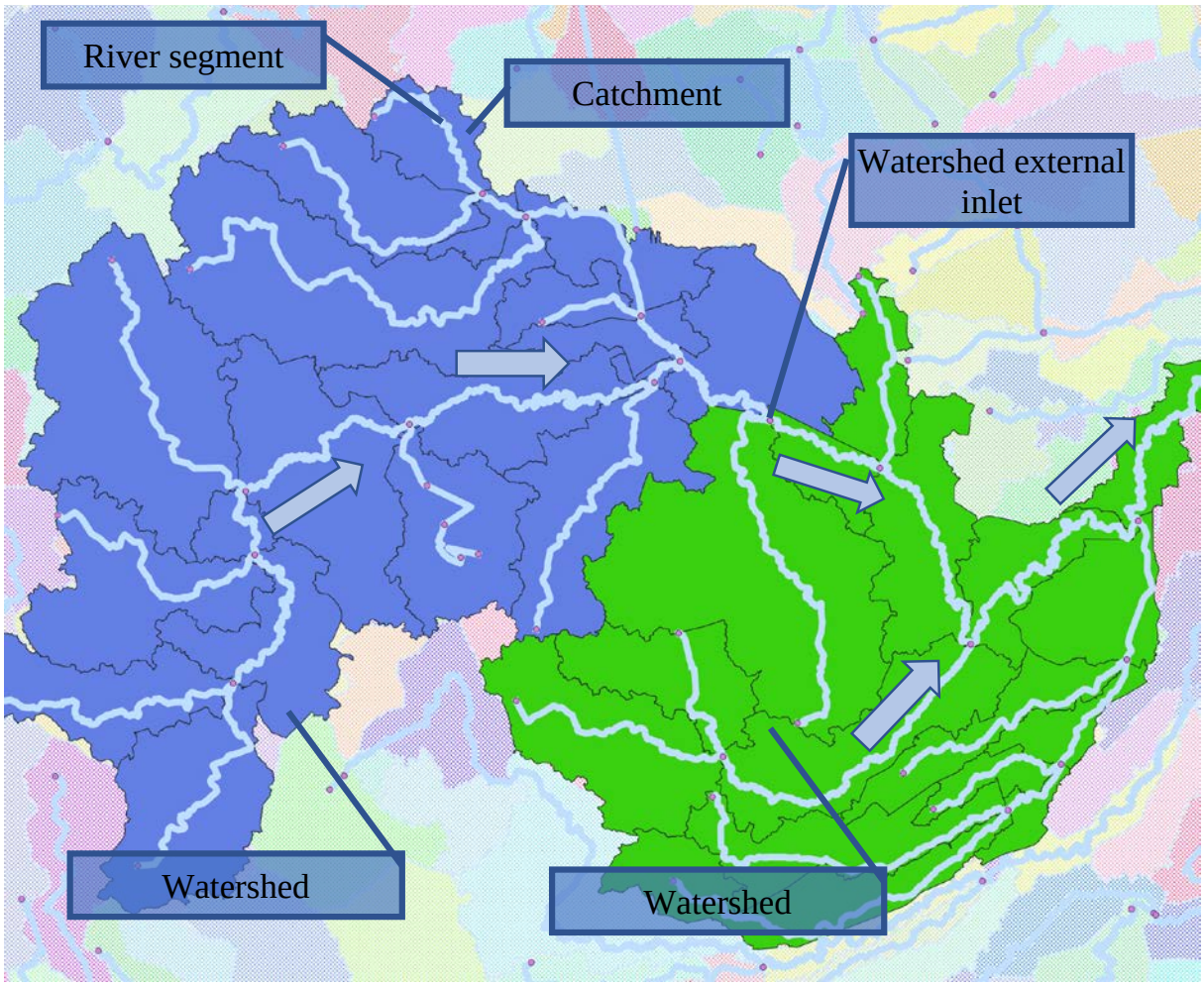
Eiropas Savienības LIFE programmas Integrētais projekts
"Latvijas upju baseinu apsaimniekošanas plānu ieviešana laba virszemes ūdens stāvokļa sasniegšanai"

CALIBRATED AND VALIDATED MODELLING SYSTEM

DELIVERABLE R2



Prepared within the LIFE GoodWater IP Action C1: "Development of the water quality and quantity system for the territory of Latvia"

Rīga, 2022

CALIBRATED AND VALIDATED MODELLING SYSTEM

Uldis Bethers, Juris Seņņikovs, Pēteris Bethers, Andrejs Timuhins, SIA "Procesu analīzes un izpētes centrs"

Recommended citation: PAIC, 2022. CALIBRATED AND VALIDATED MODELLING SYSTEM. LIFE GoodWater IP, Rīga, 47p.

Document prepared in the frame of the integrated project "Implementation of River Basin Management Plans of Latvia towards good surface water status" (LIFE GOODWATER IP, LIFE18 IPE/LV/000014), project has received funding from the LIFE Programme of the European Union and the State Regional Development Agency.

The information reflects only the LIFE GoodWater IP project beneficiaries' view and the European Climate, Infrastructure and Environment Executive Agency is not responsible for any use that may be made of the information contained therein.

© PAIC, 2022

Document versioning sheet	
Document version number	**v 1.0**
Document planned elaboration date	**06.2021**
Document elaboration date	**06.2022**
Document actual version elaboration date	**06.2022**
Project activity/sub-activity number	**C1**

Summary

This document is a description of the electronic deliverable: calibrated and validated modelling system for the water quantity and quality in the territory of Latvia. This document describes the organization of the elements of the modelling system: the background technologies, the structure of the modelling system data and tools, the scripts of the modelling system as well as the organisation of the electronic deliverables.

Document is written in English, it contains 47 pages, 4 figures, 1 table and 2 references.

Table of contents

Introduction

This document is a description of the electronic deliverable: calibrated and validated modelling system for the water quantity and quality in the territory of Latvia. The modelling system is developed on the basis of the data base prepared within Deliverable R1, PAIC (2020).

