CANDY 22 – MANUAL I

Description of Usage

Contributors:

Uwe Franko, Burkhard Oelschlägel, Stefan Schenk, Martina Puhlmann, Katrin Kuka, Janine Mallast née Krüger, Enrico Thiel, Nadia Prays, Katharina Meurer, Eric Bönecke, Lukas Hey, Stefan Gasser

Software available for download at www.frug.info/candy_main.php

Inhaltsverzeichnis

INTRODUCTION **4**

GENERAL SYSTEM SETUP **8**

Install Interface / Initialization *8*

USER INTERFACE **11**

MODEL INPUTS AND INPUT TAB 12
General *13*
Management *14*
Observations *15*
Climate data *16*
Soil parameters *21*
Output features *23*
MODEL PARAMETERS 24
RESULT TABLES 26
EVALUATION 27
FITTING 29
Simplex algorithm *32*
CHECK INI FILE 34
SQL INTERFACE 35
STC VIEWER 36
MAIN MENU 37
CONTEXT MENUS 38
SIMULATION RUNS 38
Single simulation run *38*
Periodic simulation runs *39*
Batch file settings *39*
Batch call parameters *39*
MODEL SETTINGS 42
CANDY ERROR HANDLING 43
Checklist to avoid errors *43*

APPENDIX A: POSTGRES INSTALLATION **45**

APPENDIX B: MAIN DATA MODEL **46**

CDY_FXDAT *47*
CDY_MADAT *48*
CDY_MVDAT *51*
CDY_CLDAT *52*
PROFILE *52*
CDY_HRZN *52*
CDY_PROP *53*
CDY_PROP_SEL *53*
CDY_PROP_LST *53*
CDY_RSLT *54*
CDY_RSLT_SEL *54*
CDY_RSLT_LST *54*

APPENDIX C: WORKING WITH BIG DATA SETS **55**

APPENDIX D: IMPORT DATA FROM AN OLD CANDY21 MSACCESS-DATABASE **56**

APPENDIX E: DYNAMIC CROP GROWTH MODEL **60**

APPENDIX F: AUTOMATIZATION OF CANDY RUNS **62**

CDY_AUTOMAT *62*

LIST OF FIGURES

Figure 1: symbolic representation of processes that are handled by CANDY 5
Figure 2: part of the file pg4candy.ini that has to be in the same directory as cdy22_GUI 8
Figure 3: starting form of cdy_prepare 9
Figure 4: a) start searching the postgres library; b) import option for MS-ACCESS data 9
Figure 5: Candy user interface. Main elements are a database selector (red frame), a tree view presentation of the field plots (indicated) and a tab view (right) for a detailed data presentation. The message box (indicated) is filled according to the selected tab and shows specific hints. 11
Figure 6: view and edit cdy_fxdat information 12
Figure 7: view and edit the management data 14
Figure 8: view and edit the experimental observations 15
Figure 9: view and edit climate data after clicking the header 16
Figure 10: txt-file example in the editor window as template to prepare climate data import. 18
Figure 11: view and edit the soil properties. 22
Figure 12: view or edit a schema for result recording 23
Figure 13: view or edit a schema for 'virtual' observation events to get the corresponding model output .. 24
Figure 14: check selected parameter tables 24
Figure 15: import missing parameter records from repository 25
Figure 16: check the recorded result data 26
Figure 17: check model results related to observation events 27
Figure 18: option to calibrate model parameters 29
Figure 19: dialogue window to add a new parameter 30
Figure 20: options to start the optimisation. 31
Figure 21: Basic triangle simplex is BNW (red) with the worst point W, next best point N and best point B. M is the centroid of all Pi without W, thus of B and N. R is a reflected W. E is reflected and expanded W. CR and Cw are contraction points from R or W depending on the result of the reflection. Adapted from Bezerra et al. (2016) and Nelder and Mead (1965). B: If the direction is wrong and reflection, expansion and contraction failed, a multiple contraction of the basic triangle BNW towards the best point B takes place and results in a new triangle BNnewWnew. 33
Figure 22: check and edit the configuration file pg4candy.ini 34
Figure 23: simple interface to apply SQL commands and scripts 35
Figure 24: view the content of *.stc files 36
Figure 25: schematic representation of periodic simulations for nrep=4. The repletion starts at the beginning of the first day of the second year and ends at the last day of the last year of the basic mananagement interval. This means, that the time from the original stop date to the end of the last year is filled without events (like fallow). While management is repeated in multiple cycles the timeline for results is represented by the time scale of climate data. 39
Figure 26: Example for running candy22 via batch calls from master (1) and slave (2). The latter is using parameters defined in the master, in this case information about soil (%1, blue), weather (%2, red), management/file_name (%3, green) and an "index", combining the essence of the previous given parameters with the plot- and subindex (%4, orange). Besides the mandatory "candy call" in the slave batch-file, additional scripts can be executed before/after candy is/was called. 41
Figure 27: a) selectable components for postgres installation; must select PosgreSQL Server & pgAdmin 4; b) provide a password for the default superuser; c) provide port number (default 5432); d) installation summary. Check if port and directories are as desired. 45
Figure 28 :data model showing links between main user related tables 46
Figure 29: open the MS-ACCESS file 56
Figure 30: start data transfer for each table 57
Figure 31: join names to fill up all required fields 58
Figure 32: make the database available for CANDY 58
Figure 33 : autocrop section of the parameter tab showing all essential parameters used be the autocrop model 60

Introduction

This document provides an overview of:

- the general system setup,
- preparing and running simulations with the CANDY interface,
- the usage of batch files for simulations without the CANDY interface, and
- a checklist to avoid errors.

CANDY simulates the dynamics of carbon and nitrogen in the unsaturated (vadose) zone of agriculturally used soils. The model application should preferably be focused on sites with a rooting zone of up to 2 m. The soil profile is divided into homogeneous layers of 1 dm thickness each. The simulation proceeds in daily time steps. Starting from initial values for all state variables, the model simulates the impact of management and climate on them. The following processes are included (see Figure 1):

- climate conditions (access to database or generating climate data, correction of systematic errors of observed precipitation),
- soil water dynamics (potential and actual evapotranspiration, percolation through soil),
- soil temperature dynamics,
- turnover (mineralization and humification) of organic matter in soil,
- nitrogen dynamics (mineralization, immobilization, uptake, leaching, gaseous losses, symbiotic N fixation).

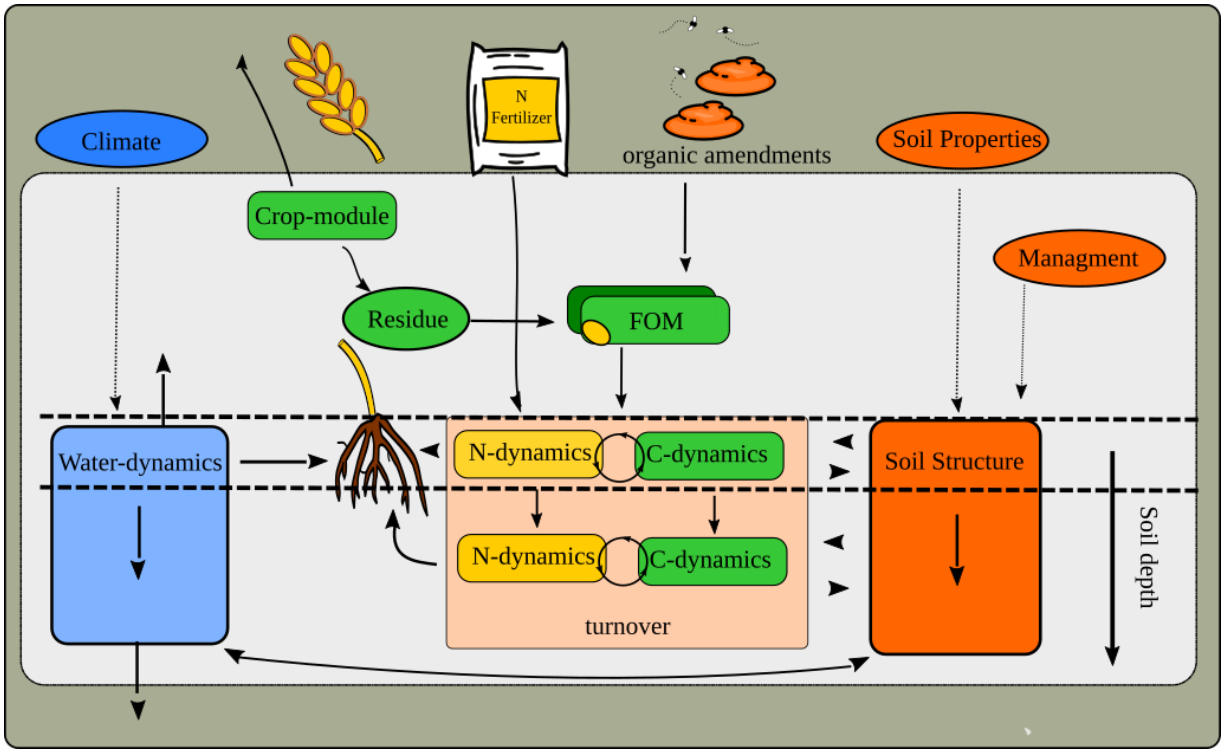


Figure 1: symbolic representation of processes that are handled by CANDY

CANDY comprises a graphical user interface (GUI;Figure 5) that organizes the access to the data, the preparation of simulation tasks, and the simulation model itself. The data is stored in a postgres database.

Upon executing the application, the main window opens as pictured in Figure 1. A list of available databases (red box in Figure 1) is shown in the main window and the user can quickly select the desired database. Each database may contain a number of folders representing meaningful subunits, e.g. different experiments or groups of fields or farms. Each folder contains a number of single farm fields or experimental plots whereby each of them is treated homogeneously over time. That means if fields have been divided in the past it is necessary to handle them as separate plots right from the beginning of the simulation. No division of plots can be made as part of a simulation run. A plot is the smallest unit for the process simulation.

A simulation task covers site description and scenario data. The site description includes climate data and parameters of the horizons in the soil profile.

The necessary climate data (daily time steps) consist of:

- air temperature at 2 m height (°C)
- global radiation (J cm-²) or daily sunshine duration (h)
- precipitation (mm)

For each soil horizon the following parameters are used:

- depth of the horizon (dm)
- stone content (Vol. %)
- bulk density (g cm^{-3})
- soil moisture (Vol.%) at field capacity, wilting point as well as pore volume
- soil texture: mass fraction (M.%) of clay, fine silt, and silt (the model also accepts the amount of particles < 6.3 µm as FAT parameter)
- saturated hydraulic conductivity (mm d^{-1})
- C_{org} concentration (M%); if physical parameters are considered to be state variables, the value has to be consistent with the given soil physical parameters

The scenario data include the agricultural management and the description of initial conditions, e.g. in terms of SOM content at the beginning of a simulation. The latter can be reconstructed from detailed observations. If no measurements/observations are available, the model initialization can be based on information about former SOM reproduction. It may be helpful to apply the CNP model beforehand (http://www.frug.info/candy_cnp.php) to reconstruct carbon reproduction flux from crop rotation and average yields in history. Furthermore, information about soil moisture (given as a part of field capacity) and an indication of the nitrogen state at starting time is required as an initial condition. Alternatively, it is recommended to include several years as a 'pre-treatment' for model spin up.

In order to be able to run a simulation, the model needs the description of a management activity. The following farming activities (managements) are currently included in the Candy model:

- soil tillage: date, used device, tillage depth (cm), maximum regarded depth (dm)
- irrigation: date, applied water amount (mm)
- application of mineral N fertilizers: date, N amount (kg ha^{-1}), type of fertilizer, share of NH4-N (%)
- organic amendments: date, type of organic matter, fresh matter amount (dt ha^{-1}), C amount (kg ha^{-1}) (Note: In case of slurry application, additional information about dry matter content, total nitrogen content, and mineral nitrogen content can be used by the model.)
- cropping: date of sowing and/or emergence and harvest, crop yield (dt ha^{-1}) and N uptake with main and side product (kg ha^{-1}), usage of the side product (removed or

left on field) (Note: If by-products are not removed and data are available it is recommended to apply this as organic amendment instead of using the 'left on field' option.)

- ploughing: (see soil tillage) or ploughing up
- cutting of grassland and forage crops: date, optional: yield (dt DM ha^{-1})
- grazed grassland: start/stop grazing: day, number of cattle units driven to/from the plot

Please note that information on yield and biomass have to be provided in the unit decitonnes per hectare (dt ha^{-1}). 1 decitonne is equivalent to 100 kg.

General system setup

Install Interface / Initialization

Necessary components

Before starting the installation of the Candy model make sure that you have both apps, **cdy22_GUI** and **cdy_prepare**. The first app is the Candy model itself. The app **cdy_prepare** helps to set up a new CANDY database or to transfer a database from the older format (ACCESS) to postgres (only in 32 bit version) – this makes the data available for Candy simulations.

Postgres can be downloaded from

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

A description of the installation process can be found in Appendix A.

The CANDY user interface provides all tools for data manipulation. Experienced users may find that more complex data management tasks can be accomplished more efficient using SQL tools to edit the data tables. The postgres installation includes a simple administration tool: pgAadmin. More flexibility is provided by the tool DBeaver (https://dbeaver.io/download/) that can handle not only postgres databases.

A successful database connection requires an ini-file named *pg4candy.ini*, which should be located in the directory of the candy app providing necessary details about the postgres databases. Figure 1 provides an example of the first part of an ini-file containing a list of two databases (in square brackets):

```
[driver]
w32=C:\Users\Public\Documents\Embarcadero\Studio\FireDAC\libpq.dll


[databases]
candydb0=1
candydb1=2
...
```

Figure 2: part of the file pg4candy.ini that has to be in the same directory as cdy22_GUI

This ini-file (or configuration-file) can be created automatically by running the **cdy_prepare** application.If no previous database is available, it is recommended to install the included demo database. Before proceeding with **cdy_prepare**, the user needs to know some basic information about the postgres server. The creation of a new database and (if desired) a new

user, requires an account (username and password) which you already should have created during the postgres installation (superuser and superuser-password). The next important pre-requisite is that the driver library matches the system and the chosen bitness of the CANDY app.

On windows system the driver library is called libpq.dll and can be found in the postgres installation directories. On MAC it is called libpq.dylib.

It may be useful to copy this library to a place where it is accessible by the CANDY software. The following chapter describes the preparation of the database step by step.

First preparations before modelling begins

1. start the application cdy_prepare.exe

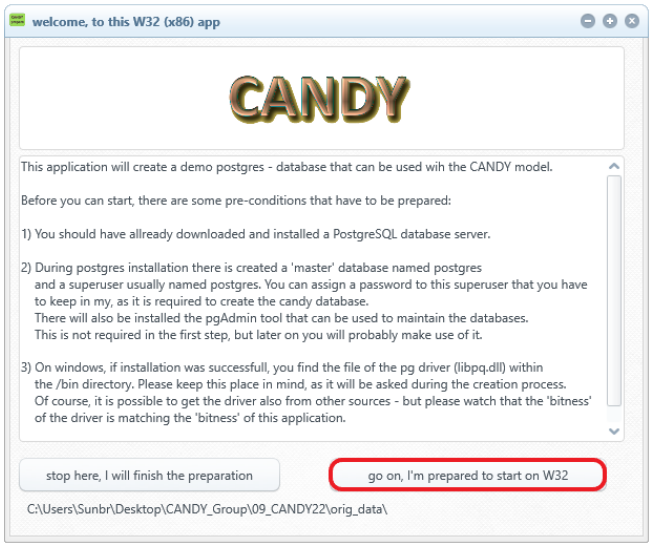


Figure 3: starting form of cdy_prepare

2. using the account data information from your postgres installation click: go on (red frame)

a) **b)**

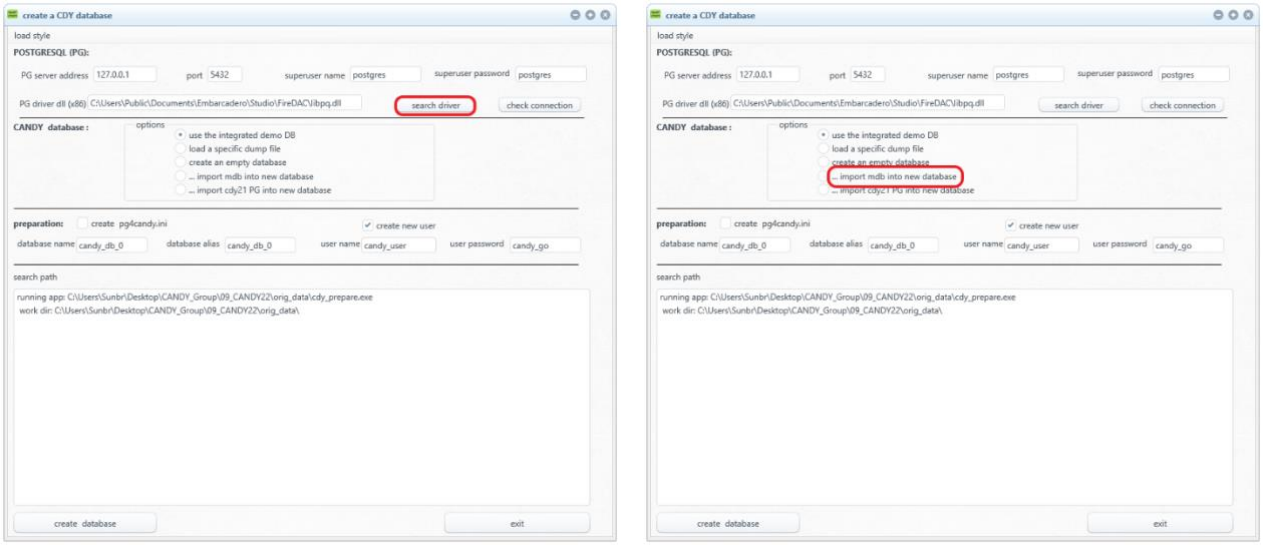


Figure 4: a) start searching the postgres library; b) import option for MS-ACCESS data

If the path to the postgres library is not known click the search button to scan for the required file. Search-results are shown in the listbox. You can stop the search manually, if you think the proper library has been found.

But careful, there is an issue with the bitness of the library on windows systems. It has the same name for 32 and 64 bit, but has to fit to the bitness of the installed postgres server. Probably this will be no problem if you have only one server running.

After activating the proper library, you can continue with creating the database. In order to do so, different options are available:

- Create the integrated demo database
- Import a database from an existing dump file
- Create an empty database
- import an ACCESS database from candy21 (figure 3 b)) red frame; only Windows 32 bit)
- transform a candy21 postgres database to candy22 compatibility.

Please give the database a name and assign it to a user account. If you have already created a database, you may un-check:

- the option to create an ini-file and
- if already existing, the option to create a user account.

3. Option: start the import of an ACCESS database

This will only work with 32bit Versions and may be not relevant for most users. Please refer to Appendix D for more details.

User Interface

After a successful database creation or import you can start the user interface (**cdy22_GUI**) which should be located in the same directory as the ini-file. The application window should look similar to that in figure 8.

The desired database to be used can be activated from the main menu (red frame). In order to see the content of this database you have to inflate the tree-view by clicking its root (green frame). You will then see optional folders that contain the actual fields/plots. Activate a field and proceed with data processing as described in the following chapters. More databases can be connected from the menu point "add existing databases" which activates a form to define an existing databases connection (must be included in the ini-file). You also have the option to import a dump from a third-party source (i.e. to exchange data between different local postgres servers). Creating a database dump is relatively simple by using the backup tool from DBeaver (right click on a database → Tools → Dump database). It is recommended to delete (or comment using "--") following lines in the dump SQL file:

CREATE SCHEMA public;

COMMENT ON SCHEMA public IS 'standard public schema';

SET default_table_access_method = heap;

Make sure that the last character of the script is a semicolon ';' (no empty lines at the end).

