceptibility has been mainly low for this season and is also expected to be low for the coming 7 days.

Susceptibility for leaf blast at lat/lon=20.9/96.5 at the 5th of July 2020 with a sowing date of 2020-04-07

Disease maps

Similar to the weather maps, the disease monitoring service does not provide ready made maps but instead returns the data to generate the maps. The services uses the same regions as the weather map service and provides the disease susceptibility value for all regions in Myanmar and for the 7 days of the DarkSky forecast. Further, the services provides a Z score that indicates how the current value relates to the historical record. The interpretation is the same as Z scores that are computed for the weather service: 0 means equal or lower then the minimum historically reported for this variable. 1 means equal or larger then the the maximum historically reported for this variable. Values between 0 and 1 indicate a the position within the historical distribution. The variables provided by the disease mapping service are provided in the table below.

Variable name	Explanation
F_WEATHER	The disease susceptibility value for the weather conditions on this day.
F_WEATHER_SCORE	The Z score for the current disease susceptibility

1.4 GPRE system and installation

1.4.1 Overview of the repository

The list below shows an overview of the directory layout of the GPRE code and a description of the content of the different sub-directories:

```
└── GPRE                              - Root directory and various scripts for docker and␣
↪flask
    ├── docker_base                   - Scripts for creating the base docker image;
    └── gpre                          - the GPRE code root directory
        ├── cache                     - cache and temporary files
        ├── config                    - System configuration and settings
        ├── db_struct                 - SQL scripts for building the MySQL database
        ├── doc                       - Documentation for GPRE, built with Sphinx
        │   └── figures               - Figures for the documentation
        ├── downloads                 - File downloads are written here
        ├── gpre                      - Python code implementing GPRE services
        ├── logs                      - Log files are written here
        ├── notebooks                 - Notebooks, mainly for estimating GDD for␣
↪phenology for different crops
        ├── pcse                      - The PCSE package that is used by the models for␣
↪crop stages prediction and disease
```

(continues on next page)

(continued from previous page)

```
        ├── phenology                - The model for simulating phenological development
        ├── tasks                    - Python scripts that implement scheduled tasks
        └── webserver                - Python scripts that implement the HTTP API to␣
↪call GPRE services
```

1.4.2 System configuration

The configuration of GPRE has been implemented as a python package which allow to import the whole configuration and access the various configuration settings by typical python notation (e.g. `config.database.default_user`). The configuration has been split into sections with the following structure:

```
config/__init__.py    - Top level configuration (logging, site name, etc.)
    ├── database.py   - Database credentials to access the database
    ├── disease.py    - Disease related settings
    ├── simulator.py  - Settings related to phenology simulation
    ├── weather.py    - Settings related to weather (e.g. DarkSky API key)
    ├── webserver.py  - Web server related settings (IP address under which GPRE is␣
↪running)
```

Most of the configuration settings do not need to be touched. However if database credentials or IP addresses change this may be required.

1.4.3 Setting up GPRE on your local desktop

Often if is useful to run the GPRE services on your local desktop in order to debug a service, develop additional services or have a closer look at a certain output. However, every GPRE service needs to connect to the GPRE database in order to retrieve weather data and/or crop parameters. Therefore, installing GPRE locally involves several steps:

1. Installing a python distribution and the required packages
2. Installing the Google Cloud SQL proxy to connect to the MySQL database
3. Starting GPRE through python, flask or docker

Below, we assume that GPRE will be installed on a PC running Linux (Ubuntu or similar).

Setting up a python environment

The most convenient way to setup a python environment is through the miniconda python installer. After downloading and installing miniconda a new python environment must be created. Current GPRE still runs on a python 2.7 environment. Although python 2.7 is officially end-of-life this will not be problematic as long as the GPRE API is not directly exposed to the internet. A new python environment can be created with the following command:

```
conda create -n py27_gpre python=2.7  pymysql sqlalchemy pandas numpy pyyaml flask xlwt␣
↪xlrd requests jupyter sphinx
```

When the environment has been created, it can be activated with:

```
(base) wit015@d0137094:~$ conda activate py27_gpre
(py27_gpre) wit015@d0137094:~$
```

Some additional packages still need to be installed with the *pip* installer:

```
(py27_gpre) wit015@d0137094:~$ pip install yattag dotmap plotly==3.6
...
Successfully installed dotmap-1.3.17 typing-3.7.4.1 yattag-1.13.2 plotly-3.6.0
(py27_gpre) wit015@d0137094:~$
```

The conda environment is now ready for use on your local PC. Note that for the operational deployment of GPRE, installation of python and its dependencies for GPRE are taken care of by the docker image.

Connecting to the MySQL database

The MySQL database is running in the Google cloud as a managed database service. A database running within the Google cloud cannot be access directly from outside for security reasons. Therefore special software has to be used to make your local desktop PC connect to the GPRE database, this special software is the Google Cloud_SQL_proxy. The cloud SQL proxy creates a secure tunnel between your local desktop and the Google Cloud project where the database is running and forwards the requests to the database. Use of the Cloud SQL proxy is free of charge.

Installing the Cloud_SQL_proxy is simple as it is a self-contained binary, however it requires that the Google Cloud SDK is installed and that the Google project where the database is running is registered as the default project. With the Google SDK installed, starting the cloud proxy can be done using:

```
wit015@d0137094$ ./cloud_sql_prox -instances=gpci-266802:us-central1:gpre-mysql=tcp:3310
2020/07/07 12:01:16 Rlimits for file descriptors set to {&{8500 1048576}}
2020/07/07 12:01:16 Listening on 127.0.0.1:3310 for gpci-266802:us-central1:gpre-mysql
2020/07/07 12:01:16 Ready for new connections
```

The cloud proxy has now connected to the google cloud and created a tunnel to the GPRE MySQL database which listens at the localhost (127.0.0.1) at port 3310. The id string that is given after the *-instances* option can be taken from the DB administration page on the Google Cloud portal.