The documentation of the activities which resulted in the calibrated and validated modelling system will be presented in the Deliverable R4 – "Documentation of development, calibration, validation and results of SWAT+ modelling system".

This document describes the organization of the elements of the modelling system.

The background technologies are described in Chapter 1.

The structure of the modelling system data and tools are described in Chapter 2.

The scripts of the modelling system are described in Chapter 3.

The organisation of the electronic deliverables is described in the Chapter 4.

1.Background technologies

The background technologies are based on SWAT+ modelling system, new generation Python 3 (and a large amount of free libraries) programming language, and the open access GIS systems, databases and libraries.

The following technologies are chosen for the the modelling system:

- Database: PostgreSQL version 13.1-1[1] or later with spatial extension PostGIS version 3.0.3 or later for Windows-x64[2] is required for storing all modelling system.
- Python 3.8.6: WinPython64-3.8.6.0cod.exe[3] portable distribution of the Python programming language for Windows-x64 is used for all the scripts forming the modelling system.
- Geospatial information system QGIS with GDAL libraries[4] for operations with geospatial data is used.
- SWAT+ version rev. 60.5.4[5] released on 13-Apr-2022.
- SWAT+ Editor[6] v 2.0.1 source code.

The required background tools and systems mentioned in this Chapter are installed by running batch file.

[1] https://www.postgresql.org/download/
[2] https://postgis.net/install/
[3] https://winpython.github.io/
[4] osgeo4w-setup-x86_64.exe from https://trac.osgeo.org/osgeo4w/
[5] https://swatplus.gitbook.io/docs/installation
[6] https://bitbucket.org/swatplus/swatplus.editor/src/master/

2.Structure of data and scripts

2.1. Scripts of the modelling system

The structure of the modelling system data and the blocks of the system software (script) are shown in Figure 1.

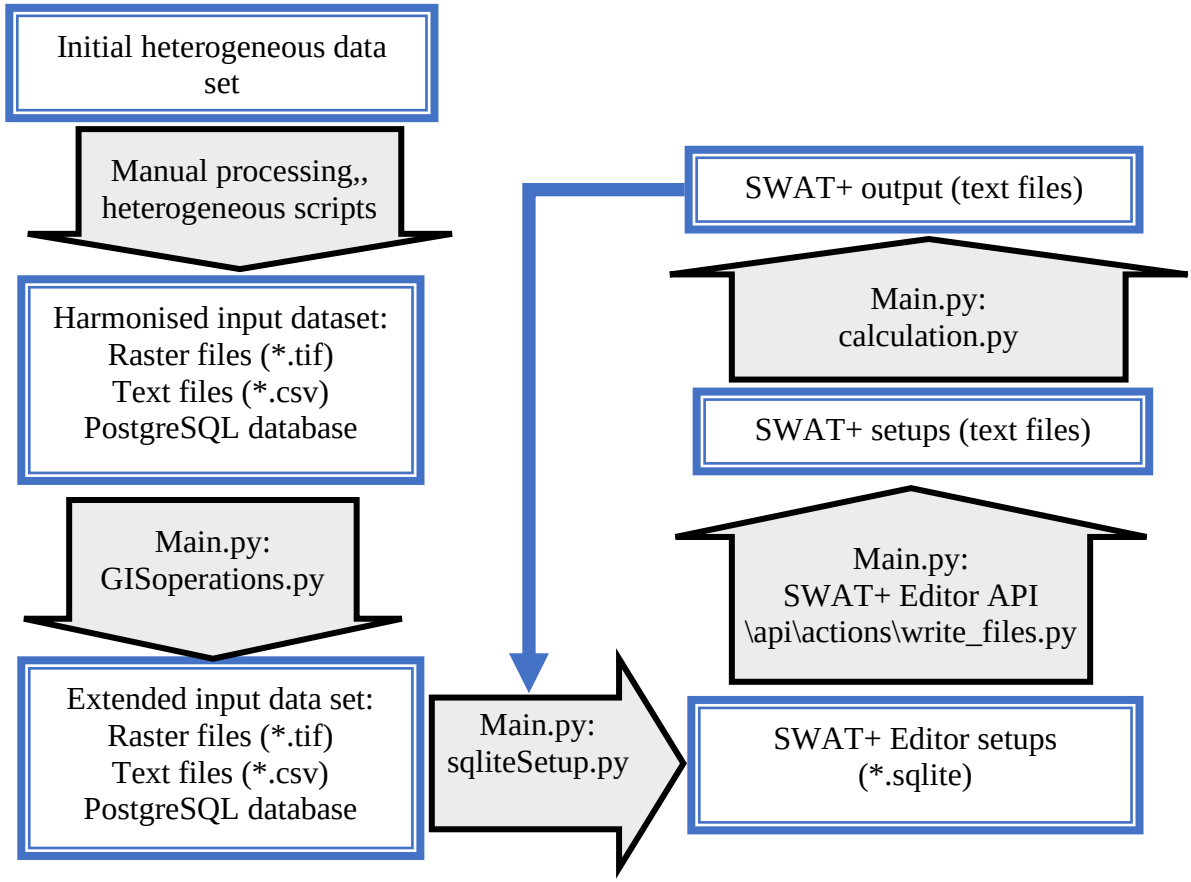


Figure 1: Elements of the modelling system.

The modelling system principles are the folowing:

1. The initial input data are manually (or with a help of proprietary scripts outside the modelling system) processed forming the harmonised input dataset (text, raster files and PostregreSQL database). The further data processing is performed by Python script *main.py*.

2. The geospatial operations with the harmonised dataset are performed with module *GISoperations.py*, forming the extended input data set.

3. Module *sqliteSetup.py* creates SWAT+ Editor setups from the input data.

4. Module *write_files.py* uses SWAT+ Editor API to creat SWAT+ setups.

5. Module *calculations.py* performs the runs of SWAT+.

Water quality model preparation process is designed to be fully automated. The functionality of the main scripts is described in Chapter 3 dealing with the model preparation process. In this Section we provide the overall scheme of developed scripts as well as their list.