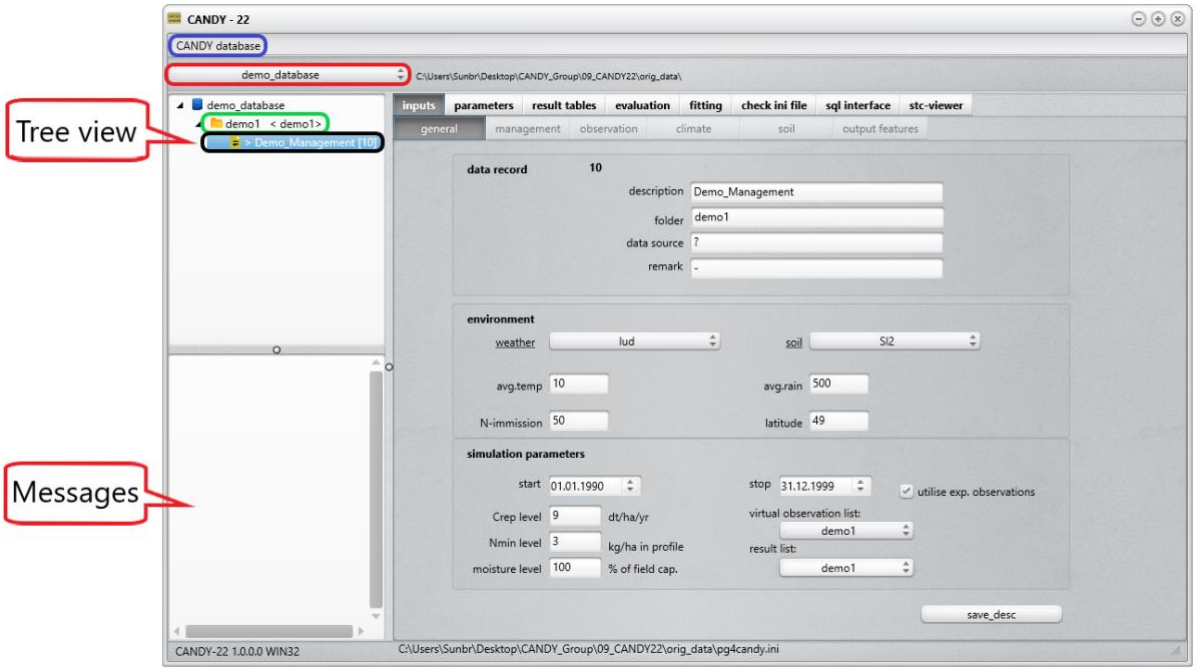


Figure 5: Candy user interface. Main elements are a database selector (red frame), a tree view presentation of the field plots (indicated) and a tab view (right) for a detailed data presentation. The message box (indicated) is filled according to the selected tab and shows specific hints.

In the tree view, you must first select a folder (green frame) and then a field plot (black frame), with the active field plot being highlighted, clicking on tree view items opens context menus to start specific actions. The selection of a field plot updates the information tab view. The individual tabs will be explained in the following chapters.

Further options of the main menu (Figure 8, blue frame) are for the export and import of datasets. This provides the opportunity to wrap all data from a single folder including the required parameter records in a SQLite file that can be transferred for either exchange with other users or as a backup of the user data. Nonetheless it is always strongly recommended to regularly make a full backup of the current database by using the tools provided by postgres or DBeaver.

The data import from a SQLite file may require the insertion of data into the user tables without touching the parameters. The parameter tables are available in the trans schema and can be used for an update if required using the same functionality as with updating from a repository – only in this case it should be avoided to update the transfer tables.

Model inputs and Input tab

Here we find several sub-tabs for specific information classes.

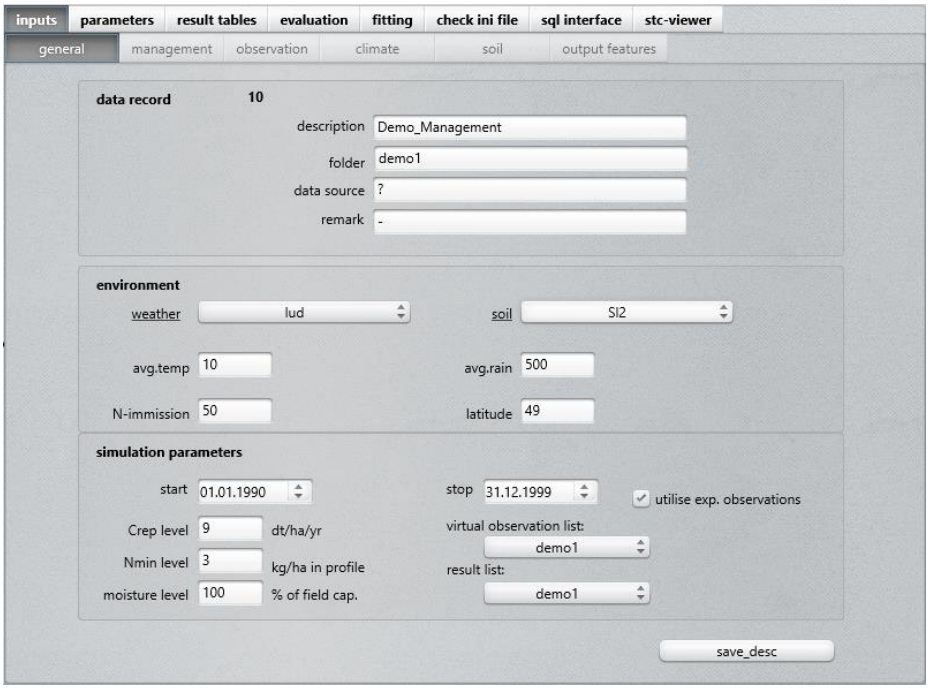


Figure 6: view and edit cdy_fxdat information

General

The *general* sheet shows all data describing the plot and the simulation scenario, with the sub-panels for *environment* settings and *simulation parameters* showing the main content of the CDY_FXDAT table of the selected field plot.

Environment

The description of environmental site conditions include:

- weather: climate station selection from popup (should be prepared in advance, see chapter Climate data); a little asterisk will be shown beside the *weather* label if a climate generator file is available for the selected station.
- soil: profile selection from popup (should be prepared in advance, see chapter Soil data)
- avg. temp.: long-term annual average temperature
- avg. rain: long-term average of annual precipitation
- N-immission: annual average of solid, liquid, and gaseous N compounds
- latitude: latitude of the field plot (only required to transform sunshine duration into global radiation)

Simulation parameters

The simulation parameters provide details about the start and initialisation of a model simulation run, such as start and stop dates, rough values to initialise carbon (default: 9 dt/ha), nitrogen (default: 100 kg/ha) and water (default: 100 %) and the output options for selected result and observations which are described later.

Management

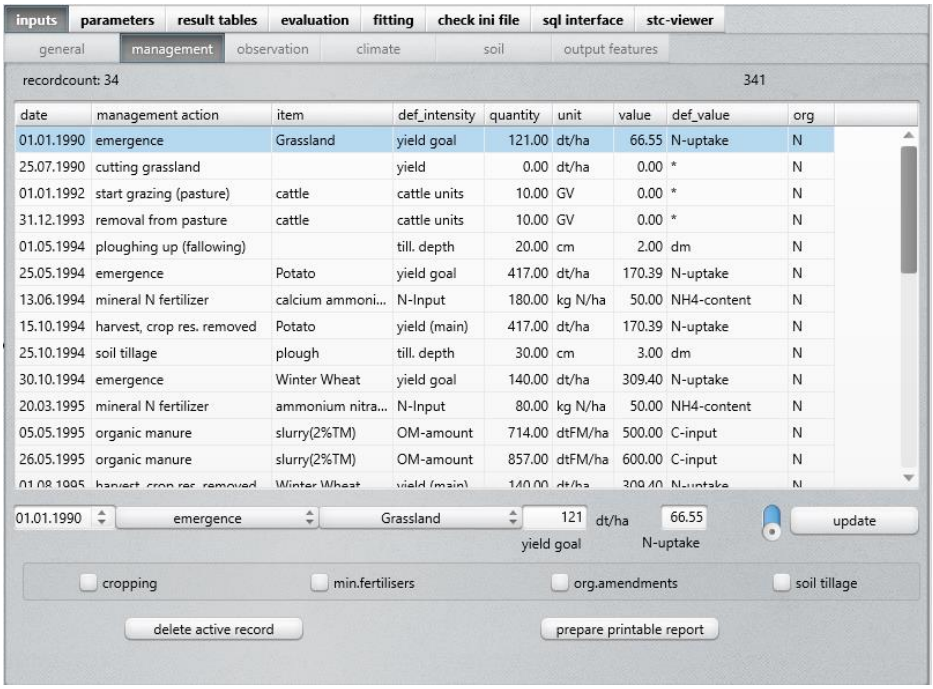


Figure 7: view and edit the management data

The [management] sheet allows to view, insert or update the farming activities for the simulation scenario of the selected field plot. Use the switch to *update* or *insert* a record. You need to specify the date, select a management action (via drop down menu) and edit the respective details (subject and intensity). To further specify the intensity, you may change the auto-filled content of the rightmost field.

Data will be stored with max. 2 decimals – there will be a warning if data loss may be possible. The data for *quantity* and *value* are usually connected by the specific parameter values (i.e. a given amount of manure represents a specific amount of C). There may be cases where the *value* (C amount of organic amendments or N-uptake with yield) is known and should not be calculated from standard parameters. Those records will show a yellow marker beside the *value* edit field and a warning message will appear before the record is saved.

Records from the active management can be deleted with the *delete active record* button and/or can be printed using the *prepare printable report* button.

The check boxes in the lower part may be helpful to filter long data sets for a better overview. You can add or remove tick marks to specify which parameters should be displayed: crops, mineral fertilization, organic amendments, and soil tillage.

Observations

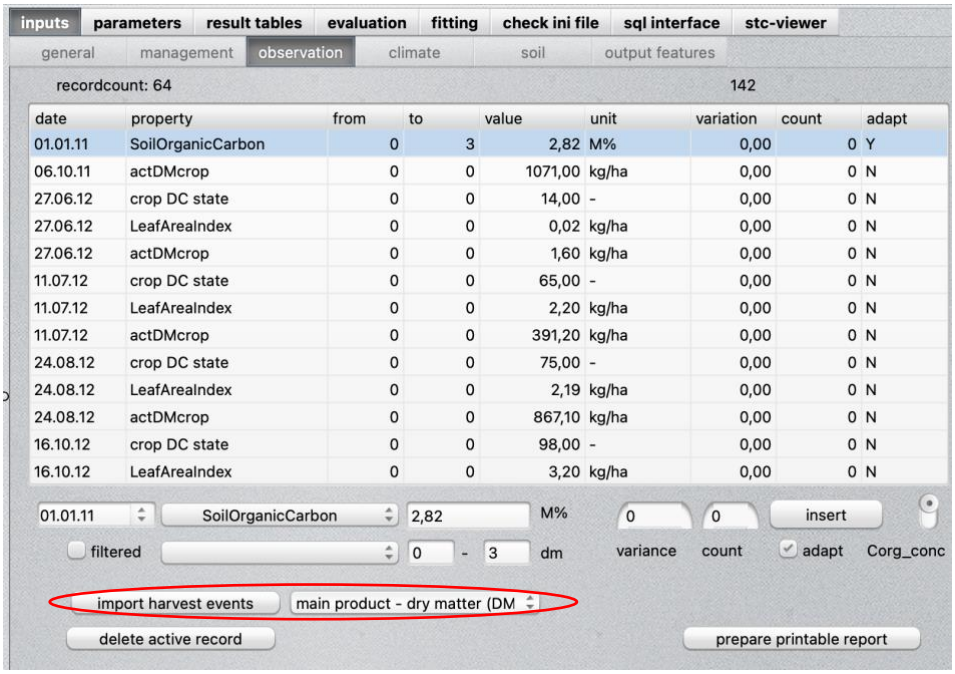


Figure 8: view and edit the experimental observations

Select the [observation] sheet to edit the experimental data. To let the model fit right through a data point, the adaptation option *adapt* (adapt = Y, default = N) should be checked. In this case the internal model values will be overwritten with the desired observed value. Usually, this option is unchecked, and CANDY stores the calculated values for an evaluation of the simulation results.

To reduce the number of selectable properties you can filter by parameter type and select the respective corresponding parameters in the drop-down field below.

Variance and count are both optional inputs that may be used for statistics and can be left blank to insert standard values.

If the management data of the selected plot contain harvest records of crops that are modelled with the dynamic crop growth model (modell='agro1'), it is possible to insert observation records of several harvest attributes for the complete scenario. Otherwise, this option (marked with red in Figure 8) is not enabled.

The input of sampling depth is restricted to integer numbers in [dm] because the model considers soil layers of 1 dm height. Results are located at the centre of a layer. It is possible to relate to measurement depths in 5 cm steps. The depth values 1 – 1 relate to 10 depth and provides the average model result of the first and second calculation layer. The depth values 0-1 relates to 5 cm depth and provides the model value for the first calculation layer. Above ground properties can be specified with depth 0-0. All results in the soil profile are in depth >= 5 cm. The selection of depth steps 0-0 for below ground data is not supported.

Climate data

Data check

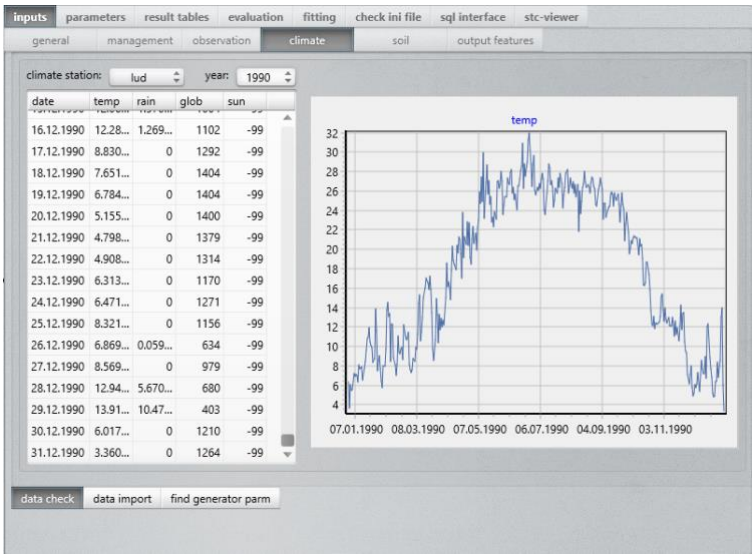


Figure 9: view and edit climate data after clicking the header

The [climate] sheet provides the possibility to view, edit, verify or import climate data. The chart is updated after clicking on the header of the table at the left-hand side, displaying the data for the selected climate station and specific year.

Climate data must be provided in daily time steps without any missing values and gaps for the entire period of the simulation scenario. ***Datasets must start at the beginning of the year at 1.1.*** *even if the simulation scenario starts later. The end of climate data is arbitrary but should not finish before the end of the simulation scenario. While precipitation data from a local dataset should be used, practice shows that temperature and radiation data from adjacent stations can be sufficient. CANDY will try to activate a weather generator If climate data are missing within the simulation scenario*

The import of climate data is possible from a textfile containing daily information about date (*dat*), precipitation (*rain*), temperature (*temp*), sunshine duration (*sun*) and global radiation (*glob*). Data columns have to be separated by a semicolon , decimal point must be used for numbers and the file must not have a header line. If glob data are available the sun data should be coded as -99 (missing value). The following text boxes show a batch file for windows and a shell script for mac users that can be used as templates. Still, several adaptations are required. The relevant parts are emphasized by bold types. The weather station is here ***xyz*** and should be replaced by an appropriate symbol. The textfile in the example is ***cdy_cldat_headno.csv*** and should be written in utf-8 format. If the date format is different from the German style, the format template in ***create table tmp_clidat1..*** has to be adapted accordingly.

Alternatively, you can use other tools (like DBeaver) to compile climate data directly in the database.

```
set pgr=c:\Program Files\PostgreSQL\15\bin\
echo %path% |find /i "%pgr%">nul  || set path=%path%%pgr%;
set PGPASSWORD=candy_go
set adb=candydb0
set host=10.0.2.2
set port=5432
set user=candy_user
set wst=xyz
set acsv='cdy_cldat_headno.csv'
psql -h %host% -p %port% -U %user% -d %adb% -c "CREATE UNIQUE INDEX if not exists cdy_cldat_datum_idx ON public.cdy_cldat (datum,wstat) "
psql -h %host% -p %port% -U %user% -d %adb% -c "drop table if exists tmp_clidat; "
psql -h %host% -p %port% -U %user% -d %adb% -c "drop table if exists tmp_clidat1; "
psql -h %host% -p %port% -U %user% -d %adb% -c "create table tmp_clidat (dat varchar, rain float8, temp float8, sun float8, glob float8) ;"
psql -h %host% -p %port% -U %user% -d %adb% -c "\copy tmp_clidat from %acsv% with DELIMITER ';' "
psql -h %host% -p %port% -U %user% -d %adb% -c " create table tmp_clidat1 as select to_date(dat,'dd.mm.yyyy') dat, rain,temp,sun,glob from tmp_clidat"
psql -h %host% -p %port% -U %user% -d %adb% -c " insert into cdy_cldat (datum, nied,ltem,sonn,glob,widx,wstat) select dat, rain,temp,sun,glob , concat( '%wst%' ,'_', date_part('year', dat)) , '%wst%' from tmp_clidat1; "
psql -h %host% -p %port% -U %user% -d %adb% -c "drop table if exists tmp_clidat; "
psql -h %host% -p %port% -U %user% -d %adb% -c "drop table if exists tmp_clidat1; "
pause
```

```
#!/bin/sh
adb=candydb0
host=127.0.0.1
port=5432
user=candy_user
wst=xyz
# textfile without header containing dat,rain,temp,sun glob
acsv='/Users/.../cdy_cldat_headno.csv'
alias pgr="psql -h $host -p $port -U $user -d $adb -c"
pgr "CREATE UNIQUE INDEX if not exists cdy_cldat_datum_idx ON public.cdy_cldat (datum,wstat) "
pgr "drop table if exists tmp_clidat; "
pgr "drop table if exists tmp_clidat1; "
pgr "create table tmp_clidat (dat varchar, rain float8, temp float8, sun float8, glob float8) ;"
pgr "\copy tmp_clidat from $acsv with DELIMITER ';' "
pgr " create table tmp_clidat1 as select to_date(dat,'dd.mm.yyyy') dat, rain,temp,sun,glob from tmp_clidat"
pgr " insert into cdy_cldat (datum, nied,ltem,sonn,glob,widx,wstat)
select dat, rain,temp,sun,glob , concat( '$wst' ,'_', date_part('year', dat)) , '$wst' from tmp_clidat1; "
# clean up
pgr "drop table if exists tmp_clidat; "
pgr "drop table if exists tmp_clidat1; "
```

It is also possible to use an R script to do the job. The best way is somehow dependent from the actual format of the climate data source. If data are formatted like shown in this example :

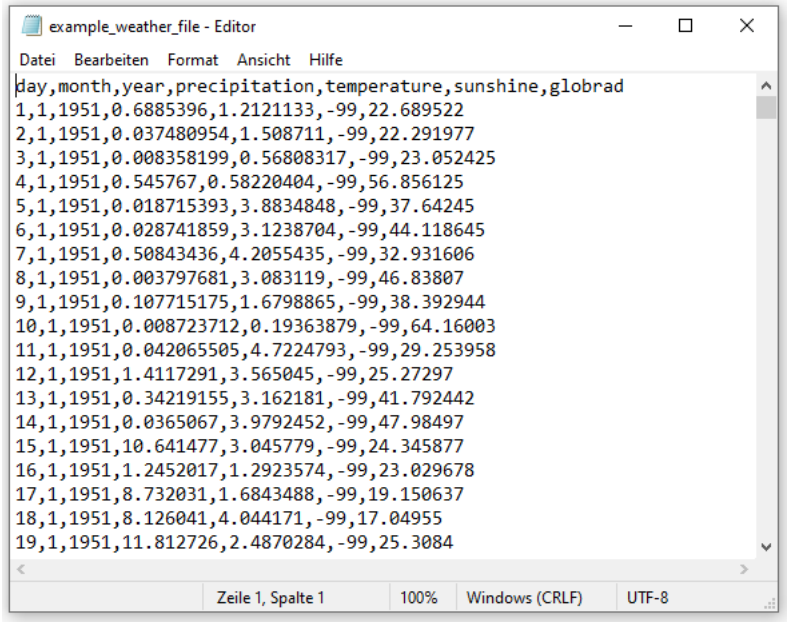


Figure 10: txt-file example in the editor window as template to prepare climate data import.