The actual connection can now be made using any MySQL client using the proper database username/password. In the example below we use the MySQL commandline utility:

```
wit015@d0137094$ mysql -u gpre -p -h 127.0.0.1 -P 3310
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 880049
Server version: 5.7.25-google-log (Google)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use gpre;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from crop;
+---------+-----------+
```

(continues on next page)

(continued from previous page)

```
| crop_no | crop_name |
+---------+-----------+
|       1 | Maize     |
|      11 | Rice      |
|      13 | Sugarcane |
|      14 | Mung bean |
+---------+-----------+
4 rows in set (0.11 sec)

mysql>
```

The Cloud SQL proxy will also show that a new connection has been made:

```
2020/07/07 12:05:27 New connection for "gpci-266802:us-central1:gpre-mysql"
```

Similarly the connection can be made from within python using the database credentials that are set in the GPRE configuration at *config/database.py*. However, it is important that the environment variable *DEVELOP* is set in order to indicate to GPRE that we are running in DEVELOP mode and not in the docker image on the Google Cloud. Note that we are adding the location where GPRE can be found to the python path which will vary depending on where you put GPRE on your system:

```
(py27_gpre) wit015@d0137094:~$ export DEVELOP=1
(py27_gpre) wit015@d0137094:~$ python
Python 2.7.18 |Anaconda, Inc.| (default, Apr 23 2020, 22:42:48)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path.append("/home/wit015/Projects/SAMM/GPRE/gpre")
>>> import config
using DEVELOP DB settings!
>>> import sqlalchemy as sa
>>> engine = sa.create_engine(config.database.dbc)
>>> DBcon = engine.connect()
>>> cursor = DBcon.execute("select * from crop")
>>> print(cursor.fetchall())
[(1, 'Maize'), (11, 'Rice'), (13, 'Sugarcane'), (14, 'Mung bean')]
```

We can now start to test GPRE on the local PC or run the notebooks for setting up a new crop for the crop prediction service. Note that the Cloud SQL proxy should be running whenever we want to connect to the MySQL GPRE database.

Running GPRE locally

For running GPRE from a local python interpreter examples can found in the folder *tasks/*. Running the script *tasks/gpre_examples.py* will produce the following output:

```
(py27_gpre) wit015@d0137094:~/Projects/SAMM/GPRE/gpre/tasks$ python gpre_examples.py
/home/wit015/Projects/SAMM/GPRE/gpre/tasks
using DEVELOP DB settings!
{
 "managementalerts": [
  {
   "msg": "Weed control",
```

(continues on next page)

(continued from previous page)


```
  "msg_id": "7",
  "day": "2019-08-28"
 },
 {
  "msg": "Scouting for pest and disease control",
  "msg_id": "12",
  "day": "2019-09-04"
 },
 {
  "msg": "Second fertilizer application",
  "msg_id": "5",
  "day": "2019-09-08"
 },
 {
  "msg": "Weed control",
  "msg_id": "8",
  "day": "2019-09-07"
 },
 {
  "msg": "Check leaf colour for nutrient deficiency",
  "msg_id": "10",
  "day": "2019-09-07"
 },
 {
  "msg": "Scouting for pest and disease control",
  "msg_id": "13",
  "day": "2019-09-16"
 },
 {
  "msg": "Third fertilizer application",
  "msg_id": "6",
  "day": "2019-10-11"
 },
 {
  "msg": "Weed control",
  "msg_id": "9",
  "day": "2019-10-10"
 },
 {
  "msg": "Check leaf colour for nutrient deficiency",
  "msg_id": "11",
  "day": "2019-10-10"
 },
 {
  "msg": "Scouting for pest and disease control",
  "msg_id": "14",
  "day": "2019-10-14"
 },
 {
  "msg": "Scouting for pest and disease control",
  "msg_id": "15",
  "day": "2019-10-31"
```


(continues on next page)

(continued from previous page)


```
  },
  {
   "msg": "Prepare for harvest",
   "msg_id": "16",
   "day": "2019-12-14"
  },
  {
   "msg": "Harvesting can begin: check weather forecast",
   "msg_id": "17",
   "day": "2019-12-19"
  },
  {
   "msg": "Thresh and dry maize kernels",
   "msg_id": "18",
   "day": "2019-12-23"
  },
  {
   "msg": "Arrange proper storage (sacks and warehouse). Check for pests",
   "msg_id": "19",
   "day": "2020-01-02"
  }
 ],
 "weatheralerts": [],
 "phenology": [
  {
   "bbch": "BBCH_01",
   "day_current": "2018-08-15T00:00:00",
   "dap_avg": 1,
   "dap_current": 0,
   "dap_diff": 1
  },
  {
   "bbch": "BBCH_10",
   "day_current": "2018-08-23T00:00:00",
   "dap_avg": 9,
   "dap_current": 8,
   "dap_diff": 1
  },
  {
   "bbch": "BBCH_13",
   "day_current": "2018-09-05T00:00:00",
   "dap_avg": 21,
   "dap_current": 21,
   "dap_diff": 0
  },
  {
   "bbch": "BBCH_30",
   "day_current": "2018-09-12T00:00:00",
   "dap_avg": 28,
   "dap_current": 28,
   "dap_diff": 0
  },
```


(continues on next page)

(continued from previous page)


```
  {
   "bbch": "BBCH_50",
   "day_current": "2018-10-14T00:00:00",
   "dap_avg": 61,
   "dap_current": 60,
   "dap_diff": 1
  },
  {
   "bbch": "BBCH_60",
   "day_current": "2018-10-18T00:00:00",
   "dap_avg": 64,
   "dap_current": 64,
   "dap_diff": 0
  },
  {
   "bbch": "BBCH_70",
   "day_current": "2018-11-02T00:00:00",
   "dap_avg": 78,
   "dap_current": 79,
   "dap_diff": -1
  },
  {
   "bbch": "BBCH_80",
   "day_current": "2018-11-26T00:00:00",
   "dap_avg": 98,
   "dap_current": 103,
   "dap_diff": -5
  },
  {
   "bbch": "BBCH_89",
   "day_current": "2018-12-23T00:00:00",
   "dap_avg": 127,
   "dap_current": 130,
   "dap_diff": -3
  },
  {
   "bbch": "BBCH_99",
   "day_current": "2018-12-29T00:00:00",
   "dap_avg": 132,
   "dap_current": 136,
   "dap_diff": -4
  }
 ]
}
```

Running GPRE using flask

The HTTP API for GPRE has been implemented using Flask. Flask is a micro-web development framework which makes it easy to build an HTTP interface on top of python code. For debugging the web interface and the GPRE services it is often required to start Flask and test the output of the code in your browser. For this purpose Flask has a built-in webserver that can be used for development but should not be used for production environments.