The hierarchy of the Python scripts used in the system is given below, while the structure of folders of the system is described in Section 2.2. Two scripts are on the top of the hierarchy:

- **settings.py** – contains the description of the information used by the SWAT water quality modeling system as well as links to all the input data and intermediate result locations needed for other scripts of the system. The detailed description of the contents of **Settings.py** is given in Section 2.3.

- **main.py** – contains the sequence and parameter references needed to run the other scripts during the automated preparation of modeling system. **main.py** is divided into two parts – „geospatial" and „database". It invokes all other scripts to run them sequentially as well as provides the necessary iterations through the Watersheds (setups) of the modeling system for database scripts. Thus, this script is running the sequence of model preparation described in Chapter 3.

The rest of the scripts may be divided as (a) the ones processing the geospatial information (their functionality is described in Section 3.1, and they are invoked via **GISoperations.py**), (b) the others processing the SWAT file structure and SWAT databases (their functionality is described in Section 3.2) and (c) the service scripts. The list of the scripts used in the system is provided below in an alphabetical order:

atmosphericdeposition.py – script which creates a table of atmospheric deposition in catchments and updates the atmospheric deposition parameters in the database.

bufferstrips.py - script which creates a table of buffer strips in catchments and updates the buffer strip parameters in the database.

calculateSlopeRaster.py – script which prepares a slope raster from the DEM raster.

catchments.py – defines classes needed for holding all the information needed for creating the geometry of the modeling system: catchments, watersheds and rivers.

coefficients.py – script which defines how to change the SWAT parameters in the database.

common.py – utility which defines a set of common functions needed for conversion of data types.

config.py – configuration file referring to the executables and databases needed for other scripts.

drainage.py - script which creates a table of drainage fraction in HRUs and updates the melioration (drainage) parameters in the SWAT+ database.

GDAL_helper.py – utility module containing basic raster operations.

GISoperations.py – script through which all other geospatial scripts are called, it lists all possible GIS operations.

gw.py – script which applies initial concentrations of nutrients in the ground water.

HRU.py – utility script for data importing.

HRUrasterOps.py – script containing multiple geospatial functions needed to create HRUs.

initializeSetups.py – subroutine for preparation of the directory hierarchy of the watersheds.

initPAICSWAT.py – script which prepares all PAICSWAT related files.

lakesReservoirs.py – script which contains all operations done with lake and reservoir information.

lup.py – script which creates and updates the tables of land use update data.

managenment.py – script which calculates and updates all information associated with agricultural management practice (fertilizers, plowing, etc.).

observations.py – script used for reading observation data from respective files.

PG_helper.py – utility module containing basic operations with PostgreSQL database.

pointsources.py – script which contains all operations to read observed point source data and update it in the model.

precipCorrection.py – script containing functions needed to implement precipitation correction in the model.

prepareRuns.py – script used for creating and management of the hierarchical SWAT+ model runs in the modeling system.

regionalization.py – script containing routines to account for regionalization (hydrological regions).

riverdata.py – script containing functions needed for the calculation of river depth, width and Manning coefficient.

RunExe.py – utility script used for running executables in Python enviroment.

sqliteSetup.py – utility module used for creation of the SWAT+ Editor setups from the input data.

transboundary.py – script used for updating of transboundary inlets; depending on the variable of *transboundaryInflow* in **settings.py** it takes the inflow from either measured data (if *False*) or from transboundary SWAT+ model (if *True*).

txtReader.py – utility script for reading .txt files.

wetlands.py – script for updating wetland parameters in the PND table.

2.2. Structure of data in modelling system

There are 2 main parts of the modelling system: PostgreSQL database and data represented in the file system. The database table names are given relative to database **defaultPGDB** of *settings.py.*

The folder structure of the water quality modeling system is given relative to the root folder (**MAINPATH** of *settings.py*). The folder names are defined in *settings.py* file and they are summarised in Table 1.

Table 1. Values of variables in *settings.py* defining the water quality modeling systems.

Path	**Name of variable**
c:\WQMS	MAINPATH
\lib	pyPAICSWAT
\py	LIBRARYPATH
\bin	binPath
\swatplus.editor	swatplus_editor_source
\Projects	
\Common	commonDataDir
\Observations	
\Pointsources	
\Rasters	
\Setup	projectDir
\Data	dataDir
\Observations	
\Pointsources	
\Rasters	
PAICSWAT	
\Scripts	
settings.py	
main.py	
\Watersheds	setupsDir
….	

- Folder **pyPAICSWAT** (default value *lib*) contains the utility software modules:
  - The python modules are placed in folder **LIBRARYPATH** (default value *lib/py).*
  - The executable modules are placed in folder **binPath** (default value *lib/bin).*
  - SWAT+ Ediror source code is placed in folder **swatplus_editor_source** (default value *lib/swatplus.editor*)

- Folders commonDataDir contain the common data for the projets (modellind system currently contain only one projet - projectDir):
  - Observations
  - Pointsources
  - Rasters

- Folders **projectDir** contain the SWAT+ setups:
  - Subfolder *Scripts* contains the main script module (*main.py*) and the settings module (*settings.py*) configured for the respective SWAT+ setup.
  - Subfolder *Data* contains SWAT+ setup specific data
    - Observations
    - Pointsources
    - Rasters
  - Subfolder **setupsDir** (default value *Watersheds)* contains the setups of SWAT+ models.

The above folder structure of SWAT setups should correspond to the values of the variables defining the directory structure in the corresponding files *settings.py*:

1. **projectDir** must be set to the root folder of the respective modelling system.

2. **setupsDir** must be set to folder where setups of SWAT+ models will be located. By default it is a folder *Watersheds* relative to **projectDir**.

3. **dataDir** must be set to folder where the input data is located.

The water quality modeling system is generated automatically by executing the script ***main.py*** from folders **projectDir** */Scripts*.