the following R script can be used to import the data – after adapting it to the special case. All adaptations have to be implemented in the top part after `### REQUIRED INPUT INFORMATION ###` and before `### SCRIPT TO IMPORT WEATHER DATA ###`

It is required to set the variables *workingDirectory* and *filename_weather* (the textfile that should be imported) as well as setting the *global_radiation_unit* to the proper value. Finally, the parameters for the database connection have to be provided (*database, db_user, db_password, db_host,* and *db_port*). The script will import the data first into a helper table before inserting it into cdy_cldat.

```
###################################
##    Import Weather Data Into PostgreSQL  ##
###################################
### REQUIRED INPUT INFORMATION ###
###################################
## required packages - if missing, need to be installed via install.packages('PACKAGENAME') or via Tools -> Install Packages...
require(tidyverse)
require(data.table)
require(RPostgres)
require(udunits2)
## required file path(s) - note that R uses '/' instead of '\' as file separator
workingDirectory        <- "C:/your/datapath/to/the/weatherfile"
## filename to be loaded
filename_weather        <- "example_weather_file.txt"
name_weatherstation     <- "TST"   # limited to THREE character
## global radiation settings
# IMPORTANT: is global radiation in the correct format [J/(cm^2*d)] ?
# --> if not, set 'convert_global_radiation' to TRUE and provide the actual unit as character in vector 'global_radiation_unit' (e.g. global_radiation_unit <- "W/(m^2)")
# --> if in correct format, set 'convert_global_radiation' to FALSE and ignore vector 'global_radiation_unit'
# INFO: unit transformations are done with the function ud.convert() from package udunits2 - see function details for more information
convert_global_radiation   <- T
global_radiation_unit     <- "W/(m^2)"
## database information
database    <- "candy_db_0"
db_user     <- "candy_user"
db_password <- "candy_go"
db_host     <- "localhost"
db_port     <- 5432
#####################################
### SCRIPT TO IMPORT WEATHER DATA ###
#####################################
## set working directory
setwd(workingDirectory)
## load in file (accepted format(s): '.txt')
if(grepl(".txt", filename_weather)){
  file_weather <- suppressWarnings(data.table::fread(filename_weather, sep = 'auto')) %>% dplyr::mutate(V1 = NULL)
}else{
  cat("WARNING:\nNon supported file format - please change data to one of the following format(s):\n.txt")
}
## convert to CANDY-compliant format
file_weather_CANDY <- file_weather %>%
  dplyr::mutate(.,
            datum = lubridate::dmy(paste(day, month, year, sep = "-")),
            nied  = as.numeric(precipitation),
            ltem  = as.numeric(temperature),
            sonn  = as.numeric(sunshine),
            glob  = {if(!convert_global_radiation){as.integer(globrad)}else{as.integer(udunits2::ud.convert(globrad, global_radiation_unit, "J/(cm^2*d)"))}},
            wstat = {if(nchar(name_weatherstation) > 3)
                      {cat(sprintf("WARNING:\nName of weatherstation \'%s\' exceeds limit of 3 characters \n\n", name_weatherstation))}else{tolower(as.character(name_weatherstation))}},
            widx  = paste0(wstat, "_", year)) %>%
  dplyr::select(datum, nied, ltem, sonn, glob, wstat, widx)
## establish database connection
channel <- dbConnect(RPostgres::Postgres(),   dbname = database,   host = db_host,  port = db_port,   user = db_user,  password = db_password)
## DROP cdy_cldat_{name_weatherstation} IF EXISTS
suppressWarnings(dbSendQuery(channel, paste0("DROP TABLE IF EXISTS cdy_cldat_", tolower(name_weatherstation), ";")))
## WRITE cdy_cldat_{name_weatherstation}
suppressWarnings(dbWriteTable(channel, name = paste0("cdy_cldat_", tolower(name_weatherstation)) , file_weather_CANDY, rownames = FALSE, overwrite = TRUE))
## APPEND cdy_cldat_{name_weatherstation} values to cdy_cldat
suppressWarnings(dbSendQuery(channel, paste0("INSERT INTO cdy_cldat (datum, nied, ltem, sonn, glob, wstat, widx)    SELECT *    FROM cdy_cldat_", tolower(name_weatherstation), ";")))
## optional: remove dataframes from Global Environment
rm("file_weather", "file_weather_CANDY")
```

From version 22.8.2 onwards, there is an option to import text files from the GUI (Figure 11). Following requirements must be considered:

- The data set must be homogeneous in terms of radiation data. All records contain global radiation or sunshine duration.
- The text file should have the header **dd;mm;yyyy;temp;rain;rad** , where the columns for dd,mm and yyyy contain integers and temp,rain and rad real numbers. User decide during import if rad is sunshine or global radiation.
- All columns are separated with **;** (semicolon).
- Decimal separator of the real numbers must be same as specified in the regional settings of the computer system.

Figure 11: climate data import from text file

Data import follows three steps. First, click *select a text file* and check the content in the list box below this button. Please check once again the format requirements. Next, decide about the nature of the rad column (global radiation as energy sum, average power or sunshine duration), edit the short label of the climate station as destination for the import and click the *check data* button. If everything went straight, the table below this button shows a summary about the data containing the fields yr (running year), cnt (record count within this year), avgt (annual average of temperature), srain (annual sum of rainfall, rmx (yearly maximum value of rad), rmn (yearly minimum of rad). Mind: at this stage the rad unit are not yet convertet.

If the shown information is suspicious (non-reasonable numbers or incomplete years), there is an option to exclude these years from import. Double click the year and confirm that this data should be excluded. It will then be deleted from the list. If everything is fine, click on *save data to cdy_cldat* to finally complete the import.

Generated weather

There may be cases when a simulation scenario contains incomplete weather data. CANDY can use statistically generated climate data records to replace but only a complete missing year. Parameters for this procedure need to be generated before the simulation by using a selection of years within a complete climate data set.

Soil parameters

The [soil] sheet provides an overview about the available soil data within the database. The detailed soil parameters refer to a specific soil horizon and the respective soil profiles are constructed from these horizons where they are put together with the site-specific dimension. To create a new profile, select *new soil description* from the drop-down menu at the right sub panel and enter the name of the new the soil profile. If the new soil description forms from existing horizons, you can select an item from the horizon pop-down below, enter the depth and click *add horizon*. If the horizon is not yet defined, select *define new horizon*, give it a name and save the new horizon record. It is still possible to edit the soil parameters of each horizon afterwards. The parameters for each soil horizon are listed on the left-hand side of the sheet. The soil texture is mainly characterized by its clay content and the amount of fine particles < 6.3 µm (clay + fine silt). If available, the value for silt (particles ≥ 2 and ≤ 63 µm) should be added. The parameters of a soil horizon include:

active topsoil:	Yes (checked) = tillage horizon, modelling of OM-turnover, No = otherwise
hydromorph:	if checked, horizon is permanently saturated with water
rel. root resist:	modifier (0..1) to adapt default root development to the specific soil
skelett:	stone concentration [Vol. %]
fine particles:	soil particles < 6.3 µm
clay:	soil particles ≤ 2 µm
silt:	soil particles ≥ 2 µm and ≤ 63 µm
bulk density:	bulk density (g cm-^{3})
pore volume:	porosity (Vol. %)
field capacity	water content at field capacity (Vol. %); usually at pF 1.8
perm.wilt.point	water content at permanent wilting point (Vol. %); at pF 4.2
sat.cond.(Ks):	saturated hydraulic conductivity (mm d-1)
heat capacity:	heat capacity of soil substrate (default: 0.16)

related SOC: organic carbon [M. %] is required if the impact of SOM on soil physics shall be simulated. It gives the C_{org} concentration related to the specified physical properties.

Furthermore, the description of a topsoil horizon includes parameters to model the physical protection of soil organic matter that should be fitted to existing observations of SOC and/or Nt. Reasonable values are: lambda ≈ 3000; structural turnover (k_deg) ≈ 0.000001; protection capacity = empty/blank

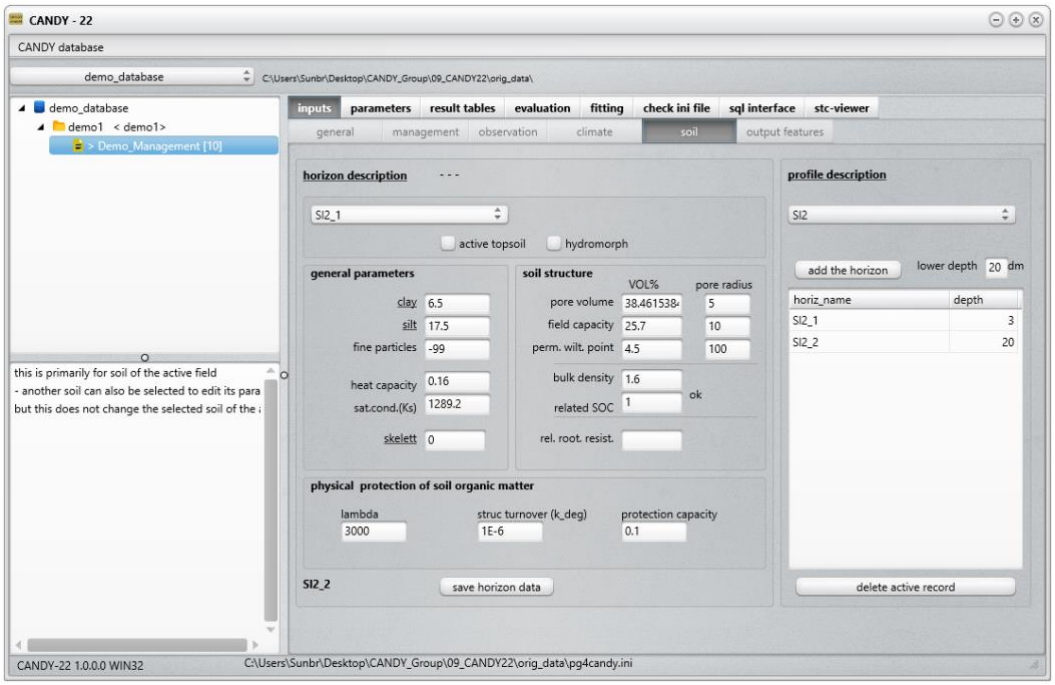


Figure 12: view and edit the soil properties.

Output features

Result data

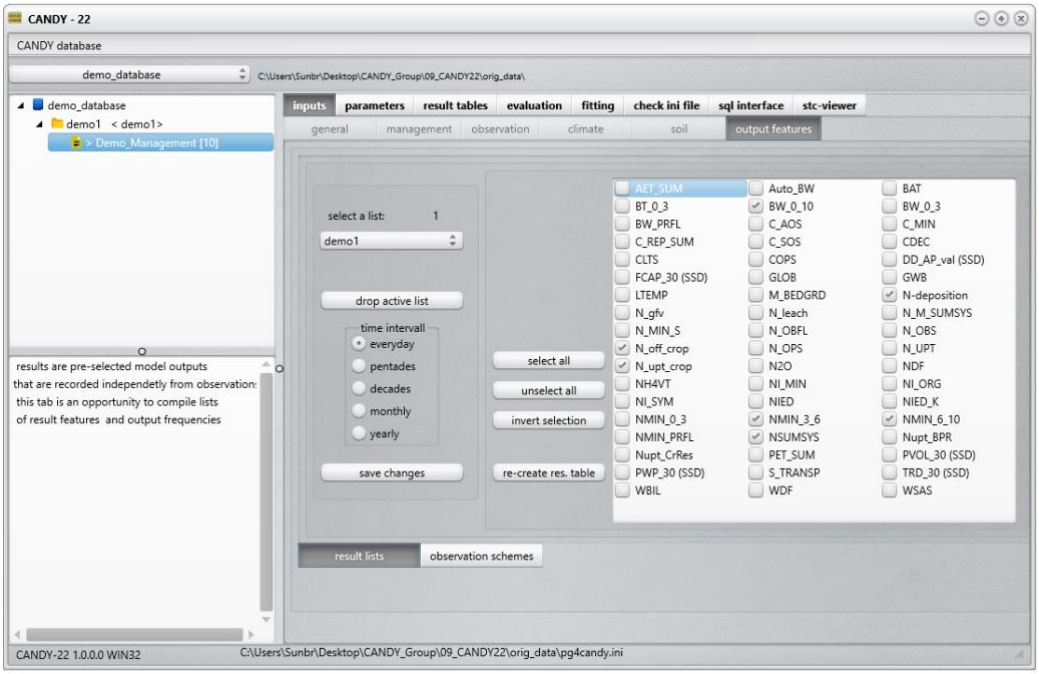


Figure 13: view or edit a schema for result recording

The [output features] sheet contains two sub-tabs to organise the output for results and 'virtual' observations. The latter is explained in the next paragraph. Results are pre-defined and can be recorded in fixed time steps. They can be bundled into different lists and will all be recorded in the same time step during the full simulation time. Select an existing list or define a new one using the pop-down menu. For an existing list, the selected results and the time step will be indicated. Hold the mouse pointer over one short name to see a longer description together with the numeric ID of the result. After editing the result selection click 'save changes' to store in the database. In the *general* tab the list can then be linked to one or more plots and will be used during the simulation run of this plot.

Please be aware that result recording takes some time. Especially daily recording will remarkably increase the simulation time. Therefor, it is recommended to use this only if there are good reasons.

The selectable results are defined within the program code and may change with new program versions. Click 're-create res. table' to update an existing CANDY database to the new result definitions.

Observation data

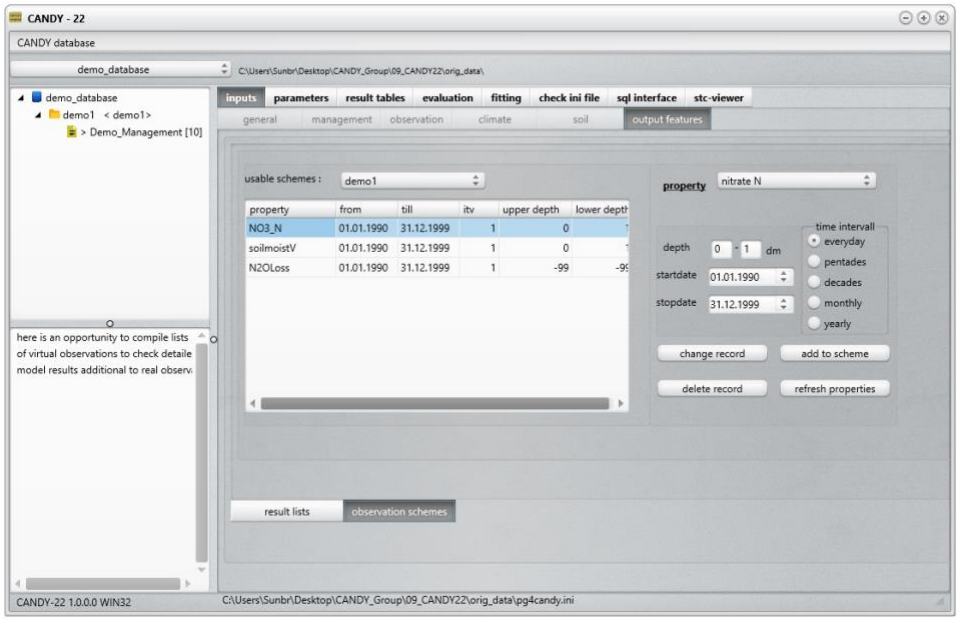


Figure 14: view or edit a schema for 'virtual' observation events to get the corresponding model output

The [observation schemes] sub-tab allows the selection of an already existing scheme or the definition of a new one. By clicking into the table, you are able to view and edit the details for the depth, start and stop date as well as the time interval of recording for your desired observation values. Click *change record* to update an existing record and *add to scheme* for a new defined item. Clicking *delete record* will eliminate the active item.

Model parameters

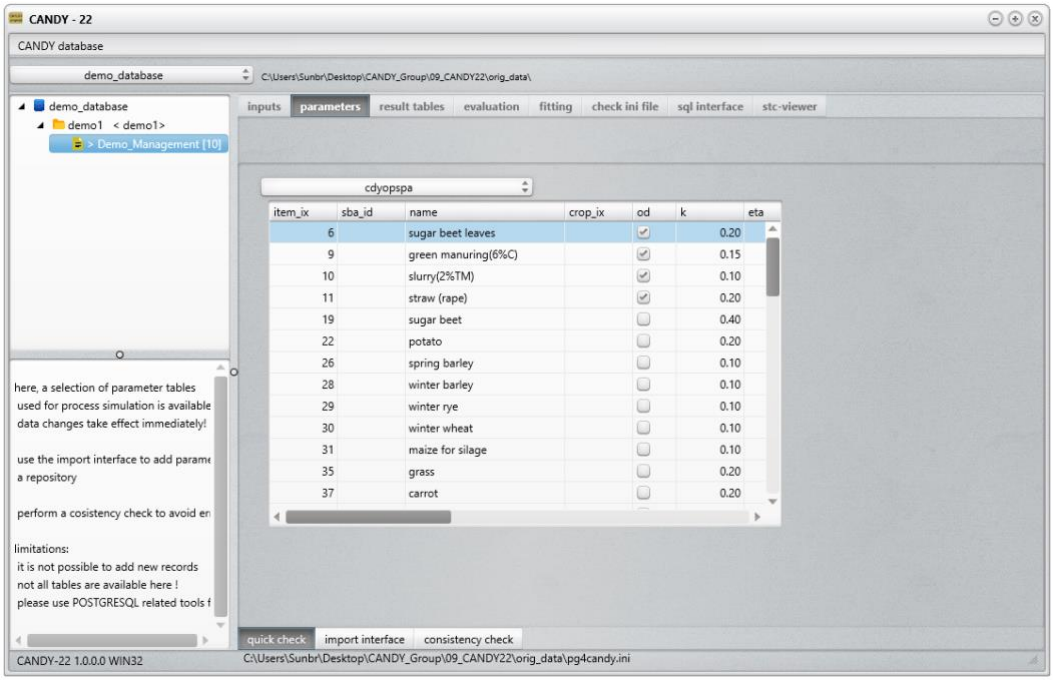


Figure 15: check selected parameter tables

The model parameter tab provides five sub-tabs:

- an opportunity to [quick check] the most important parameters.
- an [import interface] to acquire parameter records from a central (git) or local repository or directly from trans schema. The import from the central repository requires the git software on the local computer.
- the opportunity for a [consistency check] of parameters and management data
- the [inventory] creates a text file (inventory.sql) with all used items in your database formatted as insert statements.
- the [autocrop] tab helps to edit the numerous parameters required for a crop belonging to the 'agro1' model family.

The [quick check] sheet allows for a simple opportunity to view and edit data tables contained within your currently used database. This table will be sorted after clicking the header of any column. Usually, it is more convenient to edit data in DBeaver.