Starting Flask to run the GPRE services must be done from the GPRE root folder using the following commands:

```
(py27_gpre) wit015@d0137094:~/Projects/SAMM/GPRE$ export DEVELOP=1
(py27_gpre) wit015@d0137094:~/Projects/SAMM/GPRE$ python gpre/webserver/flask_app.py
using DEVELOP DB settings!
 * Serving Flask app "flask_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
[INFO] -  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
[INFO] -  * Restarting with stat
using DEVELOP DB settings!
[WARNING] -  * Debugger is active!
[INFO] -  * Debugger PIN: 182-242-591
```

This starts the flask webserver on the local host at port 5000. The GPRE services can now be reached at their HTTP address. For example the following URL generates the weather charts for the given latitude/longitude. Note that the charts are provided as HTML DIV strings and first have to be embedded in the proper HTML layout in order to be visualized.:

```
http://localhost:5000/api/v1/get_weather_charts?latitude=21&longitude=97
```

Running GPRE using docker

Building the GPRE Docker image

As a first step for deploying GPRE on the Google infrastructure it is required to build and test the Docker image that packages GPRE. First, Docker must be installed on the local PC. There is good documentation available for installing Docker on Ubuntu so we will not repeat that here. Instead we assume that docker is properly installed and the docker commands can be executed by an ordinary (non-root) user.

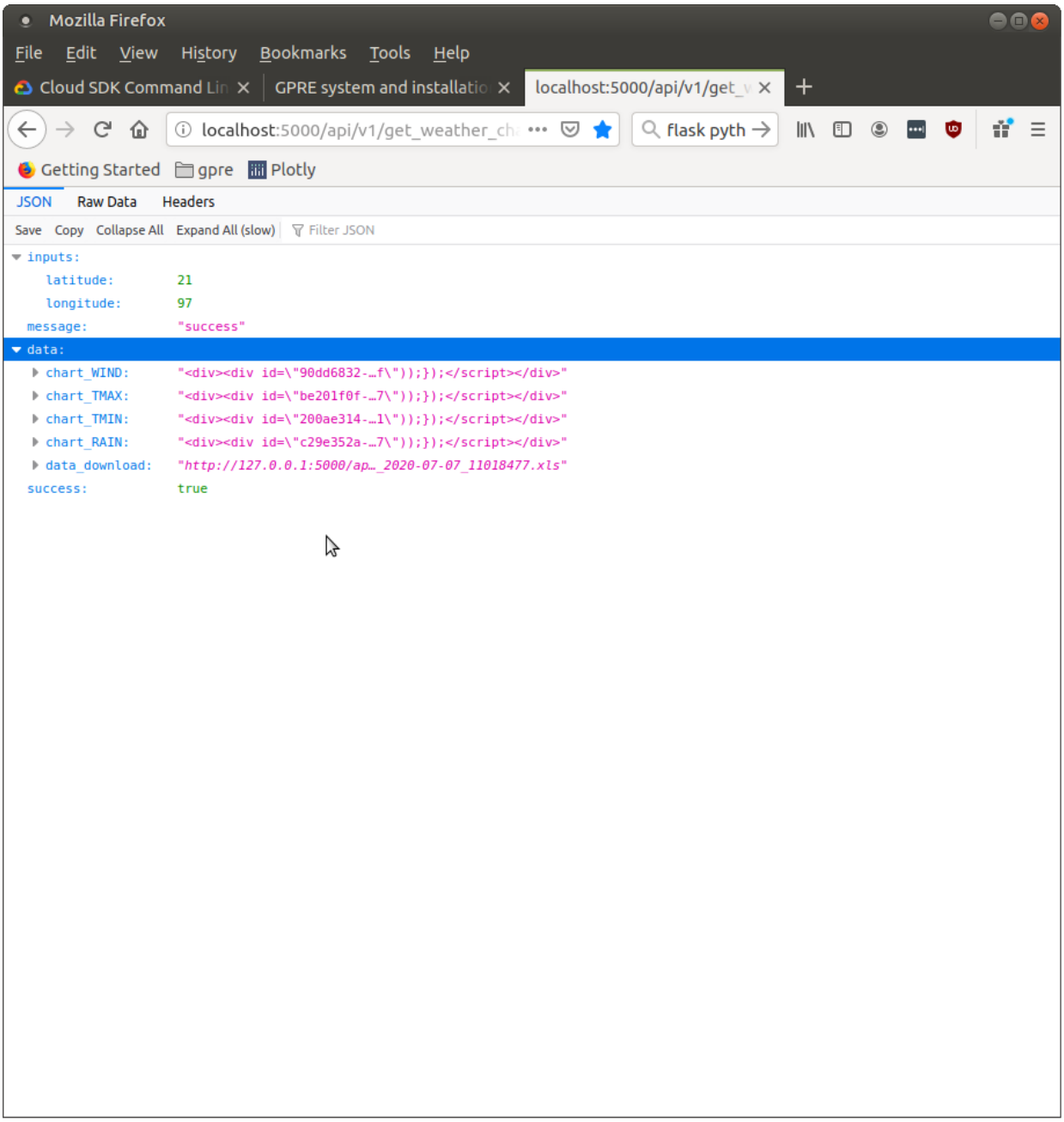
The first step in building the GPRE docker image is to build the base image. Within this image everything is prepared for GPRE but it does not yet include the GPRE code. The reason for creating a base image is that creating the base image is relatively time-consuming but only has to be done once. The actual GPRE services image will build upon the base image and will simply add the GPRE code in order to operationalize GPRE.