2.3. Structure of information in Settings.py

The file *Settings.py* contains information and links to all of the input/output data and intermediate result locations needed for other scripts. Each water quality modeling system has its respective *Settings.py* file. The input/output data and other parameters are asigned to the variables of the *Settings.py*:

1. The variables defining the file names of input data.
2. The variables defining the file names of output data.
3. The default values of numerical (or text) variables.
4. The default values of parameters defining the field names of the input files (databases).
5. The values of variables defining the executables and configuration files.
6. The variables defining the structure of the modeling system.

The contents of this file, i.e. the description of the water quality modeling system is given below:

1. Description of the digital elevation model data.
   a. Variable *DEMFile* is set to the location of the digital elevation raster. This raster should contain one raster band with Z values of topography. Default value of the variable is **commonDataDir**+"Rasters*/dem_5m"*.
   b. Variable *SlopeFile* is set to the location of the topography slope raster. This raster could be calculated by the function *HRUrasterOps. calculateSlopeRaster*. The default value of the variable is set to **commonDataDir**+"Rasters*/slope_5m.tif"*.
   c. Variable *SlopeClasses* is a list of slopes (in percent rise!) used for defining slope classes. List should contain a list containing ranges of slope classes and class number as a list of python types [(startvalue1, endvalue1, classnumber1),...]. By default there are 2 slope classes 0.0 – 4.0 % is slope class 1, and >4.0% is slope class 2.

   (SlopeClasses=[(4.0,99999,2),(0.0,4.0,1)])
   d. Variable *slopeClassRaster* represents a slope class raster file that will be prepared for classification of HRUs. Default value of this variable is **commonDataDir**+"Rasters*/slopeclasses.tif"*.

2. Description of the landuse geospatial data (class *LandUse*).
   a. Field *TableShapes* contains the location of the **defaultPGDB** table representing landuse. This file should contain at least an attribute *LUCODE* representing the landuse name. Default value of this variable is ***(landuse, landuse)***

b. Variable *LUvsSWATTable* contains a location of the **defaultPGDB** table of remapping of the original landuse classes to SWAT+ landuse classes. This table should contain at least two attributes (1) *globalcode* – the landuse name corresponding to the landuses in landuse feature class; (2) *SWATCODE* – the SWAT+ landuse name corresponding to the names in the tables *crop* and *urban* (maximum 4 characters long). Default value of this variable is ***(landuse, globallookup)***.

c. Variable *Raster* represents a landuse raster file that will be prepared for the classification of HRUs. Default value of this variable is **commonDataDir**+"Rasters*/LUraster.tif*".

d. Variable *TABLENAME_RASTER_SWAT_ID_LOOKUP* contains a location of the **defaultPGDB *landuse*** shema table of remapping of the SWAT landuse classes to pixel valies (*FIELDNAME_RASTER_ID*) of the raster. Default value ***landuse_swat_raster_lookup.***

e. Variables *FIELDNAME_LANDUSENAME_LU, FIELDNAME_LANDUSENAME_RM, FIELDNAME_SWATLUNAME* provides a possibility for user to enter the fieldnames used for *LUCODE*, *globalcode* and *SWATCODE.*

f. Variable *TABLENAME_RASTER_LUGROUPS_ID_LOOKUP* contains a location of the **defaultPGDB** temporary table (for debug purpose only)

3. Description of the soil geospatial data.

a. Variable *SoilFile* contains the location of the **defaultPGDB** table representing the soil classes. This table should contain at least an attribute *SoilCode* representing the soil name. Default value of this variable is ***(soil, union)***.

b. Variable *SoilvsSWATTable* contains the location of the table of remapping of the original soil classes into the SWAT soil types. This table should contain at least two attributes (1) *DetailedCode* – the soil name corresponding to *SoilCode* in the soil feature class and (2) *SWATCODE* – the SWAT+ soil name corresponding to the names in table ***usersoil*** from ***soil*** *schema*. Default value of this variable is ***(soil, globallookup)***.

c. Variable *SoilRasterFile* is a name of the soil raster file that will be prepared for classification of HRUs. Default value of this variable is **commonDataDir** +"Rasters*/soilraster.tif*".

d. Variable *SoilFileVars* provides possibility for user to enter the fieldnames used for *SoilCode, DetailedCode* and *SWATCODE.*

4. Subbasin and watershed definition data block provides geospatial data of catchments and river segments, catchment connection to watersheds, river network connectivity, outlets and inlets.
   a. We use the following meanings (definitions):
      i. **Subbasin** is a polygonal entity representing one river catchment and eventually one LSU and aquifer in SWAT+ setup.
      ii. **River segment** is a polyline representing the main river in one subbasin.
      iii. **River node** is a point that is placed on either of the ends of a river segments.
      iv. **Watershed** is a collection of spatially connected subbasins representing one river basin that corresponds to one SWAT+ setup.
      v. Two level naming of watersheds is provided allowing to group them into **Basins**.
      vi. **Outlet** is an object that is placed on any external outflows from the river network. Outlets should be placed on all outflows.
      vii. **Inlet** is an object at the external inflows into the river network.
      viii. **External catchment** is a representing additional area outside the modelling domain that is not directly modeled, but rather accounted by increased area of receiving catchment.
      ix. **Water transfer** is an object which allows for [fractional] water transfer from a predefined outlet to predefined river segment.
   b. The following variables are used for definition of rivers, subbasins and watersheds:
      i. Location of the subbasin table is provided by variable *CatchmentFile*. Default value of it is ***(catchments,catchments)***. One entity of the file should correspond to one SUBBASIN. Subbasins should fill all modeling domain (Latvia territory). They should not overlap, and the boundaries of the neighbouring catchments should topologically correspond to each other. Each subbasin should correspond to only one river segment. Integer attribute *CatchmentID* must be present, and it must correspond to unique catchment identifier.
      ii. Location of polyline feature class representing river segments is provided by variable *RiverSegmentFile*. The default value of this variable is ***(catchments,riversegments)***. Polyline segment direction should have the same direction as river flow. Rivers should be split at catchment boundaries. Only one node could be placed at each particular river confluence or splitting point. Following attributes must be present in the *RiverSegmentFile*:

1. *segmentID* – an unique integer river segment identifier of the same value as corresponding *catchmentID.*
2. *catchmentID – an* id of *the* catchment that contains the river segment.
3. *FlowTo* – a *segmentID* of the receiving river segment or *outletID* of the receiving outlet.
4. *riverID* - an id of the river in the supplemental rivers table

iii. The location of the point feature class representing river nodes is provided by variable *RiverNodeFile*. Deafult value of this variable is ***(catchments, nodes)***. An attribute node_ID representing unique mode identifier must be present.

iv. Location of the watershed table is provided by variable *WatershedTableFile*. The default value of this variable is ***(catchments, watersheds)***. It should contain three attributes:

1. *watershedID* is a unique watershed identifier;
2. *BasinName* – a basin name to which the watershed belongs;
3. *watershedName* – name of the watershed.

v. Location of table that provides a grouping of the subbasins into watersheds is provided by variable *CatchmentsByWatershedTableFile*. The default value of this variable is ***(catchments, catchmentsbywatersheds)***. It must contain two attributes:

1. *watershedID* – is the watershed identifier;
2. *catchmentID* is a catchment identifier of the subbasin. One subbasin could belong to only one watershed.

vi. Location of a table of the outlets is provided by variable *OutletTableFile*. Default value of this variable is ***(catchments, outlets)***. It must contain two attributes:

1. *outletID* – a unique integer identifier of outlet, it must be different from any of the river segment identifiers;
2. *outletName* is the name of the outlet.

vii. Location of a table of external inlets is provided by variable *TransboundaryInletFile*. Default value of this variable is ***(catchments , inlets)***. It must contain the following three attributes:

1. *inletID* – a unique integer identifier of inlet, it must be different from any of the river segment identifiers and any outlet identifiers;
2. *inletName* – the name of inlet;
3. *segmentID* – river segment identifier to which the inflow is directed.

viii. Location of table of external catchments is provided by variable *TransboundaryCatchmentFile*. The default value of this variable is ***(catchments, transboundary)***. Two attributes must be present:
   1. *catchmentID* – a catchment identifier of the subbasin with external contribution;
   2. *AddedArea* – an area in km^2 of the catchment located outside of the modelling domain.

ix. The location of the table that provides possibility of the water transfer between an outlet and another segment is given by variable *WaterTransferTable*. The default value of this variable is ***(catchments,watertransfer)***. Following fields must be present in the table:
   1. *transferID* – a unique identifier of an object;
   2. *outletID* – an identifier of the outlet from which the water transfer occurs;
   3. *flowTo* – a river segment identifier of the receiving river segment;
   4. *multiplier* – a coefficient by which an amount of the transferred water in outlet is multiplied.

x. Variable *WatershedDefinitionVars* provides possibility for user to enter the fieldnames used in subbasin and watershed defition.

xi. Variable *CatchmentIDRaster* represents a raster file containing spatially ditributed catchment IDs that are prepared for classification of HRUs. The default value of this variable is **dataDir**+*"Rasters/cmentIDraster.tif"*. Database table pointed by *CatchmentFileVars* item *"TABLENAME_RASTER_SWAT_ID_LOOKUP"* contain a remapping of the catcment id to pixel valies.

5. The block of HRU definition variables:
   a. Variable *globalHRUraster* shows the location of global HRU raster that is prepared by HRU definition script. The default value of this variable is **dataDir**+*"Rasters/globalhru.tif"*.
   b. Location of a table where the mean slope of each of the HRUs is stored is provided by variable *HRUslopeTable.* The default value of this variable is *(hru,hruslopes*).
   c. Location of a table where the mean elevation of each of the subbasins is stored is provided by a variable *CatchmentelevationTable*. Default value is ***(hru,catchelev)***.
   d. Minimal HRU area in hectares is provided by a variable *HRUDefinitionVars["MinimalHruArea"]*. Its default value is 5.0.

6. Observation data. It should be provided in PAICSWAT format. Any observation data consists of two tab-delimited text files. Both files should be in the same directory and have the same basename. The file with extension *.sta* contains the station coordinates and names. Columns of .sta file are: *stationID Name X Y* (*stationID* - station integer ID, *Name* - station name, *X*, *Y* - station coordinates). Coordinates must be in a coordinate system defined by variable *defcoordSystTxt.* The file with extension *.txt* contains the daily measurements of respective parameters
   a. Meteorological parameters: a variable *WEATHER_DATA* provides the location of meteorological observation file. Its default value is **dataDir**+*"observations/weatherdata.txt"*. The following observation data should be provided in the columns of the corresponding .txt file: *Station Date T Tmin Tmax Prec W10 RHUM Solar:*
      i. Station – a stationID corresponding to ID in .sta file,
      ii. DATE – date in yyyy.mm.dd format,
      iii. T, Tmin, Tmax - daily average, minimal and maximal temperatures, in degC,
      iv. Prec - daily precipitation amount in mm/day,
      v. W10 - daily average wind speed at 10m in m/s,
      vi. RelHum - daily average relative humidity in %,
      vii. Solar - daily amount of solar radiation in MJ/m^2/day.
   b. Variable *Q_DATA* provides a location of daily discharge observation data. Its default value is **dataDir**+*"Observations/qobs.txt"*. Columns of the corresponding .txt file are *StationID Date Q*:
      i. StationID - stationID corresponding to ID in .sta file,
      ii. DATE – date in in yyyy.mm.dd format,
      iii. Q – discharge in m^3/s.
   c. Variable *WQ_DATA* provides location of daily water quality observation data. Its default value is **dataDir**+*"Observations/wqobs.txt"*. Columns of the corresponding .txt file are: *StationID Date Flow SS DO BOD7 NH4-N NO2-N NO3-N N mineral N total PO4-P P total N-Org P-Org*:
      i. StationID - stationID corresponding to ID in .sta file,
      ii. DATE – date in yyyy.mm.dd format,
      iii. Flow – discharge in m^3/s,
      iv. The rest of columns are the values of corresponding concentrations of water quality parameters in mg/l.
   d. Variables *Q_STATIONS_CATCHMENTS* and *WQ_STATIONS_CATCHMENTS* provide the location of tables where discharge and water quality measurement station corresponding to the subbasins will be stored. Variables

OBS_FIELD_STAID and *OBS_FIELD_STANAME* define the fieldnames of above-mentioned tables corresponding to observation station ID and station name.