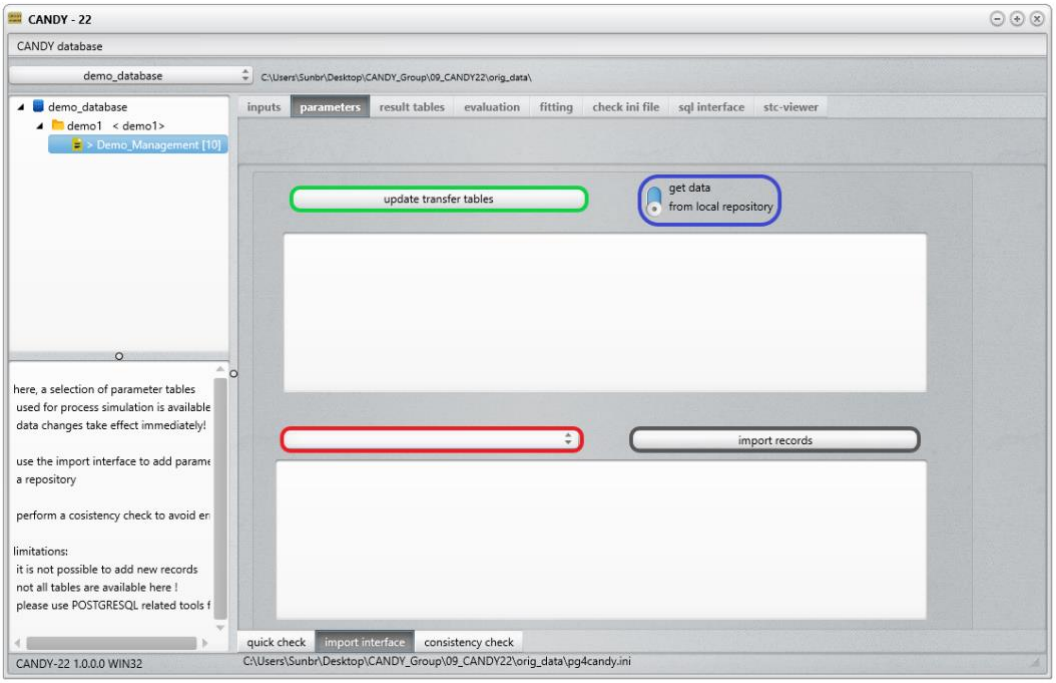


Figure 16: import missing parameter records from repository

The [import interface] sheet allows for the download of new parameters from an external repository, which may be reasonable in many cases. Here is a description how it works. Please check first if git is installed on your computer. You always start with an *update* of *transfer tables* (green frame). On github.com a central repository is accessible, containing SQL scripts that can create mirror-tables in a newly created schema *trans* of your database. For this purpose, a local git repository is created where the files are cloned. The name of this repository is a sub directory of the candy working directory that must be specified in the

configuration file. Parameter updates are performed from this local repository, allowing for an offline update once the repository is downloaded and build. To ensure, that the latest version of parameter tables is available in your local repository, it is recommended to select the remote option for a possible update via the switch (blue frame). A warning message will inform you, that the old repository and the *trans* schema will be deleted before cloning the remote repository, making it reasonable to consider a backup of your local files and the *trans* schema beforehand.

If the *trans*-tables are ready, the desired ones can be selected via the pop-down menu (red frame). The names of the available parameter records are displayed for an import selection (grey frame). If there is already an item with the same id, the user is asked whether the old item should be overwritten, again: considering backups may be reasonable.

The import of a new parameter record of the cdycrops table may lead to an inconsistency regarding missing links (ewr_ix, grd_ix, kop_ix) to the cdyopspa parameters for root and shoot material. *After import, a consistency check is advised, and missing records have to be imported separately!*

Cloning of the remote repository happens automatically on Windows systems while it requires an additional step on MacOS. There are two commands shown on the list box that have to be copy-pasted into a terminal window and executed one after another.

Result tables

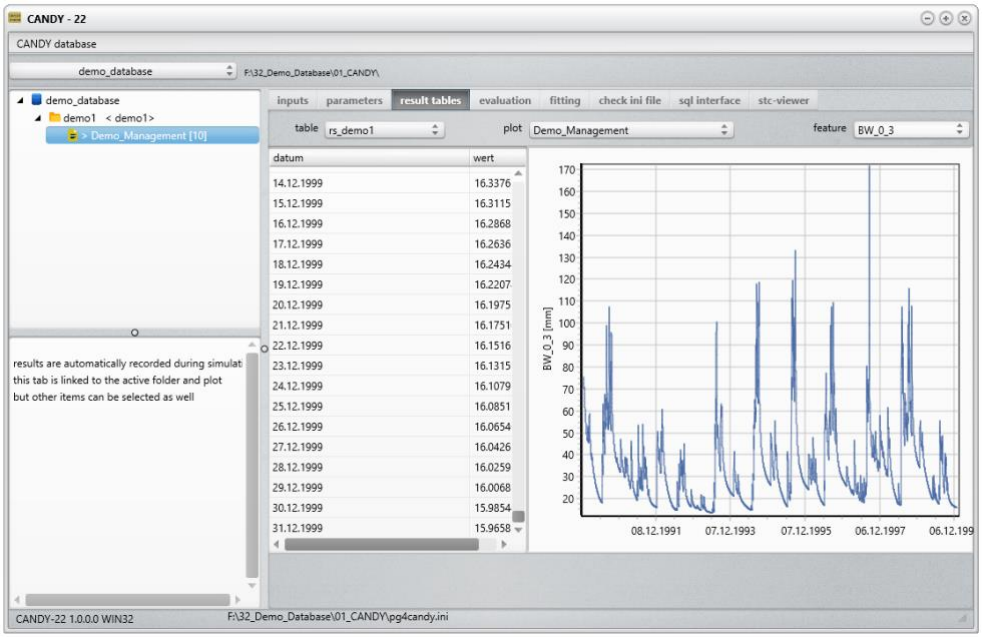


Figure 17: check the recorded result data

The result tables tab allows for a tabular and graphical display of the 'virtual' result output. Result tables are named according to the folder name the field plot is related to at simulation start. After selecting a field plot and a result feature, the table and chart will be updated

according to the selected items. Right-clicking on the table allows a copy to clipboard to move the data quickly to other applications.

Evaluation

The evaluation tab allows for the evaluation of experimental observations, if provided (marked with an asterisk in the pop-down menu). The pop-down list with selectable properties contains only items that were included in the last simulation run (i.e., where the date is within the simulation time interval) and contain model results.

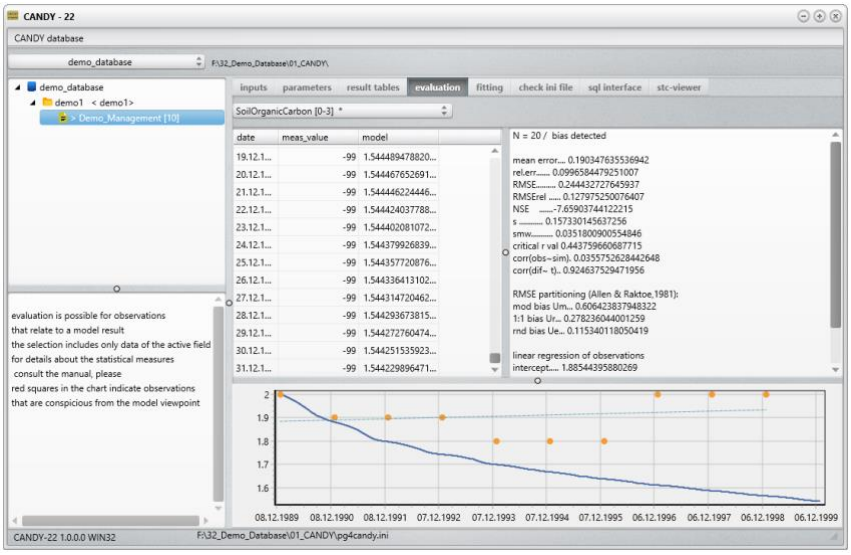


Figure 18: check model results related to observation events

Observed and simulated (predicted) values of the desired experimental trait are shown in tabular form (left hand side) and graphically (bottom). Dots refer to the observed values while the line represents simulated values. A statistical analysis provides indicators to assess the model performance on the right-hand side. A potential bias might give useful information to change initial values.

The statistical indicators are explained below:

mean error	med = residue= difference between model and observation
rel.err	relative error = med/mean(observations)
RMSE	root mean square error
RMSErel	relative RMSE rmse/mean(observations)
NSE	Nash-Suttcliffe model efficiency coefficient
s	standard deviation of the (hypothetical) normal distribution of residues (square root of variance)
smw	standard error of the residues ($\sqrt{\sigma^2/n}$)
critical r value	critical r value (significance limit – here available only if N<60)
cor(obs~sim)	correlation between observation and simulation results
cor(dif~t)	correlation between med (mean error) and time

RMSE partitioning:

Um	model bias part of RMSE
Ur	bias from 1:1 line part of RMSE
Ue	random bias part of RMSE

The partitioning of RMSE into Um, Ur and Ue is based on Allen and Raktoe (1981)

Right-click the graphic to switch between data shown in the grid (evaluation: compare observed and simulated data; simulation: only model outputs). Right-click the data grid to copy the respective data into the clipboard for further usage.

For deeper insight, it is recommended to use an R script to analyse the results. The package RPostgreSQL can be used to connect to a postgres server, as shown in the following code snippets with a sql command that extracts temperature (ltem) and rainfall (nied) from table '*cdy_cldat*' in the database '*cdy_db*' for the climate station named '*R-A*':

```
..
# load libraries
library(RPostgreSQL)
..
# connect to database building connection cn1 with user specific values for host, port, user, password and cdy_db as dbname
cn1<-dbConnect(RPostgres::Postgres(),host='127.0.0.1',port='5432',user='cdy_user',pasword='cdy_go',dbname='cdy_db')
...
# get some climate data as monthly averages
sql<-" select date_part('month',datum) as mm, date_part('year',datum)  as yr, avg(ltem) as lt,sum(nied) as rain from cdy_cldat
    where wstat='R-A' group by date_part('month',datum), date_part('year',datum) "
dst<-data.frame(dbGetQuery(cn1,sql))
# further commands using data frame dst
...
```

Fitting

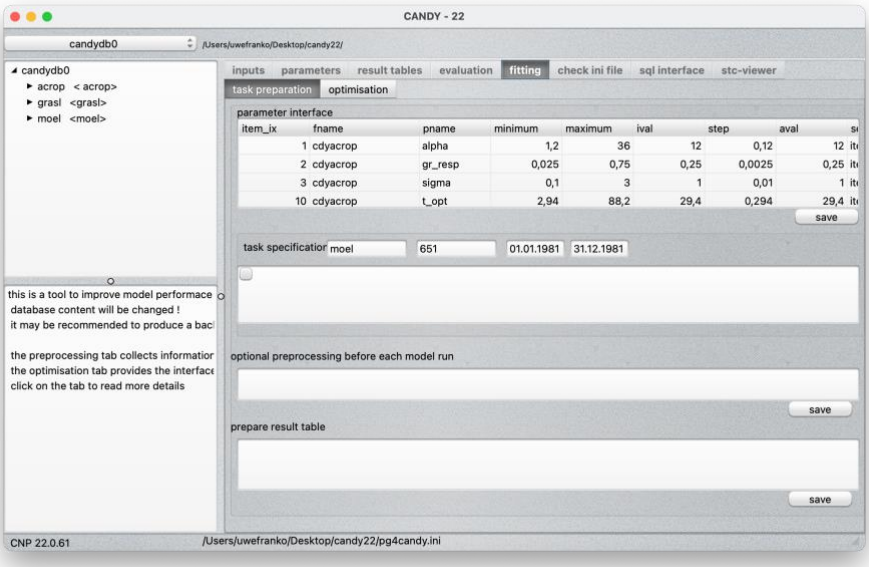


Figure 19: option to calibrate model parameters

The tab fitting is only displayed if the current database contains the schema *opti*. It implements the simplex algorithm routines described in the next chapter.

There are separate tabs for (i) preparing an optimisation task and (ii) starting the optimisation. On the tab '*task preparation*' we find a data grid for the parameters that shall be fitted and three boxes where SQL scripts can/must be loaded.

Each parameter to be fitted is defined by the table, the record within this table, and the field (the column name). It is also necessary to set the initial value at which the optimisation will start, the size of the first step, and the range for the variation of the parameter value. It may also be useful to define an alias name.

The parameter grid displays the contents of the opti.parm_int table. It is possible to populate this grid in a dialogue mode (shown in Figure 20) performing the following steps:

1. select the table containing the parameter
2. select the parameter field from the *field* popup
3. select the field that is used to refer to a specific record in the selected table.

The contents of the table are displayed in the lower part of the dialogue window. A specific record can be selected with a mouse click. The condition is then displayed with a suggestion of the range in which the optimum value should be found.

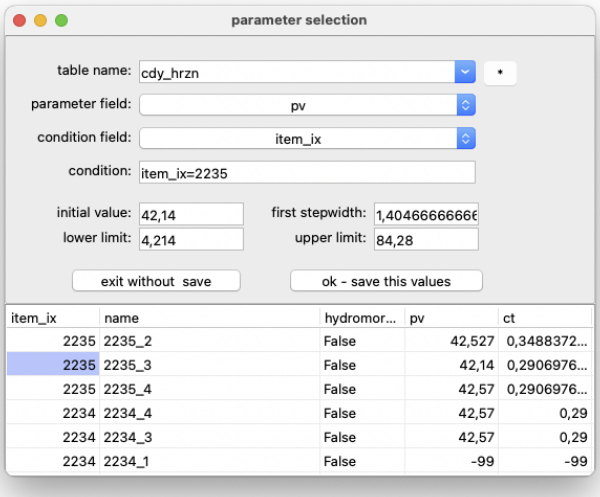


Figure 20: dialogue window to add a new parameter

The SQL-scripts must perform the following tasks:

- The pre-processing step is executed before each simulation run and may not be required in all cases. In this example we want to find a unique initial value for all three plots. For this purpose, an additional table was implemented (outside off this script) in order to store this specific value which is distributed by the pre-processing script to the individual initial values of the three plots in this example.
- The optimisation algorithm compares two rows of data between which there should finally be only a minimum difference. The third script has to produce this dataset outputting a table named opti.simresults containing the fields obsval (the observations), simval (model results) and, depending on the chosen error function, a potential indicator to build groups of data.

The scripts should be tested separately and can be loaded from text files after double-clicking on the individual boxes.

The plots that should be included in the opti-simulation scenario are shown in a listbox together with their time frame. They can be added by after right-clicking them in the tree view from the appearing context menu. The lines of the listbox can be deleted, saved, and loaded from an existing textfile.

The actual fitting will be started from tab '*optimisation*'. It is strongly recommended to first perform a test run to check that the model can simulate the selected scenario. Furthermore, it may be helpful to begin with a sensitivity analysis to be sure that the selected parameters have an impact on the simulation results. Both modes (optimisation/sensitivity analysis) are selected with a change of the switch (marked red in Figure 21).

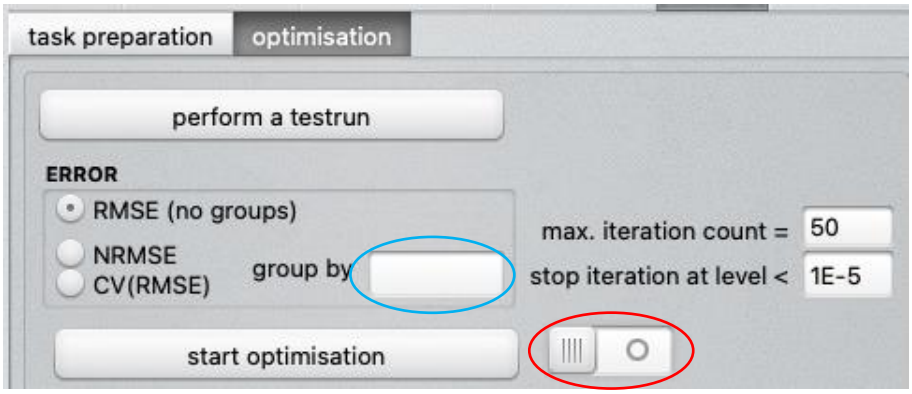


Figure 21: options to start the optimisation.

The goodness of fit between observation and simulation results can be assessed by three different error types that are calculated as follows:

```
-- rmse
 select sqr(avg((obsval-simval)*(obsval-simval))) as hs from opti.simresults

--nrsme
 select sum(hs) from
 (select sqr( avg((obsval-simval)*(obsval-simval))/(max(obsval)-min(obsval))) as hs from opti.simresults group by idx) x

-- cv(rmse)    coefficient of variation of RMSE
 select stddev(hs)/avg(hs) from
 (select sqr( avg((obsval-simval)*(obsval-simval))) as hs from opti.simresults group by idx ) x
```

Both last options can be used to aggregate over different groups. The name of the filed that contains the 'group by' indicator must be included in opti.simresult. The name can be choosen individually and has to be specified in the appropriate edit field (marked blue in Figure 21). The most used option will be RMSE if only one type of observation is used for model assessment.

At the end of the fitting algorithm the results are written in the log-file. The parameter interface (*opti.parm_int*) is updated as well. The estimated values are stores in *ival* and *aval* and also in the original parameter tables. The *error* ε in *opti.parm_int* is calculated as a relative value to be independent form units

$$\varepsilon = \frac{y_{max} - y_{min}}{\bar{y}}.$$

If users make intensive use of the fitting option, it may be convenient to store not only the settings of the last task. To this end, it is required to extend the data model adding two more tables to the *opti* schema. The SQL-script for this extension can be loaded from GIT on the *SQL interface* tab as *store_fitting_task.sql*.

Simplex algorithm

The optimization procedure of the optimizer.exe is based on the approach of Nelder and Mead (1965), also known as downhill simplex method. This numerical method is used to find a minimum of a nonlinear function having more than one independent variable (Press et al. 1992). During a series of steps, the process estimates a local optimum of a problem until it possesses a unique value. The concept uses the geometric figure of a simplex, which, in n dimensions, consists of n + 1 vertices (points) and interconnecting line segments. Hence, in one dimension the simplex is a line, in two dimensions a triangle and a polyhedron for three dimensions. Each point corresponds to a parameter set for which a functional value can be calculated and compared with the other points of the simplex. A decision process selects for each iteration step if the "worst" point W is replaced by a new and (hopefully) better point, while the "best" point is kept. The movement to the optimum is realized by three specific operations: reflection, contraction, and expansion.

For minimizing the function y_i at P_i (with $y_W = \max(y_i)$ and $y_B = \min(y_i)$) the procedure is as follows:

1. Ordering of points according to their function values as in

$$\gamma_B < \gamma_i < \cdots < \gamma_N < \gamma_W \quad (1)$$

Please note that the index "i" is the general index and "W", "N" and "B" describe the worst, next worst and best points.

2. Calculation of the centroid of all points (except for the worst point)

$$M = \frac{1}{n}\sum_{i=1}^{i\neq w} P_i \quad (2)$$

3. Reflection of worst point W as in

$$R = (1+\alpha)M - \alpha W \quad (3)$$

4. If the function value y_R of the reflected point R lies between y_B and y_W, the actual worst point W is replaced by R If y_R is lower than y_B (the current best) there is a new minimum, and the point R is expanded to E: $E = \gamma R + (1-\gamma)M$ (4)

If $y_E < y_B$, W is replaced by E; else the expansion has failed, and W is replaced by R.

5. In the case that the result for the previous reflected point R is between the worst and the second worst case $y_W > y_R > y_N$ a new W is calculated by contracting W using

$$C_R = \beta R + (1-\beta)M \quad (5)$$

If the reflection was not successful ($y_R \geq y_W$), this one dimensional contraction is performed from the initially identified worst point: $C_R = \beta W + (1 - \beta)M$ (6)

Assuming that all previous trials for improvement failed ($y_E > y_W$), new points are calculated by a n-dimensional contraction (shrinkage) towards the best point B and the process starts anew.

$N_{new} = \frac{1}{2}(P_i + N)$ and $W_{new} = \frac{1}{2}(P_i + W)$ (7)

Within the next iteration step N_{new}, W_{new} and B will be re-ordered according to $y_B < y_i < \cdots < y_N < y_W$ and labelled accordingly.

The parameter values used for reflection (α), expansion (γ) and contraction (β) are $\alpha = 1$, $\beta = 0.5$ and $\gamma = 2$.