The GPRE base image itself is based on a third party image which already prepares a linux operating system based on Debian Linux 'Buster'. It includes a python2.7 installation, the NGINX webserver and the uWSGI framework that provides the connection between NGINX and Flask. Information about this image can be found here.

The definition of the base image is laid out in the *Dockerfile* in the *GPRE/docker_base* directory. The image can be built using the command (note the trailing dot):

```
docker build -t gpre/base:v2020.05.07 .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM tiangolo/uwsgi-nginx-flask:python2.7
 ---> 04d91d0c6044
```

(continues on next page)


Mozilla Firefox
File Edit View History Bookmarks Tools Help
Cloud SDK Command Lin
GPRE system and installatio
localhost:5000/api/v1/get_v
localhost:5000/api/v1/get_weather_ch
flask pyth
Getting Started
gpre
Plotly
JSON
Raw Data
Headers
Save Copy Collapse All Expand All (slow) Filter JSON
inputs:
latitude: 21
longitude: 97
message: "success"
data:
chart_WIND: " <div id=\"90dd6832-…f\"));});</script> "
chart_TMAX: " <div id=\"be201f0f-…7\"));});</script> "
chart_TMIN: " <div id=\"200ae314-…1\"));});</script> "
chart_RAIN: " <div id=\"c29e352a-…7\"));});</script> "
data_download: "http://127.0.0.1:5000/ap…_2020-07-07_11018477.xls"
success: true

(continued from previous page)

```
Step 2/3 : RUN apt-get update && apt-get install -y --no-install-recommends --no-upgrade␣
↪python-pip python-setuptools vim.tiny && rm -rf /var/lib/apt/lists/* && rm -rf /var/
↪cache/apk/*
 ---> Using cache
 ---> fae542ac8123
Step 3/3 : RUN pip install dotmap pymysql sqlalchemy pandas numpy pyyaml flask xlwt xlrd␣
↪requests yattag plotly==3.6
 ---> Using cache
 ---> 4fcda6056a8c
Successfully built 4fcda6056a8c
Successfully tagged gpre/base:v2020.05.07
```

The option *-t* tags the docker image with a name which includes the date that the Dockerfile was defined. Subsequent versions of the base image should get a new tag updating the date.

Building the GPRE image

Next step is to build the GPRE image. The Dockerfile for building the GPRE image is very simple. It takes the base image, creates a folder structure and finally copies the python code required for GPRE into the image. Note that the *FROM* directive in the Dockerfile should point to the latest version of the GPRE base image. The Dockerfile looks like this:

```
FROM gpre/base:v2020.05.07

RUN mkdir /app/gpre /app/gpre/cache /app/gpre/config /app/gpre/downloads/ /app/gpre/gpre␣
↪/app/gpre/logs/ \
 /app/gpre/pcse /app/gpre/phenology /app/gpre/tasks /app/gpre/webserver
COPY ./gpre/config /app/gpre/config
COPY ./gpre/gpre /app/gpre/gpre
COPY ./gpre/pcse /app/gpre/pcse
COPY ./gpre/phenology /app/gpre/phenology
COPY ./gpre/tasks /app/gpre/tasks
COPY ./gpre/webserver/flask_app.py /app/main.py
```

Finally the GPRE Docker image can be built (from within the GPRE top directory where the Dockerfile resides) with:

```
$ python -m compileall .
$ docker build -t gpre/v2020.05.13 .
```

When listing the available Docker images, it should now show at least the following three images:

```
$ docker image ls
REPOSITORY                                                          TAG                 ␣
↪IMAGE ID            CREATED             SIZE
gpre/v2020.05.13                                                    latest              ␣
↪a78b2bd16b53        8 minutes ago       1.29GB
gpre/base                                                           v2020.05.07         ␣
↪4fcda6056a8c        2 months ago        1.28GB
tiangolo/uwsgi-nginx-flask                                          python2.7           ␣
↪04d91d0c6044        8 months ago        910MB
```

Starting and testing the GPRE docker image

The GPRE docker image can now be started by Docker. This means that docker creates a container from the image which is then started. One can start multiple containers from one image which could be useful if the GPRE service

gets more requests than can be handled by one container. Starting an image is done with the *docker run* command:

```
docker run --name=gpre_production --network="host" -d -e DEVELOP='1' gpre/v2020.05.13
```

Because we are still running on the local desktop some special commands have been added. The *–network="host"* option indicates that the network of the container should not be isolated from the Docker host. This host networking mode is required because the container needs to connect to the MySQL database through the SQL cloud proxy which runs on the localhost port 3310. Further, the option *-e DEVELOP='1'* sets environment variable *DEVELOP* within the container because we still are running in DEVELOP mode.

We can check that the container is running with:

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             ␣
↪STATUS              PORTS               NAMES
144b7b30624b        gpre/v2020.05.13    "/entrypoint.sh /sta..."   21 minutes ago      ␣
↪Up 21 minutes                          gpre_production
```

Now the GPRE service should be available on the localhost port 80. In fact, we are running the same code as with the Flask webserver (see above) but now through the NGINX webserver inside a Docker container. This solution is much more portable and robust than the Flask solution (which is for testing only). The GPRE services can be accessed in your browser through the following URL:

> http://localhost/api/v1/get_weather_charts?latitude=21&longitude=97

Which should give the same result as the Flask solution (see above).

Debugging Docker containers

Debugging Docker containers can be notoriously difficult. The fact that a container is isolated from the host operating system also means that little feedback is provided on the what goes wrong when running the GPRE image. When the GPRE service fails often the only direct feedback is a *502 Bad Gateway* message displayed in your browser.

The most easy approach I found when debugging docker containers is by accessing the container through the *docker exec* command. For this we first need to find the container ID:

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             ␣
↪STATUS              PORTS               NAMES
b861c364f513        gpre/v2020.05.13    "/entrypoint.sh /sta..."   4 minutes ago       ␣
↪Up 4 minutes                           gpre_production
```