7. Tile drainage data.
   a. Location of a table with polygon geometry column representing meliorated territories is provided by variable *Drainage.TableShapes*. The default value of this variable is ***(mel_dr10lt,melior_dissolved)***.
   b. Variable *Drainage.FIELDNAME_TYPE* is the fieldname of a text attribute in *Drainage.TableShapes* containing type of drainage. The default value of this variable is *'TIPAS'*.
   c. Variable *Drainage* .MELIORATION_ALLOWED_TYPES is the list of values of field *Drainage.FIELDNAME_TYPE* for which tile drainage is activated. The default values of this variable is *['D']*.
   d. Variable *DrainageVars* represent parameters of tile drainage.
      i. *DDRAIN* is a depth to subsurface drains [mm] (default – 1100.0),
      ii. *TDRAIN* is *a* base time to drain soil to field capacity (default – 24.0),
      iii. *GDRAIN* is a drain tile lag (default – 24.0),
      iv. *LIMREL* is a maximal part of HRU that is drained to activate the drainage (default - 0.05).
   e. Variable *Drainage.raster* contains a filename of the raster that will contain the drainage data. Default value of this variable is **dataDir**+*"Rasters/drainage.tif"*.
   f. Variable *Drainage.HRUDRAINAGE_TABLE* contains filename of a table where to store drainage data for each HRU. The default value of this variable is ***(hru, hrudrainage)***.

8. Atmospheric deposition data
   a. The location of the table with polygon geometry column representing atmospheric deposition is given by a variable *ATMDEP_POLYGONS*. The default value of this variable is ***(atm_deposition/atmdep2018)***.
   b. Variable *ATMDEP_BY_CATCHMENTS* provides location of the table with polygon geometry column where atmospheric deposition by catchments will be stored. Default value of this variable is ***(hru/ atmdepCatchm)***
   c. *Variable ATMDEP_VARS* lists the fieldnames of *ATMDEP_POLYGONS:*
      i. *FN_NH4_dry* - fieldname containing the dry deposition values of ammonia (mgN/m^2/year). Default – *"NH4_dry"*.
      ii. *FN_NH4_wet* - fieldname containing the wet deposition values of ammonia (mgN/m^2/year). Default – *"NH4_wet"*.
      iii. *FN_NO3_dry* - fieldname containing the dry deposition values of nitrate (mgN/m^2/year). Default – *"NO3_dry"*.

iv. *FN_NO3_wet* - fieldname containing the wet deposition values of nitrate (mgN/m^2/year). Default – *"NO3_wet"*.

v. *FN_precip* - fieldname containing the precipitation values as used for calculation of the wet deposition (mm/year). Default – *"precipitation"*.

vi. *FN_area* - fieldname containing shape area in m^2. Default – *"Shape_Area"*.

9. Buffer zone data
   a. Variable *TableShapes* of *BufferZone* class provides the location of the table with polygon geometry representing the buffer zones. The default value of this variable is ***(bufferstrips,polygons).***
   b. Location of the feature class for storing intersection of buffer zones with catchments is provided by field *Table_BY_CATCHMENTS*. The default value of this variable is ***(hru/ buffzones_catchm)***.
   c. Location of table for storing summary statistics about lengths and areas of buffer strip polygons by catchments is provided by a field *Table_STAT_BY_CATCHMENTS* = ("hru","buffzones_stat"). The default value of this variable is ***(hru,* buffzones_stat*)***.
   d. The field *COEFFICIENT* of *BufferZone* class is the value by which an average width of the buffer zone is multiplied to obtain the *FILTERW* parameter of *MGT* table. Default value 0.05.

10. Administrative county data
   a. Variable *countiesFC* provides a table with polygon geometry column containing administrative county boundaries (default value is ***(counties, counties)***. Variable *countiesIDField* contains the field name of unique integer county identifier. Default value is *"ID"*.
   b. Variable *countiesraster* provides file name of raster of county identifiers that will be prepared by script. The default value is **dataDir**+*"Rasters/counties.tif"*.
   c. *Variable countiesHRU* contains file name of raster where correspondence of HRUs to counties is stored. The default value is **dataDir**+*"Rasters/combi_cohru.tif"*.

11. River parameter data
   a. River slope
      i. Variable *nodeElevationFile* contains the name of a table where node elevations will be stored. Node elevations are used to calculate river slope. The default value of variable is ***(hru/ node_elevation)***.

ii. Variable *RiverSlopeBounds* define minimal and maximal allowed river slope. The default value *(0.0001, 0.1).*

b. *River width*

i. Variable *riverWidth_table* contains a name of the table where the calculated river width will be stored. The default value of this variable is ***(rivers, river_width)***. Variable *fieldnameWidth* is the name of the field where the river width will be stored.

c. River depth

i. Variable *riverDepthGrid* contains the name of grid of river depths. The default value of this variable is **dataDir**+*"Rasters/bath_5m.tif"*.

ii. Variable *riverFPPolygons* contains the name of table with polygon geometry column of the rivers obtained from grid of river depths. The default value of this variable is ***(rivers, polygon)***.

iii. Variable *riverPolyCatchmentIntersect* contains the name of table with polygon geometry column where river polygon intersection with subbasin polygon feature class will be stored. Default value of this variable is ***(rivers, riverpolyintersectcatchm)***

iv. Variable *segmentDepthTable* contains name of the table where river segment depth information will be stored. Default value of this variable is ***(river,segmentdepth)***.