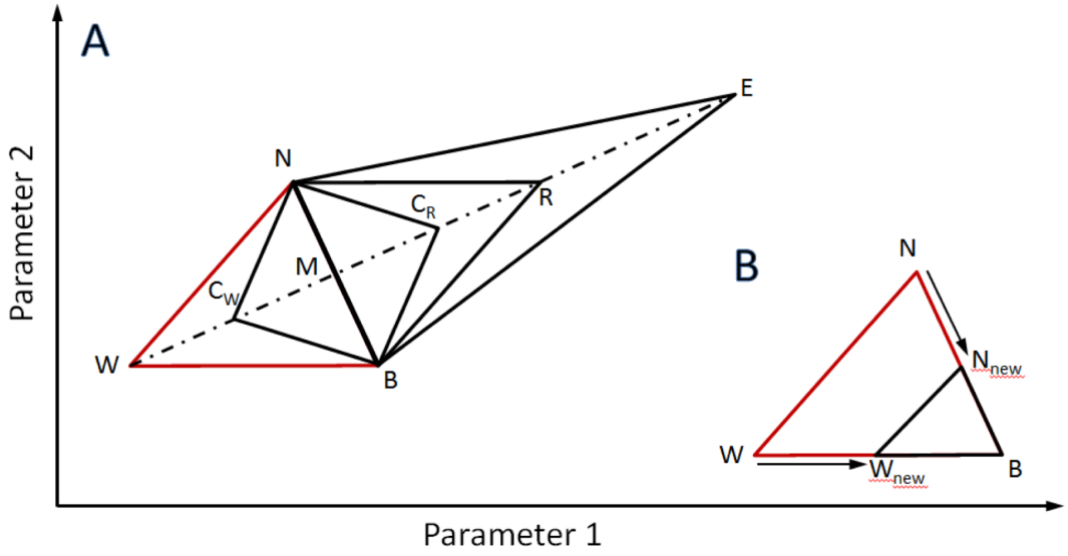


Figure 22: Basic triangle simplex is BNW (red) with the worst point W, next best point N and best point B. M is the centroid of all Pi without W, thus of B and N. R is a reflected W. E is reflected and expanded W. CR and Cw are contraction points from R or W depending on the result of the reflection. Adapted from Bezerra et al. (2016) and Nelder and Mead (1965). B: If the direction is wrong and reflection, expansion and contraction failed, a multiple contraction of the basic triangle BNW towards the best point B takes place and results in a new triangle BNnewWnew.

Check ini file

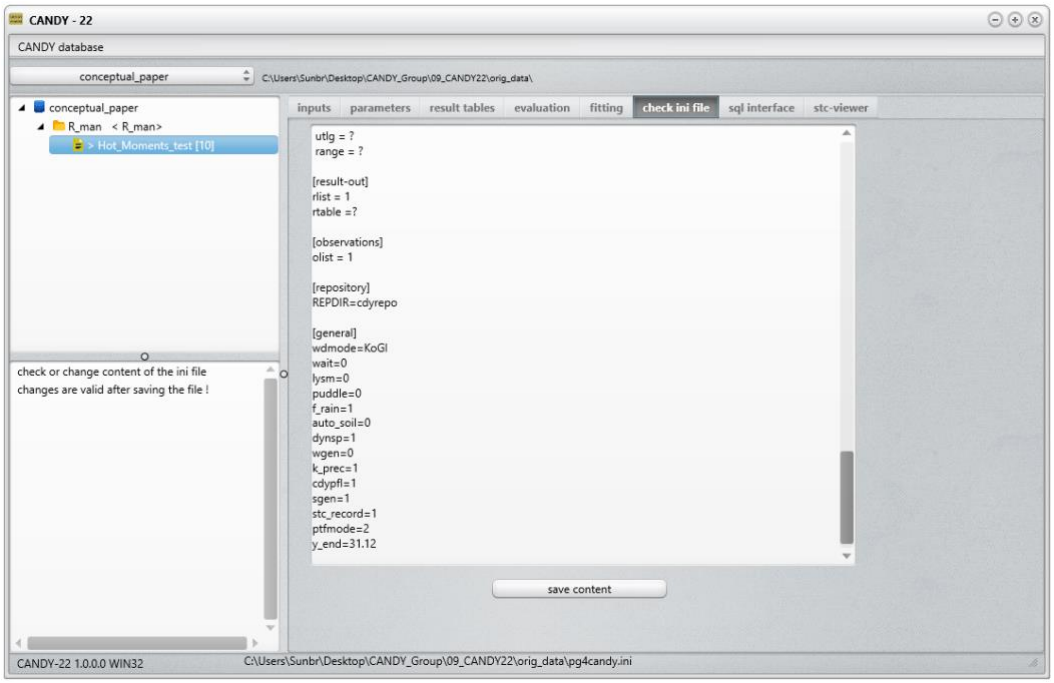


Figure 23: check and edit the configuration file pg4candy.ini

Use this tab to view and edit the content of the configuration aka ini-file. Modifications will be active after the content is saved successfully.

The first section specifies a list of the required drivers (usually only one for your actual system). The section [databases] provides a list of all databases that are imported and known by CANDY. The names here are alias names that may differ from their definitions in postgres. The first database of this list is activated and loaded as default after starting the model. Each item in this list requires its own section where the required parameters for a database connection (host, port, user + password) have to be specified. Here, the dbname is the database name on the postgres server.

Several settings that can modify model behaviour such as the selection of another pedotransfer-function or the possibility to dynamically calculate physical soil parameters, can be placed after the database sections in section [general]. The [general] content is explained in the chapter about model settings (see Model settings).

From version 22.7.6 it is possible to have different settings for each database. To do this, the individual database sections must contain the settings that differ from the general settings.

For some settings the parameter name is slightly different from the variable name that is used in source code and in table *tmp_setting* within the exchanged SQLite files:

lysm -> lysimeter; f_rain -> rain_coeff; k_prec -> nied_korr ; y_end -> yr_end;
outfr->rslt_freq; stc_recor -> stc_rec; dynsp -> dyn_sprm

SQL interface

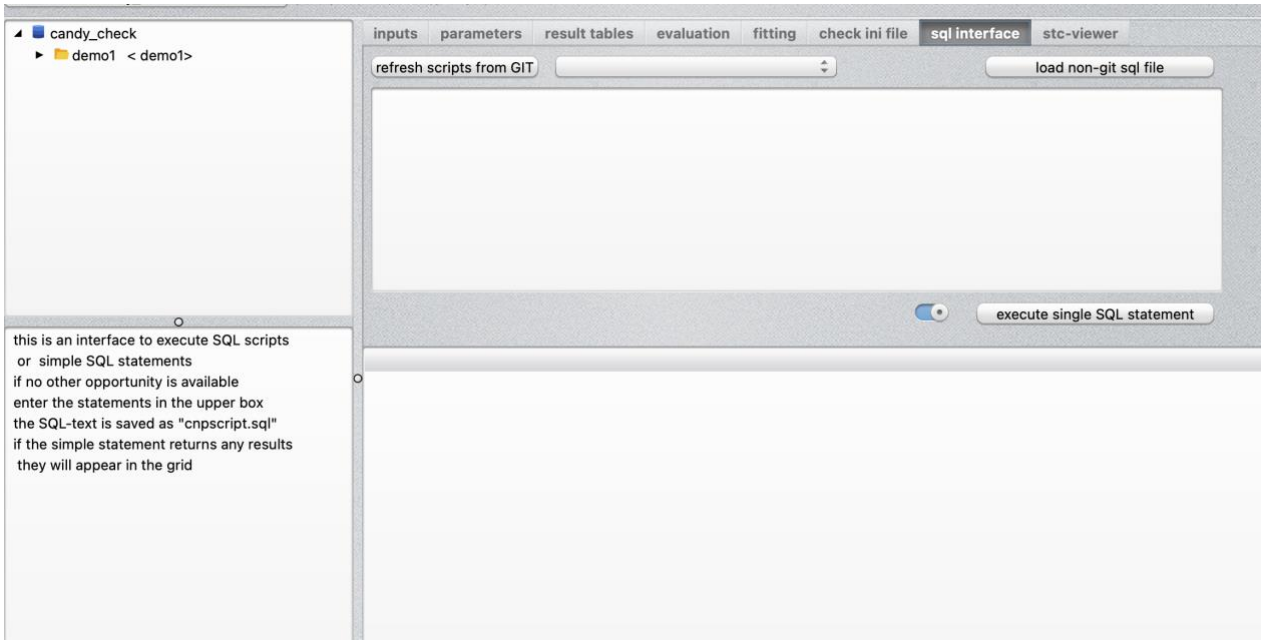


Figure 24: simple interface to apply SQL commands and scripts

In some situations, it may be convenient to execute SQL commands or even SQL scripts directly from the user interface. Commands can be executed by either directly typing SQL into the upper listbox or import complete, externally prepared, scripts. This includes an import of published scripts from a central GIT repository. Before using this option, please check first if git is installed on your computer.

If a single SQL command calls a data table it will be displayed in the data grid below (e.g. 'SELECT * FROM cdycrops' shows the complete cdycrops table).

stc viewer

If you did not activate the option for new generated initial conditions (configuration option in the [general] section of the ini- file), the system can initialize from a valid record in the status file (S__*.stc). Status recording is activated setting stc_record=1 in the [general] section of the ini-file (pg4candy.ini; s.a.). During a simulation a new status record is written after each year and additionally at the end of the simulation scenario. Opening the tab stc-viewer will show the available data for the active plot.

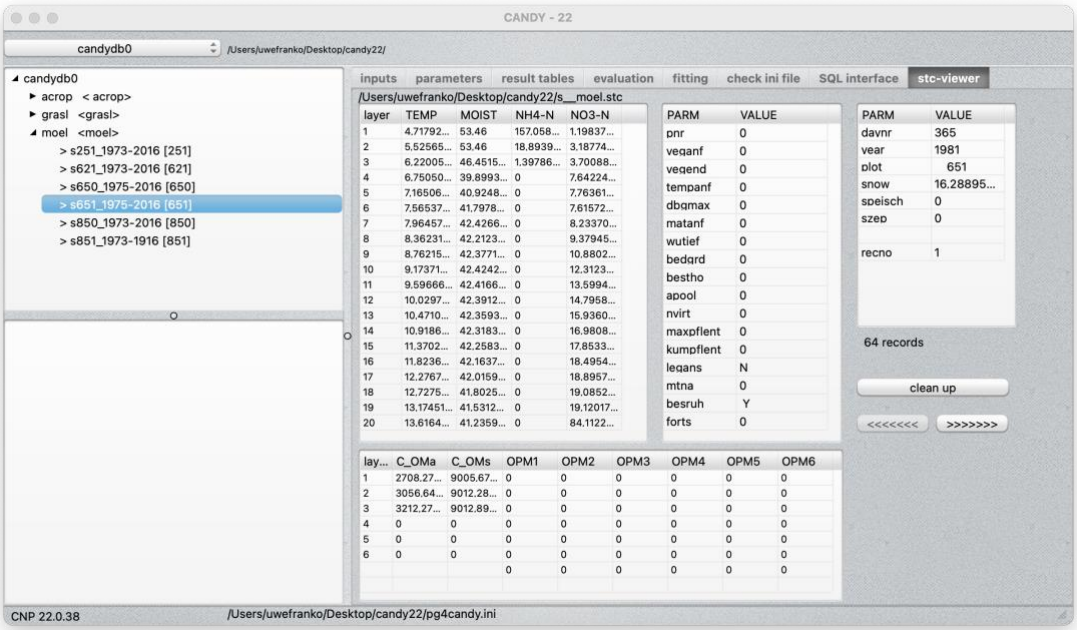


*Figure 25: view the content of *.stc files*

Main menu

Following actions are available from the main menu:

- Add another database: Here you can add a postgres database that is already existing on the server, create a new database from an existing dump file, or create an empty database with or without some parameter records. The last option could be helpful if you want to import data from a data export file (*.sqlite) of another CANDY user. Please remember to put in you master password for postgres (that you selected during the server installation) if it is different from 'postgres' which is here provided as default.
- Export data: This feature supports data exchange between CANDY users. The complete data set of a selected folder including the active settings will be exported into one SQLite database file that can easily be transferred and used as import source. The table *tmp_setting* contains the currently active settings for the active database.
- Import data: This is the counter part of the above-mentioned feature. After selecting a sqlite file CANDY will connect to the corresponding SQLite database. Only if this step is finished the move button (*move to trans schema*) is enabled. After clicking the move button, all tables from the SQLite database are transferred into the trans schema of your active postgres database. During this process it is checked if the settings of the source candy system differ from the current settings. Differences will only be shown. If considered necessary, the ini-file has to be updated manually.

  The user tables (cdy_***) are then shown in the pop-down list and will be included one after one in your active CANDY database by clicking the import data button. There may appear some error messages that not necessarily hint to severe problems. So, you should first check the result before repeating a step. If the management table (cdy_madat) is handled, the system checks if all required objects (parameter records) are available and will try to import missing records from the trans schema. Sometimes you may have to decide if it is better to keep an existing record or to replace it with a new one from the data import.
- Info: Gives information about the activated date format and checks the software version available on the web (if internet access is provided).

Context menus

Right-clicking on items of the tree view opens individual menus allowing the selection of available actions on that particular level.

Root options (uppermost (database) symbol in the tree view):

- add a folder: shows a dialog box to edit the name of the new created folder; simultaneously a new plot within this folder is created.
- run simulation: runs model simulations for all plots contained in the database
- refresh connection: database connection is refreshed and data content is updated. This may be required after changing some specific data.

Folder options:

- add new plot: a new plot (named new) is inserted within this folder
- multi-plot simulation: start sequential simulations for each plot in the folder using the plot specific simulation parameters

Plot options:

- move plot to folder copy all data into a new plot in another folder
- copy active plot copy all data into a new plot of the same folder
- delete active plot deletes the selected plot.
- add to optimiser covered adds the time frame of the plot to the simulation scenario by optimisation runs
- run simulation runs a single model simulation for the active plot
- run periodic simulation repeats management actions according to the *nrep* value specified in the [scenario] section of the ini-file

Simulation runs

Single simulation run

After you have checked all the settings, right-click on the plot and select [run simulation] to start the CANDY model. Run-specific messages as well as a progress indicator will be displayed in the listbox below the tree-view. Press [pause] (or depending on your hardware: [FN]+[B] , [FN]+[Strg]+[B]) on your keyboard to stop the simulation. The behaviour of the model can be changed by some settings within the configuration file as described in the chapter model settings.

Periodic simulation runs

In this mode, CANDY repeats management actions according to the *nrep* value specified in the [scenario] section of the ini-file while continuing with climate data.

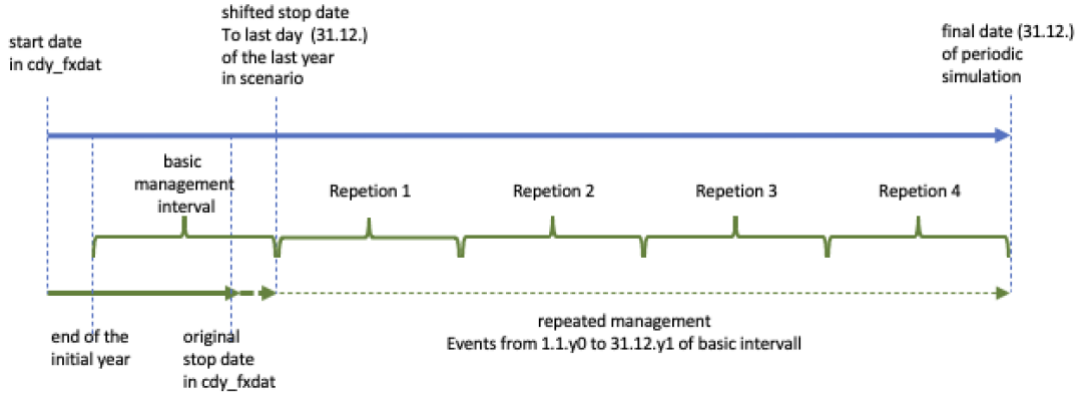


Figure 26: schematic representation of periodic simulations for nrep=4. The repletion starts at the beginning of the first day of the second year and ends at the last day of the last year of the basic mananagement interval. This means, that the time from the original stop date to the end of the last year is filled without events (like fallow). While management is repeated in multiple cycles the timeline for results is represented by the time scale of climate data.

Batch file settings

CANDY simulation can be performed without opening the graphical user interface using batch file calls. Therefore, some specific parameter settings in the batch file are required.

The general call on windows is: <Data path>\CDY22_GUI <parameters>.

On MacOS the call is a little different:

- open the terminal
- change to the candy directory
- open the application: open -a cdy22_GUI.app --args <list of parameters>

Batch call parameters

The model run is controlled by several parameters that are specified in the command line.

Additionally, the settings in the [general] section of the ini-file will modify the model behaviour additional to the parameters of a batch call. The [general] settings replace the switches of older CANDY versions in the registry of a Windows PC. Registry settings are now ignored and may be removed. Currently, CANDY will make no entry in the Windows registry.

attribute	meaning
A=	start date (A=dd.mm.yyyy) (same as *startdat* in CDY_FXDAT)
E=	end date (E=dd.mm.yyyy) (same as *stopdat* in CDY_FXDAT)
P=	name of soil profile (link to *standort* in CDY_FXDAT)
FN=	folder name (link to *fname* in CDY_FXDAT)
RL=	result list (number)
OL=	observation list (number)
SNR=	plot index (snr in cdy_fxdat)
SUB=	sub index (utlg in cdy_fxdat)
W=	short name of the climate station (sss) (link to *wetter* in CDY_FXDAT)
SRF_WC=	capacity of surface water retention; default = 5; to activate set puddle=1 in ini-file
R=	result table name (will eventually be created)
PSET=	selection of general parameters
CNF=	Ini-file with configuration parameters
DB=	Postgres database
GO	Start the simulation run immediately using the batch call parameters
AUTO AUTO=tabname	Simulates all items stored in table cdy_automat or in the specific table name of the database given by DB
NREP=	Number of scenario repetitions
BG_RUN	Optional, suppresses GUI window
PPQ=	Sql script processed before simulation start
POQ=	Sql script processed after simulation is finished
OID=	Optional identifier for result recording in batch mode
ST=	Optional reference to a binary status file where a certain record should be used for initialisation Syntax: ST=<complete filename>#<record number> example: ST=my_stat_file.stc#325

Be aware of correct using, there should be no space before and after the equal sign "=". Also be aware of setting spaces between each attribute.

An example of a batch-file setting is given in Figure 26, where a master batch-file (Figure 26 (1)) defines the command "**call xxx.bat**" (in this example: *call 02_run_candy.bat*) to initiate a model run, which is described with its settings in the slave batch-file (Figure 26 (2)). The name of the slave batch-file equals the name within the master batch-file (purple). The term "call" therefore assures that the start of the model call in the next row is initiated. The terms "**%?**" (where ? stands for an integer) in the slave batch-file work as placeholders, which automatically call a certain attribute that is described within the master batch-file (Figure 26 (1), blue, red, green and orange). Any number of attributes can be defined, their position in the master batch-file corresponding to the integer used to call them in the slave batch-file following the "%". This might be used for different states of the system, e.g. when different soil profiles, climate situations, plot index' or any other variable condition will be addressed, as multiple "**call xxx.bat**" entries can be included in one master batch-file.