Next we can connect to the docker container by starting a *bash* shell inside the container with:

```
$ docker exec -it b861c364f513 bash
root@d0137094:/app#
```

The *root@d0137094:/app#* prompt indicates that you are now inside the container. Now the GPRE services can be started manually with:

```
root@d0137094:/app# python main.py
using DEVELOP DB settings!
Traceback (most recent call last):
  File "main.py", line 46, in <module>
    from gpre.create_weather_graph import generate_weather_charts_for_location
  File "/home/wit015/Projects/SAMM/GPRE/gpre/gpre/__init__.py", line 7, in <module>
  File "/home/wit015/Projects/SAMM/GPRE/gpre/config/__init__.py", line 19, in <module>
```

(continues on next page)

(continued from previous page)

```
  File "/home/wit015/Projects/SAMM/GPRE/gpre/config/simulator.py", line 9, in <module>
  File "/home/wit015/Projects/SAMM/GPRE/gpre/phenology/__init__.py", line 1, in <module>
  File "/home/wit015/Projects/SAMM/GPRE/gpre/phenology/data_providers.py", line 20, in
↪<module>
  File "/home/wit015/Projects/SAMM/GPRE/gpre/pcse/__init__.py", line 103, in <module>
  File "/home/wit015/Projects/SAMM/GPRE/gpre/pcse/__init__.py", line 91, in setup
IOError: [Errno 2] No such file or directory: '/app/gpre/pcse/settings/default_settings.
↪py'
root@d0137094:/app#
```

The output from the python interpreter now clearly indicates that a file is missing.

1.4.4 Deploying GPRE on the Google Cloud

First-time deployment of the container

When the Docker image can be successfully deployed on the local PC and all GPRE services are working correctly, the next step is to deploy it to production into the Google Cloud. The first step is to build to the Docker image again, but instead of writing it into the local Docker registry, we write it towards the Google Container Registry. With the following commands:

```
$ docker build -t gcr.io/gpci-266802/gpre:v2020.05.13 .
$ docker push gcr.io/gpci-266802/gpre:v2020.05.13
```

The docker image is now registered and available inside the Google Container Registry. The GPRE production service in the Google Cloud can now be started using a *gcloud* command. This assumes that the Google SDK is installed and the gpci project is registered as the default project:

```
gcloud compute --project=gpci-266802 instances create-with-container gpre-production --
↪zone=us-central1-a \
              --machine-type=g1-small --subnet=default --network-tier=PREMIUM \
              --metadata=google-logging-enabled=true --service-account  \
              gpre-619@gpci-266802.iam.gserviceaccount.com  --image-family=cos-stable \
              --image-project=cos-cloud --container-image=gcr.io/gpci-266802/gpre:v2020.
↪05.13 \
              --container-restart-policy=always --container-privileged --tags=http-
↪server
```

Accessing the GPRE production service can be done either from a browser window in the google project SSH or through a terminal on the local machine. For the latter, the SSH key must first be registered in the ssh-agent through *ssh-add*. Next an SSH connection can be started using the *gcloud ssh* command:

```
$ ssh-add  ~/.ssh/google_compute_engine
Enter passphrase for /home/wit015/.ssh/google_compute_engine:
Identity added: /home/wit015/.ssh/google_compute_engine (/home/wit015/.ssh/google_
↪compute_engine)

$ gcloud compute ssh gpre-production
No zone specified. Using zone [us-central1-a] for instance: [gpre-production].
  ######################[ Welcome ]########################
  #  You have logged in to the guest OS.                  #
  #  To access your containers use 'docker attach' command  #
```

(continues on next page)

(continued from previous page)

```
  ############################################################

wit015@gpre-production ~ $
```

You are now logged on the container host (the server that hosts the container and is running it through docker). The actual container can be accessed again through a *docker attach* command using the container ID:

```
wit015@gpre-production ~ $ docker container ls
CONTAINER ID        IMAGE                                                                                 ␣
↪COMMAND                  CREATED             STATUS              PORTS                ␣
↪NAMES
4ef2e216e157        gcr.io/gpci-266802/gpre
↪"/entrypoint.sh /sta..."   2 months ago        Up 2 months                              ␣
↪klt-gpre-production-xeif
1c75589fd329        gcr.io/stackdriver-agents/stackdriver-logging-agent:0.2-1.5.33-1-1
↪"/entrypoint.sh /usr..."   2 months ago        Up 2 months                              ␣
↪stackdriver-logging-agent
wit015@gpre-production ~ $ docker exec -it 4ef2e216e157 bash
root@gpre-production:/app# ls
gpre  main.py  main.pyc  prestart.sh  uwsgi.ini
root@gpre-production:/app#
```

The output from the *ls* command inside the container shows the *main.py* file which is the entrypoint for the GPRE services to run.

Updating the container

When updates to the GPRE service become available it also becomes necessary to update the container running on the container host. This is most easily done from the Google Cloud project interface. First go the Container Registry interface and copy the full container name from the container that you want to deploy. Next go the *Compute Engine* section, select the VM Instance and choose Edit. Next to go the Container image and replace the container image with the the container name you copied from the registry. After saving the changes, the VM instance will be rebooted in order to start the new container. See screenshots below for information.

1.5 GPRE database

1.5.1 Overview of schemas

The GPRE database consists of two schemas:

- The `gpre` schema which contains all data required to run the GPRE services
- the `gpre_staging` schema which contains copies or links to the data in the `gpre` schema and can be used to the test and experiment with new services, crops or varietes.

When setting the environment variable `DEVELOP=1` GPRE will always connect to the `gpre_staging` schema. See also the section on *System and Installation*. Otherwise the `gpre` schema will be used.