d. River Manning's coefficients

i. Variable *ManningCoefFile* is name of a table with polyline geometry column where Manning's calibrated coefficient values is stored. The default value is ***(rivers,manning)***. Field name where Manning's coefficient is stored is provided by variable *fieldNameManningCoef*. The default value of it is *"n"*.

ii. Variable *segmentManningtable* is the name of a table where Manning's coefficients for river segments will be stored. Default value of this variable is *(rivers,segmentmanningcoef)*.

iii. The default value of Manning's coefficient is stored in a variable *riverDefaults["defaultManningCoef"]*. It is *0.03.*

12. Data related to management practices and fertilization

a. Field *PLANTFERT* of *Fert* class provides a location of a table name of the plant related fertilization table. This table must contain the following fields:

i. *SWAT_ID* (integer) – SWAT code of the plant, corresponding to crop table's field *SNAM.*

ii. *Plant_Name* (text) – name of the plant.

iii. *Method (integer)* – method number for calculation of the fertilization amount (1 or 2). Blank may be left in case fertilization is not calculated for the particular plant.
iv. *Std_yield* (double) – standard yield for the plant (tons/ha/year).
v. *N_Demand* (double) – nitrogen demand (kgN/ha/year).
vi. *P_Demand* – phosphorus demand (kgP/ha/year).
vii. *Preplant_N* – preplant nitrogen amount (kgN/ha/year).
viii. *Preplant_P* – preplant phosphorus amount (kgP/ha/year).

b. Field *LIVESTOCK* of *Fert* class contains the location of the MS Access database and tablename of the table storing the number of livestock units in the counties. This table must contain the following fields:
   i. *Id* (integer) – unique integer county identifier that corresponds to the *countiesIDField* of *countiesFC*.
   ii. livestock_units (double) – number of standard animal units in county.

c. Field *PLANNEDYIELD* of *Fert* class contains location of the MS Access database and tablename of table storing yield by county and crop. It must contain following fields:
   i. *SWAT_ID* (integer) - SWAT code of the plant, corresponding to the crop table's field *SNAM.*
   ii. *Id (integer)* - unique integer county identifier that corresponds to *countiesIDField* of *countiesFC*.
   iii. *Planned_yield* – yield amount in tons/ha/year.

d. Field *PLANTMGT* of *Fert* class provides the location of the MS Access database and tablename of table storing management practices for each of the SWAT landuse codes. This table must be in *MSAccess* format and contain the following fields:
   i. *SWAT_ID* (integer) - SWAT code of the landuse, corresponding to crop table's field *SNAM* or urban table's field *URBNAME.*
   ii. *LUClass* (text) – text identifier of landuse group. Possible values are *Agricultural*, *Barren*, *Forest*, *Pasture*, *Urban*, *Water*, *Wetland*.
   iii. *PHU* (double) – plant heat units to bring a plant to maturity. This variable applies if *LUClass* is *Agricultural.*
   iv. *PlantHU* (double)– base zero heat units for planting. This variable applies if *LUClass* is *Agricultural.*
   v. *HarvestHU* (double) – base zero heat units for harvest. This variable applies if *LUClass* is *Agricultural*. For winter crops it should be larger than 1.0, to indicate the next year.
   vi. *HarvestHU2* (double) – plant heat units for harvest. This variable applies if *LUClass* is *Agricultural.*

vii. *P_AplHU* (double) – base zero heat units for mineral phosphorus application.
viii. *Tillage1, Tillage2, Tillage3, Tillage4* (double) – these four fields represent the base zero heat units for application of 4 types of the tillage operations. For winter crops these fields should be larger than 1.0, to indicate the next year.
ix. *N_AplHU1, N_AplHU2, N_AplHU3* (double) – these three fields represent base zero heat units for application of the mineral nitrogen up to 3 times per year. For winter crops value of these fields should be larger than 1.0, to indicate the next year.
x. *N_Amount1, N_Amount2, N_Amount3* (double) – the fields contain fractions of total mineral nitrogen application up to 3 times per year. The total sum of fractions should equal to 1.0.
xi. *Man_apl_HU1, Man_apl_HU2* (double) – these two fields represent base zero heat units for application of manure up to 2 times per year. For winter crops the values larger than 1.0 indicates the next year.
xii. *Man_apl_amount1, Man_apl_amount2* (double) – the fields contain the fractions of the total manure application up to 2 times per year. The total sum of fractions should equal to 1.0.

e. Variable *landusegrouprаster* is the name of raster that will contain a spatial distribution of the landuse groups. The default value of this variable is **dataDir**+*"Rasters/lugroups.tif"*.
f. Variable *lugroupbycountyraster* is the name of raster that will contain geospatial distribution of the landuse groups by counties. The default value of this variable is **dataDir**+*"Rasters/lugrcounty.tif"*.
g. Variable *agriculturalgroups* is the list of land use groups for agricultural land calculations and fertilization. The default value is *["AGRICULTURAL","PASTURE"]*.
h. Variable *mineralFertilizertable* is the name of the table where calculated mineral fertilizer application per HRU will be stored. The default value of this variable is ***(hru, fertilizer)***.
i. Variable *management_vars* is a python dictionary containing the information related to the management:
i. *"past_bioinit"* is the initial biomass on the pastures (kg/ha). The default value of this variable is *5000.0*.
ii. *"past_phu"* is the number of heat units to bring the pastures to maturity (chosen larger than typical heat units per year to not mature the grass!). The default value of this variable is *3000.0*.

iii. *"frst_bioinit"* is the default initial biomass of forests (kg/ha). Default value of this variable is *100000.0*. It is used if *FORESTBIOMASS_TABLE* does not provide the information about the forest biomass for particular *SWAT_ID*.

iv. *"frst_phu"* – is heat units to bring forests to maturity (chosen larger than typical heat units per year to not mature the trees!). Default *3000.0*.

v. *"ID_MineralN"* – is id of the elemental nitrogen fertilizer from *fert* table in *SWAT_DB*. Default value of this variable is *1*.

vi. *"ID_MineralP"* – is id of the elemental phosphorus fertilizer from *fert* table in *SWAT_DB*. Default value of this variable is *2*.