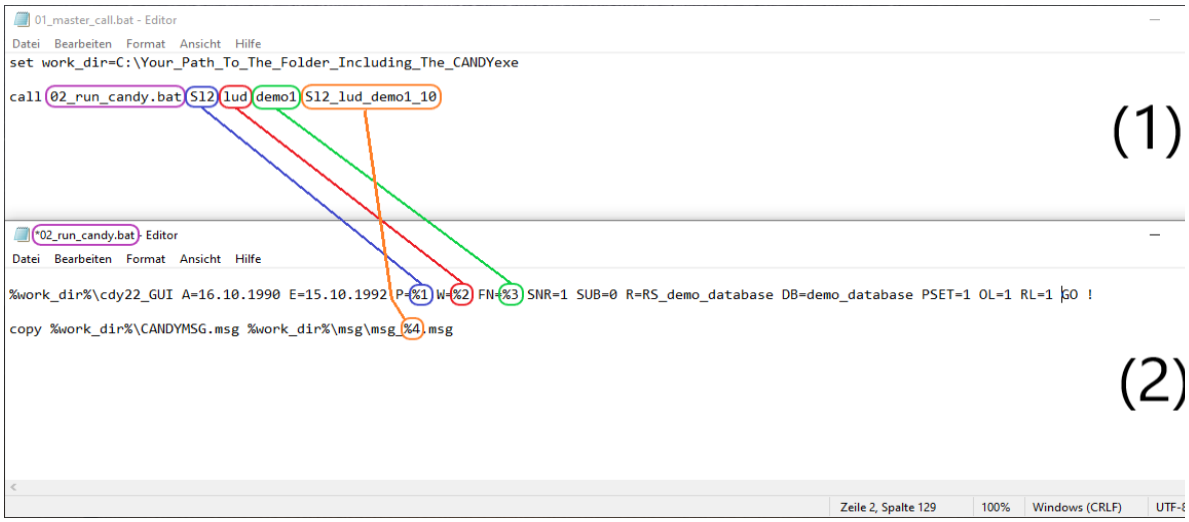


Figure 27: Example for running candy22 via batch calls from master (1) and slave (2). The latter is using parameters defined in the master, in this case information about soil (%1, blue), weather (%2, red), management/file_name (%3, green) and an "index", combining the essence of the previous given parameters with the plot- and subindex (%4, orange). Besides the mandatory "candy call" in the slave batch-file, additional scripts can be executed before/after candy is/was called.

If the simulation is controlled by batch runs, it may be useful to apply pre- or post-processing SQL scripts. For more flexibility, those scripts may contain wildcards (like '$soil') that can be updated into the actual name of the soil profile (like 'St2') embedding a windows powershell command within the batch-script:

… some batch commands

PowerShell ((get-content poq.tmp) -replace '$soil','St2') > set-content > poq.sql

cdy22_gui A=……<all parameters> POQ=poq.sql

…more batch commands

In this simple example is the text in poq.tmp (name is exchangeable) analysed and every occurrence of $soil is replaced by St2. Afterwards the new content is stored in the file poq.sql which is then used in the candy call as post processing script. If poq.sql is already existing, the new file will be created without warning.

On macOS can be used the tool sed like shown in this example where the setting for puddle mode is changed in file pg4candy.ini:

echo "switch on puddle mode"

sed -i '' 's/puddle=0/puddle=1/g' ./pg4candy.ini

#show the switch for puddle mode

sed -n 's/puddle/puddle/p' ./pg4candy.ini

Model settings

Model settings can be also imported/configurated from the configuration (ini) file. If an option is not specified, the default setting will be active.

[scenario] (here, default value are always '?', input is usually not required)

range	folder name
snr	plot number
utlg	sub plot
soil	soil profile
clim	climate station
t0	start date
t1	stop date
nrep	if *nrep* >0 the management scenario will additionally be repeat *nrep* times
lcyr	ultimate year to stop the periodic simulation runs

[general] *(Note: These settings can be overridden in individual database sections)*

pset (default: 1)	selection of general parameters (item_ix in cdyaparm)
wdmode (default: KoGl)	water dynamics following Koitzsch/Glugla approach
wait (default:true)	stop after simulation run
lysm (default: false)	lysimeter mode=no runoff
puddle (default: false)	surface retention of runoff water; capacity can be set by batch parm SRF_WC (default value=5 mm)
f_rain (default: 1)	modifies rainfall intensity (factor)
auto_irrg (default: false)	automatic irrigation (actual water inputs are en-/disabled in management)
auto_soil (default: true)	model completes missing soil properties
dynsp (default: false)	dynamic soil physical parameters d
wgen(default: false)	use weather generator
k_prec (default: true)	rainfall data biased, correction required
cdypfl (default: true)	apply internal crop module of CANDY
sgen (default: true)	generate initial conditions
stst (default: true)	initialize OM pools in steady state
stc_record (default: false)	record the system state at year endings
y_end (default: 31.12)	last day of a year
outfr (default: 0)	output frequency of result recording 1: daily 2: pentads 3: decades 4: monthly 5: yearly
rs_at_hrv (default:false)	if true: additional result output at harvest dates
ptfmode (default: 2)	switches pedotransfer mode for auto_soil 1: Brooks-Corey /Rawls&Brakensiek 2. Vereecken/van Genuchten 3: Lieberoth 4: Zacharias/van Genuchten

[observations]

olist (default: 0)

[result-out]

rlist (default:0)

rtable (default: ‚?')

[repository]
REPDIR name of local repository i.e. cdy_parm

CANDY error handling

This candy version is rather new. While tested by the developer there is a certain probability for errors. If a re-connection to the database has no effect, restarting the application may help. A clear report about the circumstances that led to the error will help to identify and fix possible bugs.

A number of typical errors can and will be recognized by the model during a simulation run. Normally the model stops showing a message box with some hint to the identified problem and terminate by closing this box. In some cases, this hint is only showing the last reason for the model to crash. It is well possible that the real cause for the termination is not directly shown - so check the data and think again to find the solution for this problem.

Checklist to avoid errors

- Are all required parameter values for a modelling run defined in the database?
- Do you have missing values or gaps in your climate, soil or management data?
- Are all records in table *cdy_mvdat* complete ?
- Is your soil profile ok ? Horizon depth has to be given in **dm** NOT cm !!!
- Have your observation data the required units ?
- Is the order of actions in your management data correct?
- Check availability of climate data for your management scenario.
- Make sure that there are no inconsistencies in the parameters (see Figure 15)

Appendix A: Postgres installation

This section guides new users through a correct installation process of postgres. As already described above, postgres can be downloaded from

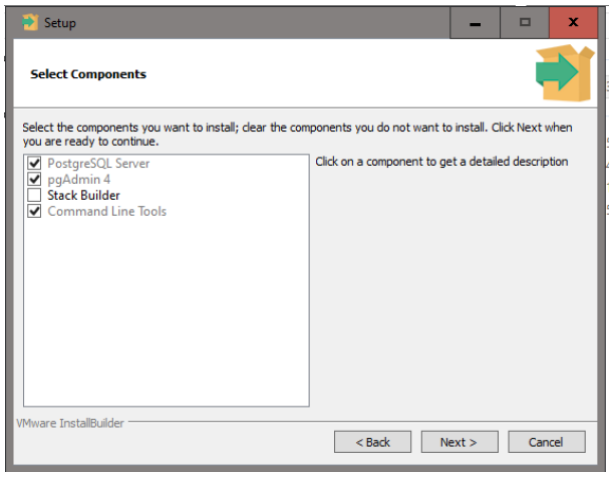
https://www.enterprisedb.com/downloads/postgres-postgresql-downloads.

After downloading and starting the installer and providing an installation directory, you are asked to select the components you want to install. Make sure to check at least “PostgreSQL Server” and “pgAdmin 4” as seen in figure 27 a). In most cases Stack Builder is not required and Command Line Tools may be usefull only for experienced users.

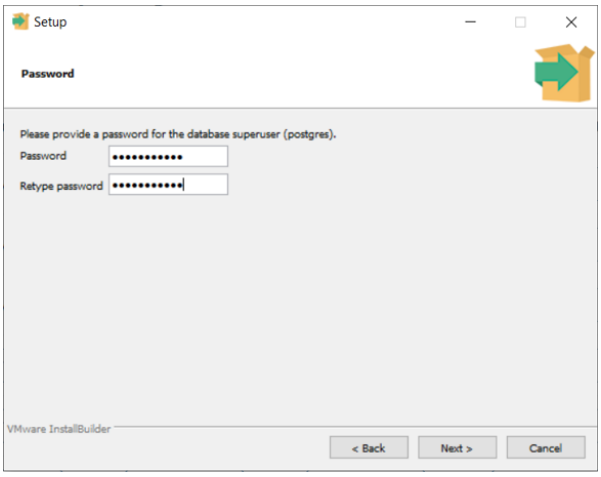
Once you selected your installation directory, you are asked to provide a **password** for your database superuser, which is by default named “postgres”. **It is important that you remember this particular password, as it is required to log into and access your databases (figure 27 b)). It is quite usual to select “postgres” also as password - as long as the database server is running on your local computer (localhost with ip 127.0.0.1).** Next provide a port number; by default use “5432” as seen in figure 27 c). Before postgres is installed, a pre installation summary is provided like that in figure 27 d). It is advised to remember the information given.

By clicking on “next” the installer should proceed and successfully complete your postgres installation. You are now able to proceed with the creation of a new database with the cdy_prepare application as described in chapter “First preparations before modelling begins”.

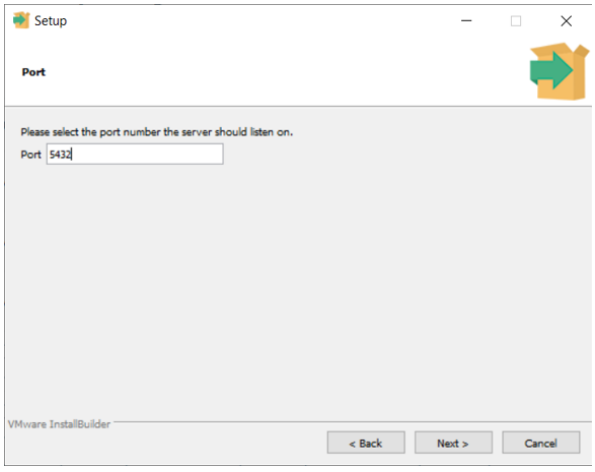
a)



b)



c)



d)

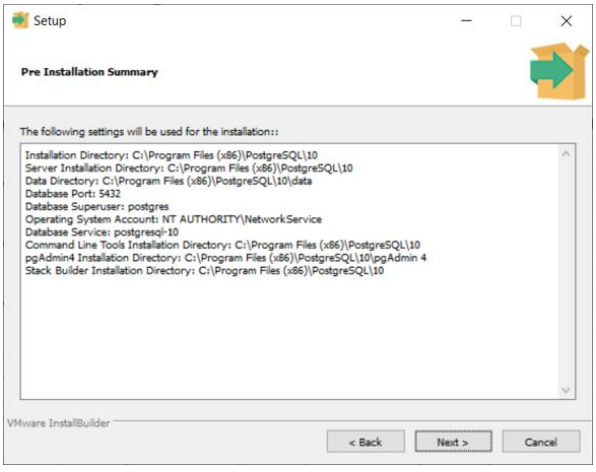


Figure 28: a) selectable components for postgres installation; must select PosgreSQL Server & pgAdmin 4; b) provide a password for the default superuser; c) provide port number (default 5432); d) installation summary. Check if port and directories are as desired.

Appendix B: main data model

This is a quick description of the data structure that contains the biggest part of user data. There is a more complete description of database issues in the document ***CANDY database***.

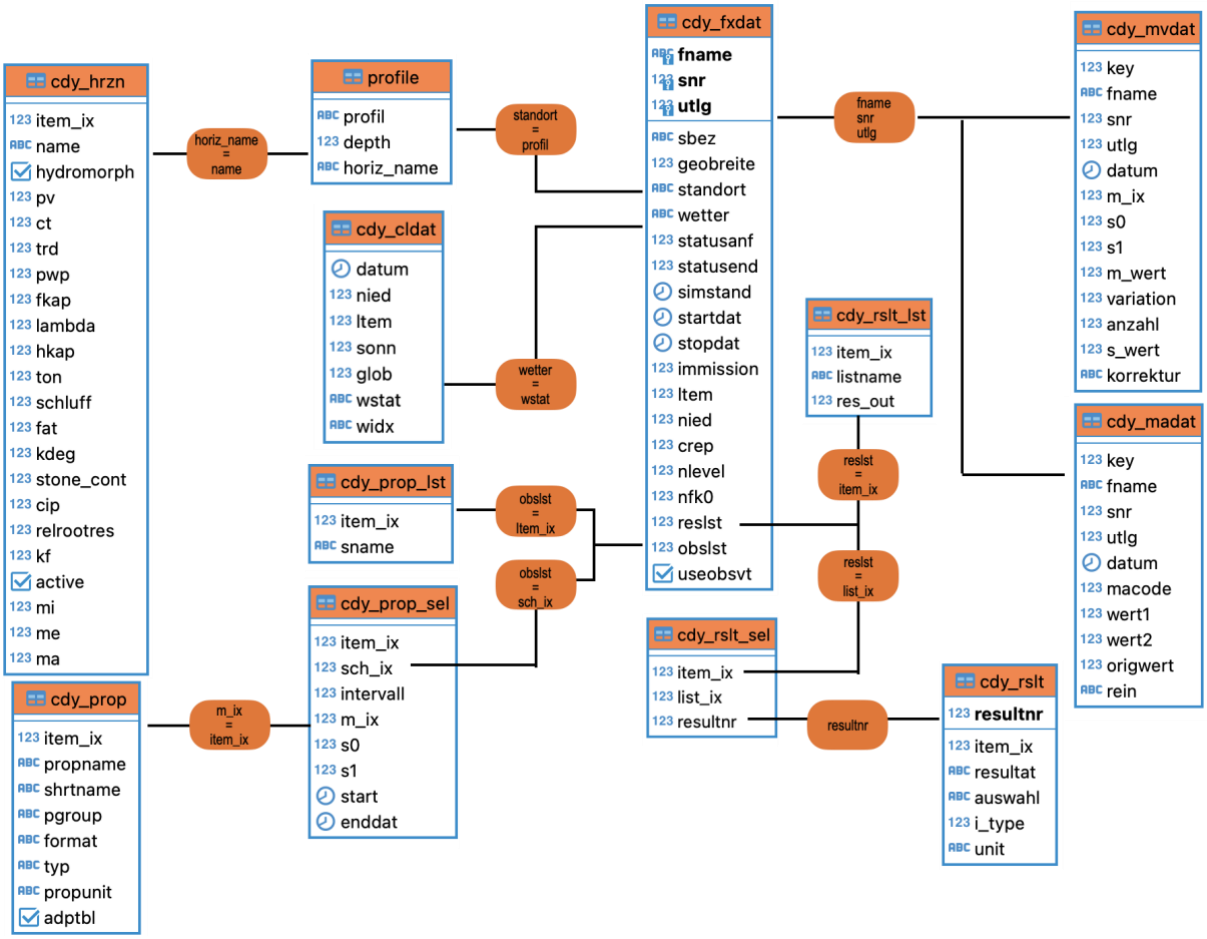


Figure 29 :data model showing links between main user related tables

All plots are defined in table cdy_fxdat and are unique in fname, snr utlg. The same items link the plot with the management data in cdy_madat and experimental data in cdy_mvdat that contain the input data given by the user. The item "standort" gives the relation to a soil profile that is linked with a number of horizons in cdy_hrzn where the detailed properties are specified. The item wetter relates to a climate station (wstat) in cdy_cldat where all the daily climate data are stored. Furthermore, two different output channels can be linked with the plot. There is a number of result-items for a fixed depth that are all defined in table cdy_rslt and are organised in different lists with a selected output frequency. All members of each list are compiled in cdy_rslt_lst. The other out channel was primarily designed model outputs that are related to experimental results. *This is activated when useobsvt is TRUE*. In addition to these real observations, there may be 'virtual' observations. Again, the observations are defined in a list of different elements, each with its specific depth (s0, s1), frequency and time interval for recording. This list is represented by table cdy_prop_lst that links to cdy_prop containing all the selectable properties and cdy_prop_lst providing a name for the selectable lists.

CDY_FXDAT

Basic information as fixed data with general description of each homogenous simulation object.

attribute	meaning	unit
sbez	plot name	
fname	folder name	
snr	plot number 1...9999	
utlg	subplot number 0...9 (optional, only if meaningful)	
geobreite	geographic latitude; only used to transform sunshine duration into global radiation	
standort	pointer to profile in PROFILE	
wetter	pointer to wstat in CDY_CLDAT	
statusanf	First record number in status file (*.stc)	
statusend	Last record number in status fiel (*.stc)	
startdat	intended start of simulation	
stopdat	intended stop of simulation	
immission	annual atmospheric N deposition	kg/ha/yr
ltem	long term average of air temperature	°C
nied	long term average of annual rainfall	mm
crep	Carbon input level ; default: 9	dt/ha
nlevel	Initial N in total soil profile	kg/ha
nfk0	Initial filling of plant available field capacity (rough guess); default: 100	%
reslst obslst useobsvt	Number of the list with recordable results Number of the list with virtual observations TRUE: values from cdy_mvdat will be included; **FALSE**: no recording and **NO ADAPTATIONS** will be applied	

The fixed data contain some information used to initialize the model:

- *nfk0* describes the relative filling of the water storage. If the starting point is in winter or early spring 100% is usually a good guess
- *nlevel* wis used to initialise the nitrate N pool.The actual nitrate amount of a soil layer (1 dm) is set to *nin0/layercount*.
- *crep* describes the level of carbon reproduction before the simulation start. 9 dt ha-1 is a good guess for normal conditions. The resulting Corg value depends also on ltem, nied and soil texture
- *ltem*, *nied* : The BAT value of the pre-simulation time will be estimated from the values of *ltem*, *nied* and soil texture. These values are of course only rough estimates to make the model run. Much better tuning is possible using appropriate observation values to adapt the model to initial conditions.