Most of the objects in the `gpre_staging` schema are views to the `gpre` schema in order to avoid data duplication. However, in order to add and test new crops the following objects are defined as tables with the same structure as the tables in the `gpre` schema:

- *crop*

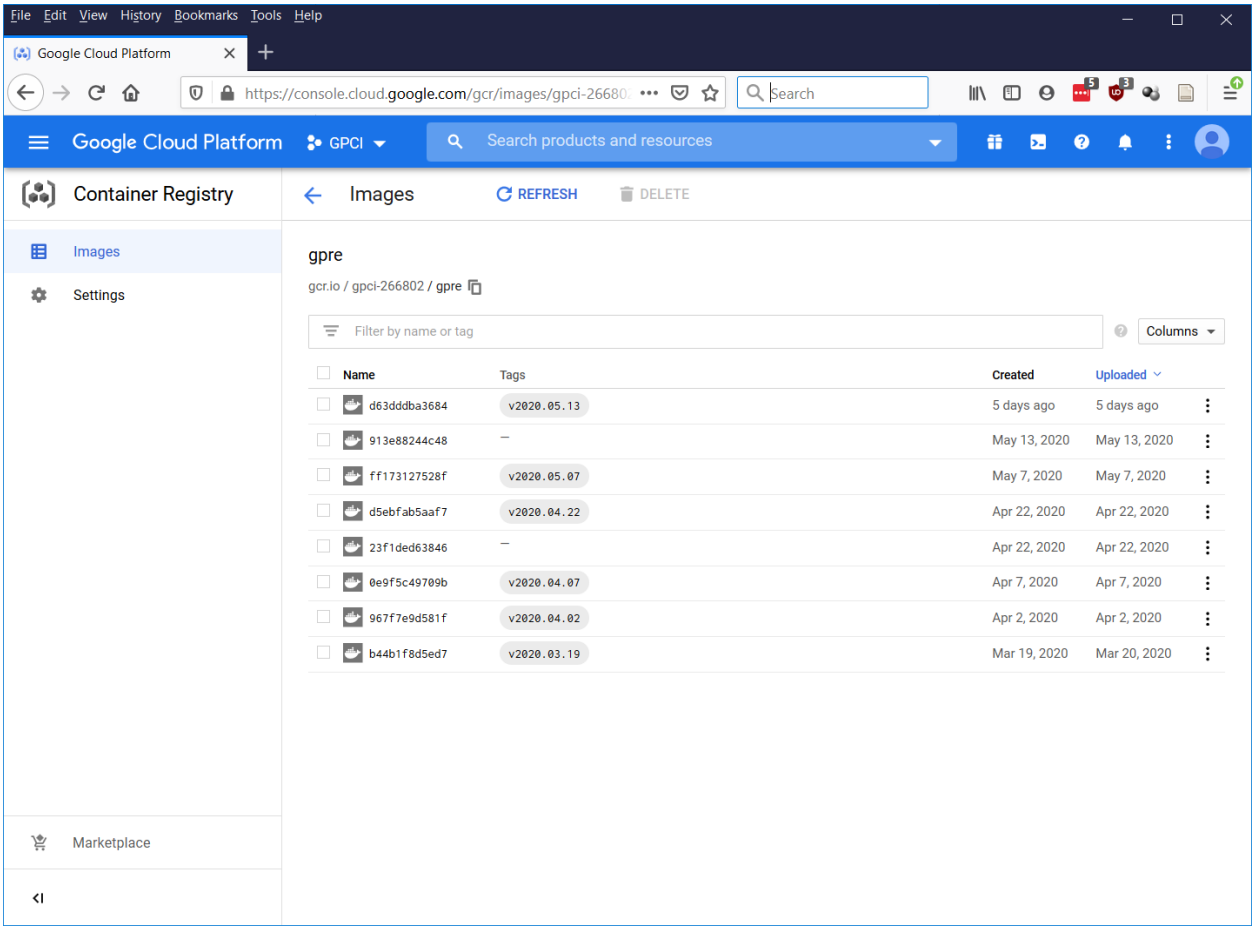


Fig. 4: Copying a container name from the container registry.