CDY_MADAT

Management activities on field – find more instructions on wert1,wert2 and origwert in the next table

attribute	meaning	unit
key	Unique record identifier	integer
fname	folder name	string
snr	plot number 1...9999	integer
utlg	subplot number 0...9 (optional, only if meaningful)	integer
datum	date of activity	date
macode	coding of activity; ointer toaction_id in CDYAKTION	integer
wert1	coding of used item (i.e. crop, fertilizer etc.); pointer to appropriate parameter table (see table 1)	integer
wert2	Intensity/quantity	float
origwert	Amount (original value)	float
rein	priority of user values	Y / N

remarks

rein = "N" – N uptakes and C inputs are calculated from model parameters

rein = "Y" – user values for N uptakes and C inputs will be used instead of model parameters

Table 1: Quantities and qualities of original value for different management codes

macode	activity	implementation	wert1	wert2	origwert
0	fallowing; ploughing up	crop growth stops, N in crop is returned to soil as OM indicated by the *grd_ix*	device index (*itemix* in CDYDEVPA)	tillage depth [dm]	tillage depth [cm]
1	emergence	CANDY_S initialisation point; begin of crop growth	crop index (*item_ix* in CDYCROPS)	expected N uptake [kg ha^{-1}]	expected natural yield in standardised FM * [dt ha^{-1}]
2	harvest (by-product **is** removed)	crop growths stop. Crop residues are added to soil	crop index (*item_ix* in CDYCROPS)	real N uptake [kg N ha^{-1}]	real yield in standardised FM * [dt ha^{-1}]
3	organic matter application	according to parameters in CDYOPSPA the OM, water, and mineral nitrogen is added to calculation layer 1	OM index (*item_ix* in CDYOPSPA)	added amount of C [kg C ha^{-1}]	FM amount of added substrate [dt ha^{-1}]
4	mineral nitrogen application	NO_3-N and NH_4-N is added to calculation layer 1	fertilizer index (*item_ix* in CDYMINDG)	Ammonium N content [%]	total N input [kg N ha^{-1}]
5	soil tillage	state variables for water, temperature, nitrogen, and all organic compounds are averaged over the tillage depth	device index (*item_ix* in CDYTILLDEV)	tillage depth [dm]	tillage depth [cm]
6	crop cutting	height and cover grade of crop is reduced	-	-	-
7	irrigation	If amount>0 then water is added to calculation layer 1; Special values: -999 enables auto irrigation; 0 disables auto irrigation	device index (*item_ix* in CDYIRRGEV)	water amount [mm]	water amount [mm]
9	harvest (by-product **not** removed)	crop growths stop. Crop residues as well as by product are added to soil layer 1 as OM	crop index (*item_ix* in CDYCROPS)	real N uptake [kg N ha^{-1}]	real yield [dt ha^{-1}]
10	pasture start	animal count is increased	animal index (*item_ix* in CDYLIVES)	-	number of animals added to the management unit
11	pasture stop	animal count is decreased	animal index (*item_ix* in CDYLIVES)	-	number of animals removed from the management unit
12	sowing	crop module is initialized but not started	crop index (*item_ix* in CDYCROPS)	expected N uptake [kg ha^{-1}]	expected natural yield [dt ha^{-1}]

* standardised FM: i.e.: 86% DM for cereals

CDY_MVDAT

Data from experimental observations (measurements).

attribute	meaning	unit/type
key	unique record identifier	serial
fname	folder name	string (5)
snr	plot number	integer
utlg	subplot number	integerr
datum	sampling date	[dd.mm.yyyy]
m_ix	pointer to the property index in CDY_PROP	integer
s0	upper boundary of the sampled soil layer	[dm], integer
s1	lower boundary of the sampled soil layer	[dm], integer
m_wert	measurement value	float
variation	variance (not required)	float
anzahl	number of replications (if any)	integer
s_wert	simulation result	float
korrektur	'Y' model adapts to observation; 'N' model writes S_WERT	Y/N

Remarks

The soil profile is segmented to single calculation layers that are numbered increasing with depth and starting from 1 of the topmost. Since each layer has a standard thickness of 1 dm, all state values (temperature, soil moisture, C_{org} etc.) are allocated to the middle of a calculation layer (0.5, 1.5, 2.5 etc. dm)

One must consider two cases for the coding of observation values:

i) The observation relates to the middle of one calculation layer, for instance 1.5 dm. In that case, it corresponds to the state variable in calculation layer 2 and the input values are S0 = 1 and S1 = 2

ii) The observation is on the borderline between two calculation layers, for instance 3 dm. In that case it corresponds to the average of the state variables in layers 3 and 4 but the input values are S0=3 and S1=3

At runtime, these inputs are used to calculate the output x from the modelled values m in the following way:

If S0 = S1 then {h1: = S0-1; h2: = S1+1} else {h1: = S0; h2: = S1}

$$sx = \sum_{i=h1}^{i=h2} m[i]$$

i = number of the calculation layer

x = sx/n (depending on the type of the state variable)

For including measurement values within your simulation runs, you must be aware of accessing the correct database table. Using the GUI, measurement values are permanently stored in the table CDY_MVDAT. For each simulation run the selected measurement values are temporarily stored in the table CDY_MSDAT. By working with the model via batch file calls, measurement values are retrieved only from the table CDY_MSDAT, which must be created by the user. During batch mode simulations CDY_MSDAT remains permanent available (only simulated data are updated). The structure of the CDY_MSDAT corresponds to CDY_MVDAT but contains the extra index field (*ix*). This index is composed as a string from attributes : fname, plotnumber, year, julian day , m_ix, so, s1

Be aware of handling the data within these tables in a date ordered way, otherwise there may occur errors or blank simulation values during the simulation run.

CDY_CLDAT

Climate data in daily time steps.

attribute	meaning	unit/type
datum	date	date
ltem	air temperature at 2 m	[°C]
nied	precipitation	[mm]
glob	global radiation	[J cm^{-2}]
sonn	sunshine duration	[h]
wstat	acronym of climate station, link to CDY_FXDAT	string (3)
widx	weather index as *wstat*||'_' || date_part('year',*datum*)	string

Remarks

glob and *sonn*: the model requires global radiation data only. If these are not available, they will be calculated from sunshine duration and latitude (*geobreite* in CDY_FXDAT) during simulation runs. If radiation data are presented in power units, the data need to be transformed taking into account that 1 Ws = 1 J, 1 d = 8.64 10^4 s, and 1 m^2 = 10^4 cm^2; therefore 100 Ws m^{-2} = 864 J cm^{-2}.

PROFILE

Includes all soil profiles of your database.

attribute	meaning	unit/type
profil	name (abbreviation) of the profile	string
horizont	depth (lower limit of the horizon)	[dm], integer
horiz_name	pointer to a record in CDY_HRZN	string

CDY_HRZN

Detailed horizon description of the profiles.

attribute	meaning	unit/type
item_ix	numeric table index	integer
name	horizon name (refers to *horiz_name* in PROFILE)	string
hydromorph	TRUE : horizon is always saturated with groundwater	boolean
active	TRUE: means plough layer /Ah horizon	boolean
ct	ref.value of C_{org}, basis for dynamics of trd, pwp, fcap,pv	[M. %]
trd	bulk density	[g cm^{-3}]
pwp	permanent wilting point	[Vol. %]
fkap	field capacity, good to estimate soil moisture in spring	[Vol. %]
pv	pore volume	[Vol. %]
kf	saturated conductivity	[mm d^{-1}]
lambda	seepage parameter after Glugla (facultative)	number
hkap	heat capacity (standard: 0.16)	number
ton	clay content (facultative)	[M. %]
schluff	silt content (facultative)	[M. %]
fat	clay and fine silt (particles ≤ 6.3 µm)	[M. %]
relrootres	relative value describing rooting resistance	[0...1]
stone_cont	percentage of gravel and stones	[Vol. %]
mi	size of micropores ; default: 5	float
me	size of mesopores :ddefault :10	float
ma	size of macropores : default 500	float

CDY_PROP

attribute	meaning	unit/type
item_ix	unique record identifier	integer
propname	Full name of property	varchar(20)
shrtname	Short name of property	varchar(10)
pgroup	Group name of the property	varchar(20)
format	potential range for state variables single : in one layer only saeule : in topsoil (active soil layer) profil : in total soil profile	varchar(10)
typ	describes how data are aggregated su : average over soil layers (adapt) fx : flux, no impact av : average over soil layers (recording) – : no impact	char(2)
propunit	Unit of property	varchar(10)
adaptbl	TRUE: related statevariable can be adapted	boolean

CDY_PROP_SEL

attribute	meaning	unit/type
item_ix	unique record identifier	integer
sch_ix	Pointer to a list record in CDY_PROP_LST	integer
intervall	Indicates output frequency; 1: daily, 2:pentades; 3: decades; 4: monthly; 5: end of each year	integer
m_ix	pointer to the property index in CDY_PROP	integer
s0	upper boundary of the sampled soil layer	integer
s1	lower boundary of the sampled soil layer	integer
start	start of recording time intervall	date
enddat	end of recording time intervall	date

CDY_PROP_LST

attribute	meaning	unit/type
item_ix	unique record identifier	integer
sname	list name given by user	varchar(20)

CDY_RSLT

attribute	meaning	unit/type
item_ix	unique record identifier	
resultnr	unique identifier; produced by CANDY not changeable	
resultat	name of this item; produced by CANDY not changeable	
auswahl	If selected: '*' ;otherwise empty	
i_type	Characteristic of the item coded by two characters , produced by CANDY not changeable first character: F: flux, cumulative value over time Z.: actual value (german 'Zustand') M. mean value over time second character: P : sum over soil layers Q : average over soil layers	
unit	unit is produced by CANDY please don't change	

CDY_RSLT_SEL

attribute	meaning	unit/type
item_ix	unique record identifier	
list_ix	unique identifier for the result list	
resultnr	unique identifier; produced by CANDY not changeable	

CDY_RSLT_LST

attribute	meaning	unit/type
list_ix	unique record identifier	
listname	list name given by user	
res_out	Indicates output frequency; 1: daily, 2:pentades; 3: decades; 4: monthly; 5: end of each year – given by *y_end* in the general section of the ini file	

Appendix C: Working with big data sets

If data sets get bigger, they may reach a point where the work with the user interface is no longer smooth.

If the reason for this is the great number of items (soils, plots etc.) the performance of the interface can be increased by building data slices.

Here, a new schema named alldat has to be created and filled with the complete data set consisting of the tables:

- cdy_fxdat
- cdy_cldat
- cdy_madat
- cdy_mvdat
- profile
- cnd_hrzn

Furthermore, two tables have to be arranged to split the complete dataset into slices:

```
CREATE TABLE alldat.scenario (
        item_ix int8 NULL,
        "name" varchar(25) NULL,
        "comment" varchar(255) NULL
);

CREATE TABLE alldat.scenario_content (
        item_ix int8 NULL,
        fname varchar(25) NULL,
        snr int8 NULL,
        utlg int8 NULL
);
```

The scenario table defines the data slices with a unique item_ix. In the table scenario_content the individual data sets are defined for each data slice using fname, snr, and utlg as indicators.

The access to each of the slices is organised over the ini- file. The database section contains an alias list of databases where the data access for each alias is described in individual sections. To build a slice one of the parameters has to be scen_id=x where x relates to an item_ix in scenario_content. If such an alias is activated the selected part of the data in the alldat schema is moved into the public schema where CANDY can use them. There will be a warning about possible data loss is the public schema, before the data move is executed. Data changes over the interface are saved primarily in the schema public. If applicable, the user will be asked to update the data sets in the alldat schema as well. This is possible for cdy_fxdat, cdy_madat, cdy_mvdat, cdy_hrzn, and partly for profiles (only changes of depth parameter).

Appendix D: Import data from an old CANDY21 MSACCESS-Database

If you use this option, a window will open that shows the content of the ACCESS database in the left listbox. The right listbox shows all the tables of the postgres database requiring a data import. After a click on a required postgres table, the system finds the matching table on the ACCESS site. The fields and attributes of each table are shown below. Matching attributes are already checked and require no further action.

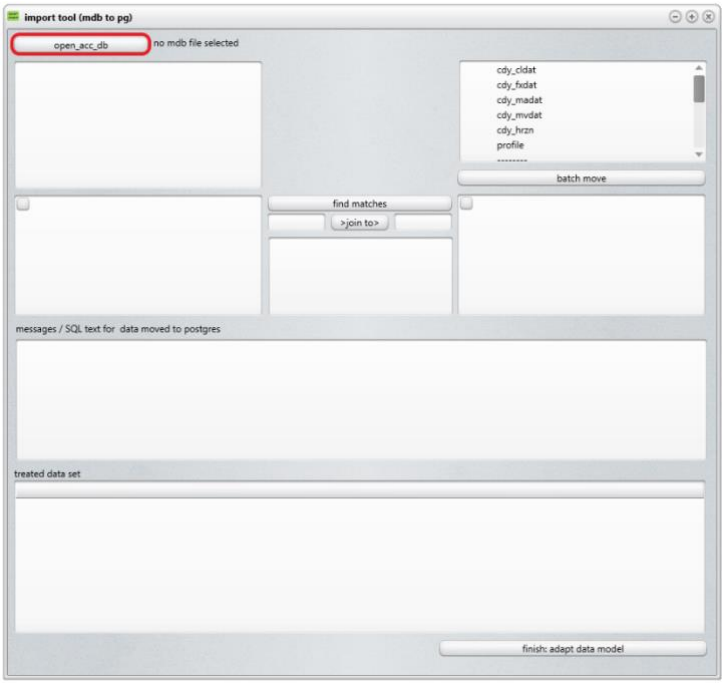


Figure 30: open the MS-ACCESS file

Click on *batch move* to complete the data transfer. After this step, the postgres table and the boxes with table attributes are cleared from the right list.

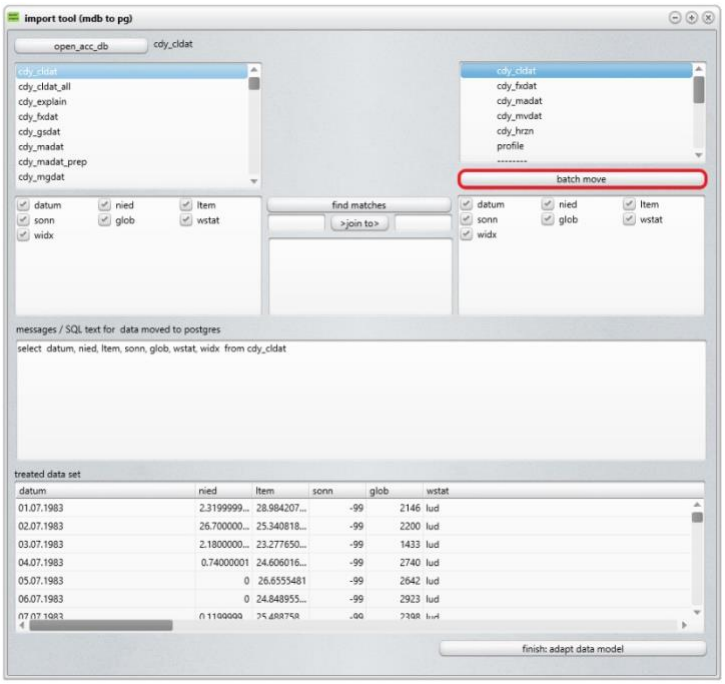


Figure 31: start data transfer for each table

In that manner, continue the data import for all tables on the postgres site. If there are no matching table names found (like probably for the table cdy_hrzn), a system error appears. In this case you have to select the matching ACCESS table manually by double clicking and continue as described above. *As the horizon data (cdy_hrzn) now contains the pore volume (pv) rather than particle density (tsd), it may be reasonable to adjust that manually in ACCESS beforehand according to pv=100*(1-trd/tsd; with trd as dry bulk density (german: Trockenrohdichte)).*

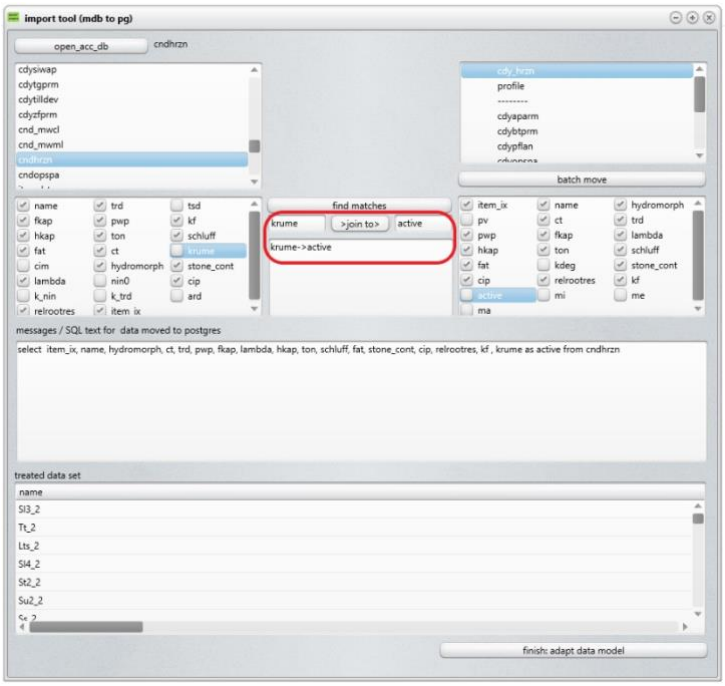


Figure 32: join names to fill up all required fields

In case that field names do not match, they have to be joined manually as well. A double-click on the field name on each site will bring the names to the join edit. After clicking the *join to* button, the join instruction is shown in the list box below and you can proceed with *batch move*.

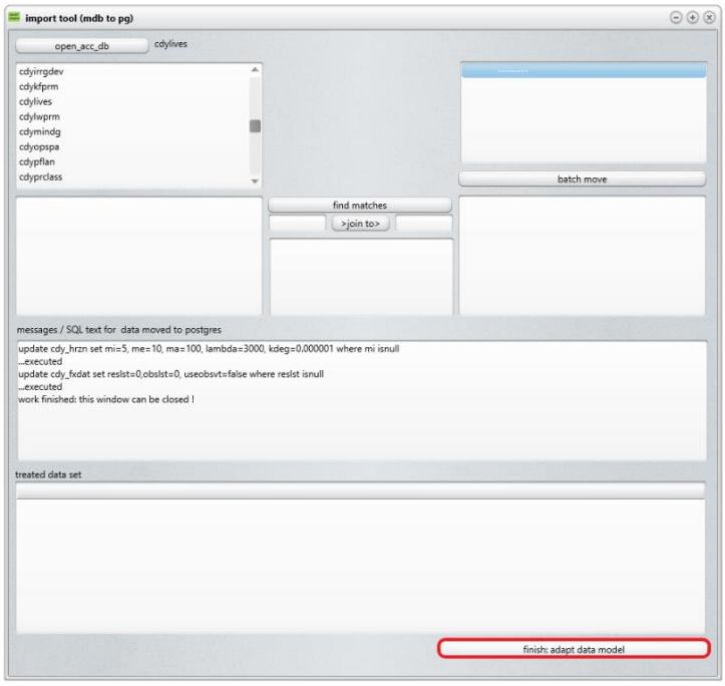


Figure 33: make the database available for CANDY

When no tables on the postgres side are left, proceed by clicking *adapt data model* to make final adjustments that are shown in the listbox above the data grid. Now, close all windows and start the CANDY application.

Because of changes in the data structure from the prior version, it may be possible that manual adjustment to the data is required. Please check the new content carefully!

Appendix E: Dynamic crop growth model

Starting with version 22.6 CANDY includes a dynamic crop growth model based on the paper by Huang et al. (2009) <doi:10.1016/j.agrformet.2008.07.013 >. If this submodel should be used there have to be provided some specific parameter tables:

- cdyacrop : provides the specific parameters for each crop
- cdyacrdp : crop specific sequences of development dependent parameter values
- cdy_clcng: time dependent climate parameters (CO2-conc) in annual time steps

The data in cdyacrop and cdyacrdp can be edited in the autocrop section of the parameter tab

Figure 34 : autocrop section of the parameter tab showing all essential parameters used be the autocrop model

The table cdy_clcng contains annual values of climate elements like the atmospheric CO2 concentration.

CDYACROP

attribute	**meaning**	**unit/type**
item_ix	unique record identifier; pointer to item_ix in cdycrop	integer
…		

CDYACRDP

attribute	**meaning**	**unit/type**
item_ix	Pointer to item_ix in cdyacrop	Int4
pname	Parameter name	varchar(6)
dvi	Specific crop development index (0..1..2)	float8
pval	Parameter value valid for the given dvi	float8

The crop development state is mainly controlled by the reference temperature sum (ref_ts). A negative value for this parameter enables an *autoharvest* when the development is completed with full maturity. This will mainly be used for scenario simulations. In this case it may be reasonable to switch to a flexible application rate of mineral nitrogen. When the amount of added N is specified as

negative number, the model will interpret this as a demand for soil nitrogen storage that should be filled up.

Starting with version 22.7 CANDY includes a dynamic grassland model where growth is driven by transpiration. Initialize grassland with action ‘sowing’ and select a crop of type ‘CDYDGRN’. Specific to this crop model is the action ‘cutting grassland’. Specific parameters are *transko, tkmin, ts1, ts2* and *nbok* to allow for symbiotic N-fixation. This model can also react in a flexible way to the actual climate data.

More details can be found in the model description.

Appendix F: Automatization of candy runs

Special cases may require a larger number of simulation-runs so that a manual start is inconvenient. Moreover, it may be necessary to combine the management data of a given folder with several soil profiles and climate data. Here it is possible to fill up the table cdy_*automat* with all required information and call the candy app with the parameters DB=<your database> AUTO.