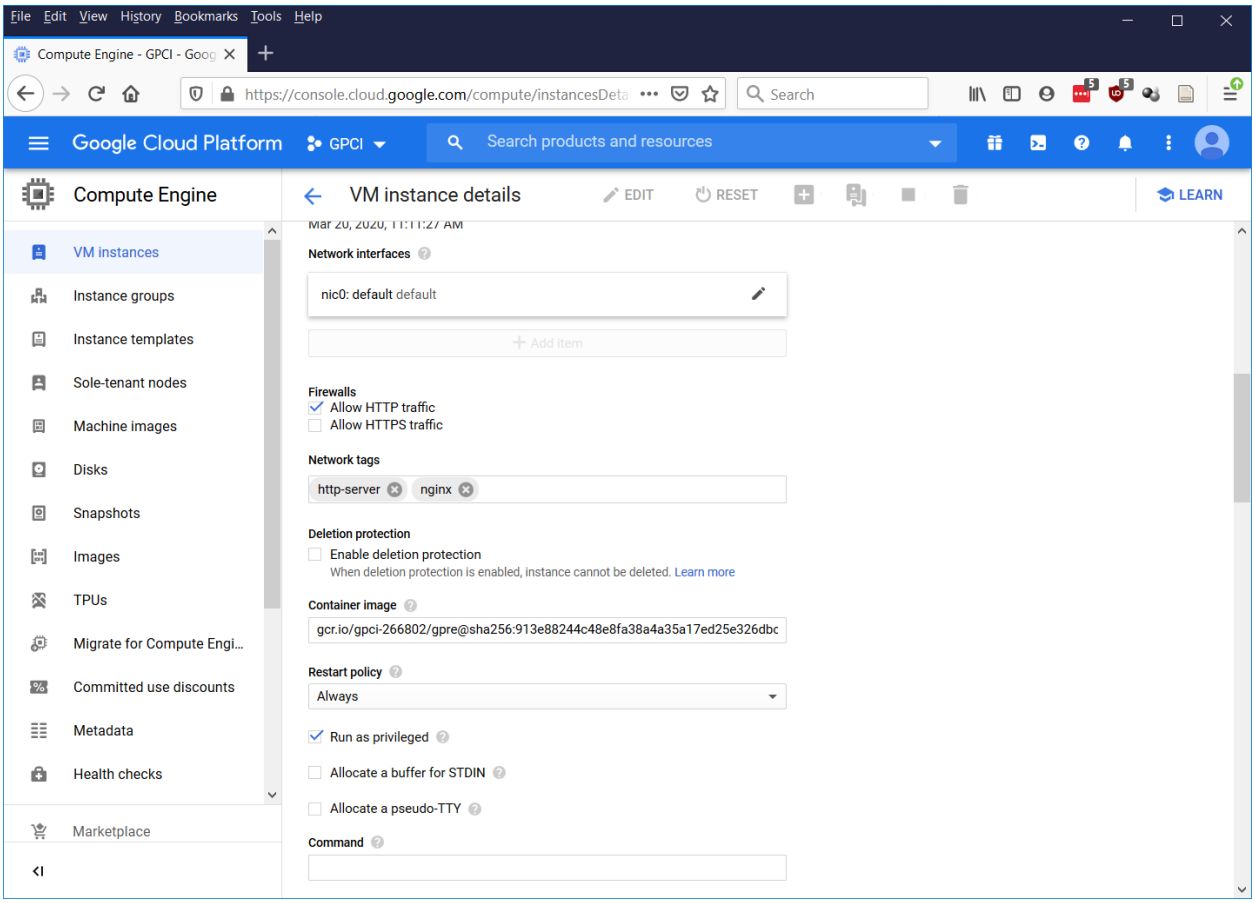


Fig. 5: Replacing a container in a Google VM Instance.

- *crop_parameter_value*
- *varieties*
- *variety_parameter_value*
- *management_alerts*
- *weather_alerts*

1.5.2 Overview of all objects

The following table provides an overview of all objects in the `gpre` schema. Object types are provided as table *T*, view *V`* or procedure *P*. Note that some tables are currently not used or have been replaced by views. Those tables have been kept in the database scheme as they may become relevant in the future.

Name	Type	Description
crop	T	Stores unique crop ID and name
crop_parameter_value	T	Stores crop parameter values for BBCH phenology model
date_manipulation	T	Auxiliary table for operations involving dates (e.g. group-by)
disease_map_cache	T	Stores the disease map data at regional level for Myanmar
era_grid	T	The grid definition of the ERA-INTERIM historical weather archive
grid	T	The 0.1 degree grid definition for Myanmar
grid_005	T	An alternative 0.05 degree grid definition for Myanmar *not used*
grid_weather_forecast	T	Table for storing the weather forecast *not used* (DarkSky API is now used)
grid_weather_forecast_d0	T	Table for storing the first day of the weather forecast *not used*
grid_weather_lta	V	View for providing the long-term-average historical weather data
grid_weather_lta_tbl	T	Table for providing the long-term-average historical weather data *not used*
grid_weather_observed	V	View for providing the actual historical weather data
grid_weather_observed_tbl	T	Table for providing the actual historical weather data *not used*
management_alerts	T	Table for providing management alerts linked to BBCH stages
regions	T	Table for providing information regions in Myanmar
season	T	Stores the season definitions
varieties	T	Stores unique variety ID and variety name for each crop
variety_parameter_value	T	Stores parameters for BBCH model specific for a variety
weather_alerts	T	Stores weather alerts linked to a BBCH phenological stage
weather_hres_grid_myanmar	T	Stores historical weather data from the ERA-INTERIM archive
weather_hres_grid_myanmar_lta	T	Stores long-term-average weather derived from the ERA-INTERIM archive
weather_map_cache	T	Stores the weather forecast for each region derived from DarkSky
get_grid	P	Returns grid ID for given latitude, longitude and cellsize
get_grid_weather	P	Returns actual weather data for given grid ID and year range

1.5.3 Base tables

The base tables in the database define properties that are used in nearly all other tables and views and are used to define relationships. The primary keys in those tables could function as foreign keys in the other tables although this is currently not enforced in the database.

Crop table

Stores the unique crop_no together with a crop_name.

Field	Type	Null	Key	Default	Extra
crop_no	int(11)	NO	PRI	NULL	
crop_name	varchar(40)	YES		NULL	

Varieties table

Stores the unique crop_no, variety_no together with a variety_name.

Field	Type	Null	Key	Default	Extra
crop_no	int(11)	NO	PRI	NULL	
variety_no	int(11)	NO	PRI	NULL	
variety_name	varchar(40)	YES		NULL	

Seasons table

Stores the identifiers for the different cropping seasons. Management alerts can be different for different cropping seasons and therefore it can be useful to descriminate between seasons.

Field	Type	Null	Key	Description
season_no	int(11)	NO	PRI	Unique season ID
season_name	varchar(40)	YES		Name of the cropping season
season_definition	varchar(60)	YES		Description of the season

Regions table

Stores unique code of the lowest level administrative regions (GID_3) including the latitude/longitude of each region and the administrative regions to which it belongs.

Field	Type	Null	Key	Default	Extra
GID_0	char(3)	NO		NULL	
NAME_0	varchar(50)	NO		NULL	
GID_1	varchar(10)	NO		NULL	
NAME_1	varchar(50)	NO		NULL	
GID_2	varchar(20)	NO		NULL	
NAME_2	varchar(50)	NO		NULL	
GID_3	varchar(20)	NO	PRI	NULL	
NAME_3	varchar(50)	NO		NULL	
TYPE_3	varchar(50)	NO		NULL	
longitude	decimal(8,3)	NO		NULL	
latitude	decimal(8,3)	NO		NULL	

grid table

Stores the unique `grid_no` for the grid definition in Myanmar. Moreover it provides the latitude and longitude of the grid centroids, the average elevation of the grid terrain (over land), and whether the grid contains land (`has_land` = 1). The additional column `idgrid_cgms14glo` provides the ID of the nearest grid in the `era_grid` table. The latter is required to build the link between the GPRE grid definition and the global ERA-INTERIM grid definition.

The tables `grid_005` has the same structure as the `grid` table. The structure of the table `grid_era5` is also similar.