A large part of cdy_automat is the same as cdy_fxdat plus some additional fields:

CDY_AUTOMAT

attribute	meaning	unit
fname	folder name	
snr	plot number 1...9999	
utlg	subplot number 0...9 (optional, only if meaningful)	
geobreite	geographic latitude; only used to transform sunshine duration into global radiation	
standort	pointer to profile in PROFILE	
wetter	pointer to wstat in CDY_CLDAT	
startdat	intended start of simulation	
stopdat	intended stop of simulation or end of basic scenario for periodic simulation runs	
immission	annual atmospheric N deposition	kg/ha/yr
ltem	long term average of air temperature	°C
nied	long term average of annual rainfall	mm
crep	Carbon input level ; default: 9	dt/ha
nlevel	Initial N in total soil profile	kg/ha
nfk0	Initial filling of plant available field capacity (rough guess); default: 100	%
reslst	Number of the list with recordable results	
obslst	Number of the list with virtual observations	
useobsvt	TRUE: values from cdy_mvdat will be included; FALSE: no recording and NO ADAPTATIONS will be applied	
pset	link to item_ix in cdyaparm to select a specific set of general parameter	
rtab	Name of the result-table i.e. rs_auto to record results given by rslst, if rtab ends on .txt like rs_auto.txt the output is directed to a text file of this name. If the file already exists, the user can decide whether to delete the file or append records.	
simtag	Marks the state: 0: simulation run required -1: simulation running (active simulation object) 1: simulation run finished -99: indicates a former model crush (unfinished run)	
obj_id	Unique identifier, recorded as objekt_id with results in the result table	
nrep *	Optional: number of management repetitions between the first '1.1.' and the last '31.12.'	

*the *nrep* column is optional. If it exists, CANDY will perform periodic simulation runs as shown in . If it is missing, CANDY will run a normal simulation.

The table *cdy_automat* can be used in pre- or postprocessing sql scripts addressing the current obj_id using the condition *where simtag=-1*

Example application to study sensitivity

Problem: impact of initial SOC concentration and level of N-fertilization on change of SOC stock and the cumulative N2O loss for a new established grassland (DB name: aver; fname: aver; snr=3 utlg=0)

Initial SOC concentration: gaussian distribution of 2 +- 0.05

N fertilisation efficiency : uniform distribution 0.9..1.1

150 samplings

```
rm(list=ls())
library(RPostgreSQL)
library(ggplot2)
library(lhs)
set.seed(1134)
x<-data.frame(randomLHS(n=150,k=2))
print(summary(x))
plot(x$X1~x$X2)
x$v1<-qnorm(x$X1,mean=2, sd=0.05)
x$v2<-0.9+x$X2/5
plot(x$v1~x$v2)
print(cor(x$v1,x$v2))
# DB Verbindungen herstellen
drv<-dbDriver("PostgreSQL")
# Datenverbindung cn1
cn1<-dbConnect(drv,host='127.0.0.1',port='5432',user='candy_user', password='candy_go',dbname='candy_gcef')
postgresqlpqExec(cn1, "SET client_encoding = 'utf-8'")
sql <- paste0( " select sbez from cdy_fxdat where fname='aver'")
##postgresqlpqExec(cn1, isq)
mytab<-x
cds<-data.frame(dbGetQuery(cn1,sql))
dbWriteTable(cn1,"mytab",mytab)
dbDisconnect(cn1)
```

---

to modify the format and slightly change the content:

```
drop table mysens;
create table mysens as
select "row.names"::integer as obj_id , v1 as soc0 , v2 as neff, -99.999 as cstock0 from mytab;
```

---

for the succeeding example we create cdy_automat as follows:

```
delete from cdy_automat ;
insert into cdy_automat (fname, snr, utlg, geobreite, standort, wetter, startdat, stopdat, immission, ltem, nied, crep, nlevel, nfk0, reslst, obslst, useobsvt, pset, rtab, simtag, obj_id)
select fname,snr,utlg,geobreite,standort,wetter,startdat,stopdat,
immission,ltem,nied,crep,nlevel,nfk0,reslst,obslst, useobsvt, 1 as pset, 'rs_sens' as rtab, 0 as simtag,
"row.names"::integer as obj_id from cdy_fxdat cf , mytab mt where cf.fname='aver' and cf.snr=3;
```

---

snapshot of example table cdy_automat, record 1 an2 are finished record 3 is in process

fname	snr	utlg	geobreite	standort	wetter	startdat	stopdat	immission	ltem	nied	crep	nlevel	nfk0	reslst	obslst	useobsvt	pset	rtab	simtag	obj_id
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	1	1
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	1	2
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	-1	3
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	4
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	5
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	6
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	7
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	8
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	9
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	10
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	11
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	12
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	13
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	14
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	15

…(line 16 -132 not shown)..

aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	133
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	134
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	135
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	136
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	137
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	138
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	139
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	140
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	141
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	142
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	143
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	144
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	145
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	146
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	147
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	148
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	149
aver	3	0	50.0	gcefavg	G-A	2013-09-01	2017-12-30	50.0	9.0	500.0	9.0	2.0	90.0	2	2	true	1	rs_sens	0	150

Example table mysens generated with an R script followed by an sql command (see above)

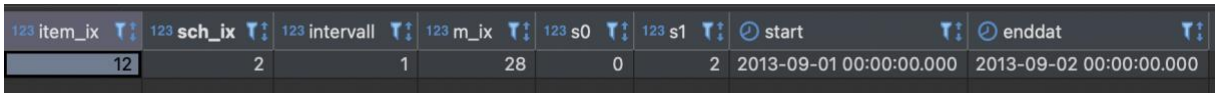
obj_id	soc0	neff	cstock0
1	1.93944944773562	0.969110385284138	-99.999
2	1.98037523928709	1.06995216675134	-99.999
3	2.09176592231151	1.00391666582475	-99.999
4	1.99695814753557	1.02954079569659	-99.999
5	2.01149277016441	0.903018214816848	-99.999
6	2.05500927808471	1.07634019957483	-99.999
7	1.98376103495725	0.929382327730457	-99.999
8	1.89363765061143	1.03408651403772	-99.999
9	2.00577925619267	0.90896480753428	-99.999
10	1.94050176651887	1.02728455548547	-99.999
11	2.04681508561061	0.936814320609905	-99.999
12	2.04145497761642	1.08277262148447	-99.999
13	2.00067964567882	1.07839919215379	-99.999
14	2.03495319140345	0.906609390702099	-99.999

…. Line 15 -141 not shown

142	1.9498865714622	1.05264341090278	-99.999
143	2.13671083591117	1.02561948935656	-99.999
144	2.02753879967154	1.00989456565709	-99.999
145	2.0151162332917	0.923031442792155	-99.999
146	1.99128680548623	1.03842190529406	-99.999
147	2.04213539226714	0.962149692840874	-99.999
148	1.96331254763713	0.965060465119469	-99.999
149	2.02909544436667	0.977196826979208	-99.999
150	1.92854897418274	1.09874382915472	-99.999

We have selected both, obslst and reslst number 2. The contents are:

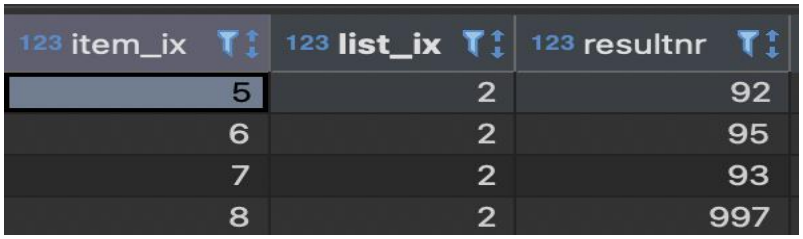
Observations:



item_ix	sch_ix	intervall	m_ix	s0	s1	start	enddat
12	2	1	28	0	2	2013-09-01 00:00:00.000	2013-09-02 00:00:00.000

Here is only selected SOC_stock (m_ix=28) in order to get the carbon stock for the initial SOC concentration.

Results (92-Cdec, 95- Clts, 93- Cops: 997- N2O loss)



item_ix	list_ix	resultnr
5	2	92
6	2	95
7	2	93
8	2	997

The C-pools are later used to get the C-stock and the difference to the initial stock but one can discuss if it is reasonable to include Cops in the calculation of the C-stock too.

We have to prepare two additional file for pre- and postprocessing. In the pre-processing step the parameter dng_eff in table cdyaparm has to be modified according to the record of table mysens that is linked to cdy_automat via the obj_id. Furthermore, the initial SOC concentration has to be changed in a similar way

preprocessing:

-- step 1: change the factor that modifies the amount of applied mineral N fertilizer

Both steps in file sensit_preproc.sql

```
-- preproc (dng_eff)
update cdyaparm set dng_eff = a.neff
from (  select neff from mysens s join cdy_automat c on s.obj_id =c.obj_id where c.simtag=-1 ) a
where cdyaparm.item_ix=1;
```

--step 2: change the initial value for SOC in table cdy_mvtab

```
-- preproc ( soc0 )
update cdy_mvdat set m_wert = a.soc0
from (  select soc0 from mysens s join cdy_automat c on s.obj_id =c.obj_id where c.simtag=-1 ) a
where cdy_mvdat.key = (select key from cdy_mvdat mv , cdy_automat ca
where mv.fname=ca.fname and mv.snr=ca.snr and mv.m_ix=7
and mv.korrektur<>'N' and ca.simtag=-1)
```

After each of the 150 simulation runs we want to know what SOC stock is represented by the initial concentration. We read this value from table cdy_msdat and update the according record in table mysens. (of course, there would be other ways to do this, but this is a simple example to demonstrate the usage of postprocessing)

postprocessing

```
-- postproc (file sensit_postproc.sql)
update mysens s  set cstock0=a.s_wert from
( select s_wert from cdy_msdat where m_ix=28 and snr=3 and fname='aver'
and datum = to_date('01.09.2013','dd.mm.yyyy') ) a
```

where s.obj_id in (select obj_id from cdy_automat where simtag=-1)

batchcall to test the pre/post sql

on MAC:

open -a cdy22_GUI.app --args DB=aver A=01.09.2013 E=30.12.2017 W=G-A P=gcefavg FN=aver SNR=3 SUB=0 RL=rs_aver PSET=1 OL=2 RL=2 GO PPQ=sensit_preproc.sql POQ=sensit_postproc.sql

remark:

to make this as script executable you add as first line

#!/bin/sh

.. and then run in terminal:

chmod u+x ./../candy_script.sh

on WIN:

cdy22_GUI.exe DB=aer A=01.09.2013 E=30.12.2017 W=G-A P=gcefavg FN=aver SNR=3 SUB=0 RL=rs_aver PSET=1 OL=2 RL=2 GO PPQ=sensit_preproc.sql POQ=sensit_postproc.sql

It may be useful to include some sql commands within the script or batch file. Here it is important that on WINDOWS the password for the specified postgres user has to be submitted beforehand to avoid the script stopping in between (if several sql statements are required). It may also be useful to put the details for postgres connection to variables at the beginning of the script.

batchcall for sensitivity analysis (all 150 records)

on MAC:

open -W cdy22_GUI.app --args DB=candy_gcef AUTO PPQ=sensit_preproc.sql POQ=sensit_postproc.sql

or in a script including a reset of the auto table:

```
#!/bin/sh
adb=candy_gcef
host=127.0.0.1
port=5432
user=candy_user
psql -h $host -p $port -U $user -d $adb -c “update cdy_automat set simtag=0”
open -W cdy22_GUI.app --args DB=$adb AUTO PPQ=sensit_preproc.sql POQ=sensit_postproc.sql
echo “ finish of candy run”
```

on WIN:

cdy22_GUI.exe DB=candy_gcef AUTO PPQ=sensit_preproc.sql POQ=sensit_postproc.sql

the batch call on windows is similar to the shell script shown above:

```
echo on
set adb=candy_gcef
set host=127.0.0.1
set port=5432
set user=candy_user
set PGPASSWORD=candy_go
"c:\Program Files\PostgreSQL..\bin\psql"  -h %host% -p %port% -U %user%-d %adb% -c "update cdy_automat set simtag=0 “
cdy22_GUI.exe DB=%adb% AUTO PPQ=sensit_preproc.sql POQ=sensit_postproc.sql
echo “ finish of candy run”
```

Result processing

after finishing the 150 runs we create a table containing the results at the end of each simulation year (excluding merkmal_id. 997 leaves all C-pools in the calculation):

```
drop table if exists time_sens_n2o;
create table time_sens_n2o as
select obj_id,yr,n2o, sum(n2o) over ( partition by obj_id order by yr) as sn2o from
(select   objekt_id as obj_id, date_part('year', datum) as yr, sum(wert) as n2o from rs_sens
where merkmal_id=997 and left(to_char(datum,'dd.mm.yyyy'),5)='31.12' group by objekt_id,datum
) q1;

drop table if exists time_sens_dsoc ;
create table time_sens_dsoc as
select m.obj_id,q1.com1,q2.soc1,m.soc0,m.neff,m.cstock0, q1.yr, q1.com1-m.cstock0 as dcom, q2.soc1-
m.cstock0 as dsoc from
(select  datum, objekt_id as obj_id, date_part('year', datum) as yr, sum(wert) as com1 from rs_sens
where merkmal_id<>997 and left(to_char(datum,'dd.mm.yyyy'),5)='31.12'
group by datum,objekt_id) q1,
(select   objekt_id as obj_id, date_part('year', datum) as yr, sum(wert) as soc1 from rs_sens
where merkmal_id in (92,95) and left(to_char(datum,'dd.mm.yyyy'),5)='31.12'
group by datum,objekt_id ) q2,
mysens m where m.obj_id=q1.obj_id and q1.obj_id=q2.obj_id and q1.yr=q2.yr
order by m.obj_id,q1.yr;

drop table if exists time_sens;
create table time_sens as select c.obj_id,c.yr,c.com1,c.soc1,c.soc0,c.neff,c.dcom,c.dsoc,n.n2o,n.sn2o
from
time_sens_dsoc c, time_sens_n2o n where c.obj_id=n.obj_id and c.yr=n.yr
order by obj_id,yr;
```

Then we use following R script to calculate and present sensitivities:

```
#clean up
rm(list=ls())
# load libraries
library(RPostgreSQL)
library(ggplot2)
# DB connection
drv<-dbDriver("PostgreSQL")
cn1<-dbConnect(drv,host='127.0.0.1',port='5432',user='candy_user', password='candy_go',dbname='aver')
postgresqlpqExec(cn1, "SET client_encoding = 'utf-8'")
sql <- paste0( " select distinct yr from time_sens order by yr")
tl<-data.frame(dbGetQuery(cn1,sql))
tl$s_soc0<--99
tl$s_neff<--99
# delta soc
for (i in 1: length(tl$yr)) {
 ayr<-as.character(tl$yr[i])
 sql <- paste0( " select * from time_sens where yr=",ayr)
 cds<-data.frame(dbGetQuery(cn1,sql))
 print(ayr)
 lr<-lm(data=cds,dsoc~soc0+neff)
 print(summary(lr))
 sx1<-sd(cds$soc0)
 sx2<-sd(cds$neff)
 sy<- sd(cds$dsoc)
 # change of C-stock
 sens_csoc0<-(lr$coefficients[2]*sx1/sy)^2
 sens_cneff<-(lr$coefficients[3]*sx2/sy)^2
 tl$s_soc0[i]<-sens_csoc0
 tl$s_neff[i]<-sens_cneff
}
g1<-ggplot(tl)+geom_line(aes(x=yr,y=s_soc0),color='green')+ylab('sens')+
 geom_line(aes(x=yr,y=s_neff),color='red')
print(g1)

# sum n2o
tl$sn_soc0<--99
tl$sn_neff<--99
for (i in 1: length(tl$yr)) {
 ayr<-as.character(tl$yr[i])
 sql <- paste0( " select * from time_sens where yr=",ayr)
 cds<-data.frame(dbGetQuery(cn1,sql))
 print(ayr)
 lr<-lm(data=cds,sn2o~soc0+neff)
 print(summary(lr))
 sx1<-sd(cds$soc0)
 sx2<-sd(cds$neff)
 sy<- sd(cds$sn2o)
 # change of C-stock
 sens_nsoc0<-(lr$coefficients[2]*sx1/sy)^2
 sens_nneff<-(lr$coefficients[3]*sx2/sy)^2
 tl$sn_soc0[i]<-sens_nsoc0
 tl$sn_neff[i]<-sens_nneff
 }
g2<-ggplot(tl)+geom_line(aes(x=yr,y=sn_soc0),color='green')+ylab('sens')+
 geom_line(aes(x=yr,y=sn_neff),color='red')
print(g2)
```

Here are the results with a squared sensitivity index sens. Over time intitial SOC has lower importance for SOC change and N2O emissions while impact of N-fertilisation is increasing , especially for N2O emissions.

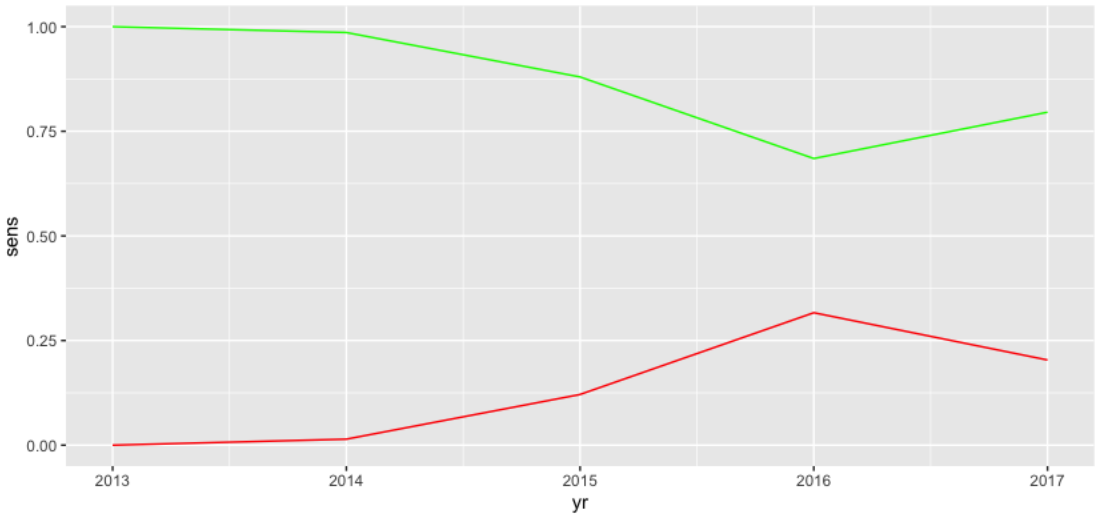


Figure 1: sensitivity of SOC change on initial SOC (green) and N-fertilisation (red)

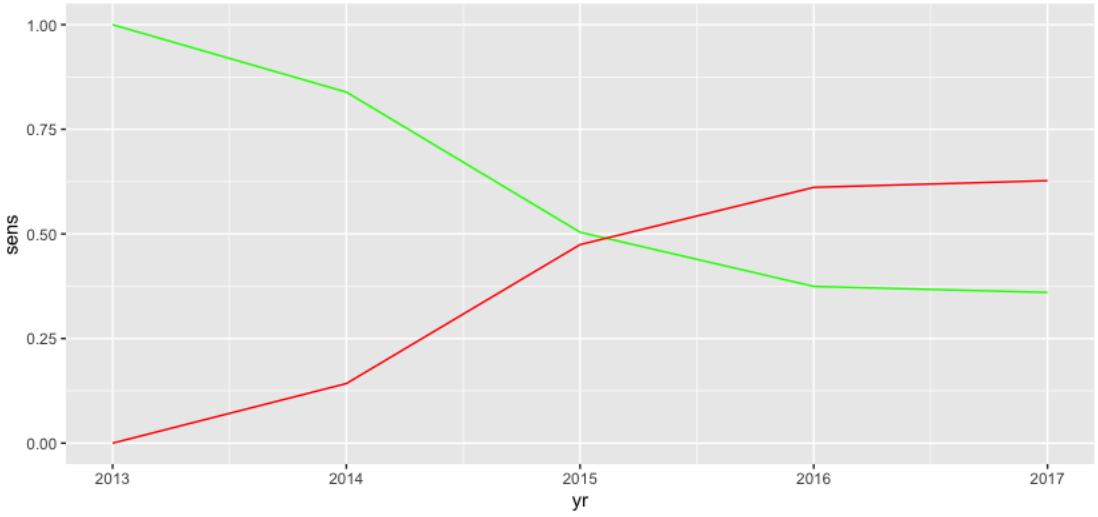


Figure 2: sensitivity of cumulative N2O emissions on initial SOC (green) and N-fertilisation (red)