Field	Type	Null	Key	Default	Extra
grid_no	int(11)	NO	PRI	NULL	
latitude	float	NO		NULL	
longitude	float	NO	MUL	NULL	
elevation	float	YES		NULL	
has_land	int(11)	NO		NULL	
idgrid_cgms14glo	int(11)	NO		NULL	

1.5.4 Weather tables

Currently, only the historical weather data and its climatology is stored in the GPRE database because the weather forecast is directly derived from the DarkSky API. The historical data is derived from the ERA-INTERIM archive that is available at Wageningen Environmental Research (WEnR). Data from the WEnR data is transferred each day for the Myanmar window. The tables are replicated from the WEnR database and therefore have a slightly different structure compared to the other weather tables. The mapping between the WEnR structure and the GPRE structure is accomplished through the views `grid_weather_observed` and `grid_weather_lta`.

The weather tables that store the ERA-INTERIM weather archive (`weather_hres_grid_myanmar`) and its climatology (`weather_hres_grid_myanmar_lta`) have the following structure.

Field	Type	Null	Key	Description and units
idgrid	int(11)	NO	PRI	Unique grid ID
day	date	NO	PRI	date or day number
temperature_max	decimal(3,1)	NO		degrees Celsius
temperature_min	decimal(3,1)	NO		degrees Celsius
temperature_avg	decimal(3,1)	NO		degrees Celsius
vapourpressure	decimal(4,2)	NO		vapour pressure hPa
windspeed	decimal(5,1)	NO		wind speed m/sec at 10m
precipitation	decimal(4,1)	NO		precipitation in mm/day
e0	decimal(4,2)	NO		open water reference evaporation in mm/day
es0	decimal(4,2)	NO		soil reference evaporation in mm/day
et0	decimal(4,2)	NO		crop reference evapotranspiration in mm/day
radiation	decimal(6,0)	NO		Incoming global radiation in kJ/m2/day
snowdepth	decimal(6,2)	YES		Snow depth in cm

The weather tables and views whose name starts with "grid_weather" have a structure that is similar to the table below.

Field	Type	Null	Key	Description and units
grid_no	int(11)	NO	PRI	grid identifier
day	date	NO	PRI	date or day number (in case of LTA
maximum_temperature	decimal(10,5)	NO		degrees Celsius
minimum_temperature	decimal(10,5)	NO		degrees Celsius
vapour_pressure	decimal(10,5)	NO		vapour pressure hPa
windspeed	decimal(10,5)	NO		wind speed m/sec at 10m
rainfall	decimal(10,5)	NO		precipitation in mm/day
e0	decimal(10,5)	NO		open water reference evaporation in mm/day
es0	decimal(10,5)	NO		soil reference evaporation in mm/day
et0	decimal(10,5)	NO		crop reference evapotranspiration in mm/day
calculated_radiation	decimal(10,5)	NO		Incoming global radiation in kJ/m2/day
snow_depth	decimal(10,5)	YES		Snow depth in cm

1.5.5 Crop parameters and alerts

Tables for phenology parameters

There are two tables for storing crop phenological parameters, these are named `crop_parameter_value` and `variety_parameter_value`. The parameter values for a specific variety take precedence over the parameter for the crop. In practices this means that temperature response functions for phenology are often specified per crop, while the number of degree-days for reaching a phenology stage are described for each variety specifically. Both tables have a structure similar to the one below.

Field	Type	Null	Key	Description
crop_no	int(11)	NO	PRI	The crop number
parameter_code	varchar(20)	NO	PRI	the parameter name
parameter_value	varchar(255)	YES		the parameter value
parameter_description	varchar(255)	YES		the description of the parameter

Tables for messages and alerts

The system contains two tables for storing messages and alerts. Management messages are stored in the table `management_alerts` which provides the crop management messages linked to a particular crop BBCH stage, see table below.

Field	Type	Null	Key	Description
crop_no variety_no	int(11) int(11)	NO NO	PRI	crop ID variety ID season ID message ID
season_no message_no	int(11) int(11)	NO NO	PRI	BBCH code to which the message corresponds
BBCH_code offset_days	varchar(45)	YES	PRI	days before (-) or after (+) reaching the BBCH
management_msg	int(11) longtext	YES	PRI	stage The message itself
		YES		

Weather alerts are signalled when (a combination of) adverse weather conditions occur that are important for a farmer to take action on. Such weather alerts can for example be defined as the probably of fog occurrence on three consecutive days, which would increase the changes of development of late blight in potato. The definition of the weather alerts is done in the table `weather_alerts` (see below). The parameters required for such an alert can be highly crop specific and therefore the parameters are stored in the table as a JSON string which is parsed by the system.

Field	Type	Null	Key	Description
crop_no	int(11)	NO	PRI	crop ID
variety_no	int(11)	NO	PRI	variety ID
season_no	int(11)	NO	PRI	season ID
message_no	int(11)	NO	PRI	message ID
parameters	varchar(255)	YES		parameters for weather alert as JSON string
weather_msg	longtext	YES		the weather alert message
signal	varchar(255)	YES		the signal to be broadcasted, see the `pcse.signals` module

1.5.6 Caching tables

Caching tables are used to stored pre-computed results which would otherwise take to long provide to the user. The system contains two caching tables, one for weather maps `weather_map_cache` and one for disease maps `disease_map_cache`. The tables just store the computed results as JSON for a given day (and disease). The HTTP API is simply returning the data for the current day from the relevant table.

CHAPTER

TWO

CODE DOCUMENTATION

2.1 GPRE code documentation

This section provides an overview of all code documentation that is part of GPRE. This part of the documentation is mostly generated from the documentation headers in the source code. Further, it can be used to quickly navigate the source code as each documented function or class is linked to the relevant code section using the [source] link.

2.1.1 GPRE service implementation

Code for all specific GPRE services is described here

Weather service

Crop stage prediction service

Disease service

2.1.2 Webserver scripts

All webserver scripts are thin wrappers around GPRE scripts that perform the actual processing. All webserver scripts use Flask through a WSGI interface for exposing the HTTP URL.

2.1.3 Simulator for phenology, management alerts and weather alerts

Simulation of crop phenology

Simulation of management alerts simulator

Simulation of weather alerts simulator

Data providers

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search