vii. *"ID_Manure"* – is the id of the manure fertilizer from *fert* table in *SWAT_DB* (this fertilizer should have 1.0 units of total nitrogen!). The default value of this variable is *55*.

viii. *"ID_Tillage"* – is a python table of length 4 containing tillage identifiers for the 4 predefined tillage types corresponding to the tillage fields in *PLANTMGT_TABLE*. The default value of this variable is *(108, 109, 1, 111)*.

j. Information about the large farms:

i. Variable *largefarmsFC* is the name of the table with polygonal geometry column containing the geospatial location of "large farms" which are used for the fertilizer amount calculations. Default value of this variable is ***(landuse,large_farms)***

ii. Variable *largefarmsraster* is the name of raster that will contain geospatial location of "large farms". Default value of this variable is is **dataDir**+*"Rasters/rlargefarms.tif"*.

iii. Variable *hrulargefarms* is the name of the raster storing information of the large farms per each of the HRUs. The default value of this variable is is **dataDir**+*"Rasters/hrufarms.tif"*.

iv. Variable *largefarmFertilizerCoef* is a coefficient by which to multiply mineral fertilizer amount in "large farms". The default value of this variable is *1.2*.

k. Information about the fertilization regional coefficients in the *Fertilizer* class:

i. Field *TableShape* is the name of the table with polygonal geometry column that contains the regional multipliers to the calculated amount of the mineral fertilizer amounts. The default value of this variable is ***(fert, fertcoeff)***. Field name of multiplier is provided by field *Field_fertCoeff* .

ii. Field *TableCoeffByCatchments* is the name of the polygonal feature class where intersection of *fertilizerRegionalCoefficients* with subbasin

polygons will be stored. The default value of this variable is ***(hru, fertcoefbycatchm)***.

l. Variable *FORESTBIOMASS_TABLE* provides a location of the tablename of the table containing the initial forest biomass in kg/ha for forest landuses. It must contain the following fields:

i. *SWAT_ID* (integer) - SWAT code of the plant, corresponding to the crop table's field *SNAM.*

ii. *Biomass* (double) – forest biomass in kg/ha.

13. Initial nutrient concentration in groundwater:

a. Variable *GW_ConcN* is a Python dictionary which contains the initial concentrations (mg/l) of nitrate in the shallow aquifer per landuse groups. Default values are:

i. *"AGRICULTURAL": 2.50*

ii. *"PASTURE": 1.15*

iii. *"FOREST": 1.12*

iv. *"URBAN": 3.8*

b. Variable *GW_ConcP* is a Python dictionary which contains the initial concentrations (mg/l) of soluble phosphorus in the shallow aquifer per landuse groups. Default values are:

i. *"AGRICULTURAL":0.02*

ii. *"PASTURE":0.05*

iii. *"FOREST":0.06*

iv. *"URBAN":0.04*

v. Lake and reservoir related variables.

14. Reservoir and lake data.

a. Variable *TableShapes* of the class *LakesReserv* is the name of polygonal feature class containing the reservoir and lake data. Default value of this variable is ***(catchments, lakes_reserv)*** This table must contain at least the following columns:

i. *Snpl* - surface area at the normal water level (ha). If value of *Snpl* is zero, it is assumed that for a particular polygon *Snpl, Savl, Vnpl and Vavl* are not given.

ii. *Savl* - surface at the highest water level (ha).

iii. *Vnpl* - volume at the normal water level (in 1000 of m^3).

iv. *Vavl* - volume of the water needed to fill the reservoir to the emergency spillway (in 1000 of m^3).

v. *Depth* - average depth of lake/reservoir (m). If it is zero, then it is assumed, that the average depth is not given.
vi. *Shape_area* – the area of lake or reservoir polygon in m^2, corresponding to the geometric area of a feature.

b. Fields *Field_** of the class *LakesReserv* contains the column names of above-mentioned fields of *TableShapes*.
c. Variable *lakesreservoirsbycatchments* is a name of the polygon feature class which will contain the intersection of the *reservoirlakefc* with the subbasin polygons. The default value of this variable is ***(hru,lakesreservoirs)***.
d. Variable *reservoirParams* contains the default depth of reservoirs (used for volume calculations if volumes are not given) and default difference between the highest and normal waterlevel (used for highest volume calculations if they are not given). The default values of this variable are *3.0* and *0.3*, respectively.
e. Variable *lakeParams* contains the default depth of lakes (used for volume calculations if volumes are not given) and the default difference between the highest and normal waterlevel (used for highest volume calculations if they are not given). The default values of this variable are 2.*0* and *0.2*, respectively.
f. Variable *reservoirReleasetime* is a time in days to release the reservoirs from maximum to normal level. It is used for calculation of the average daily release rate.

15. Parameters for annual average precipitation correction:
a. Variable *AnnualPrecipitationRaster* is the name of a raster containing the geospatial distribution of the annual average precipitation values. The default value of this variable is **dataDir**+*"Rasters/Precip.TIF"*.
b. Variable *AnnualPrecipitationStationtable* is the name of a table containing the annual precipitation values for observation stations, corresponding to the stations in *WEATHER_DATA*. The default value of this variable is ***(obs, statpcp)***. Variable *AnnualPFieldNames* determines the field names for the station ID and the annual average precipitation.
c. Variable *catchmentPrecipitationtable* is a name of the table where the catchment annual average precipitation will be stored. The default value of this variable is ***(hru, precip_by_catch)***.

16. Wetland parameters:
a. Variable *WetlandsVars* contains:
i. *"wet_normD" -* average normal depth of water in the wetlands (m). The default value of this variable is *2.0*.

ii. *"wet_maxD"* - average maximal depth of water in the wetlands (m). The default value of this variable is *3.0*.
iii. *"WetlandLandUseIDs"* is a list of landuse codes that define a wetland. The default value of this variable is *["WETN", "WETF"]*.

17. Pointsource variables:
    a. Variable *PS_catchment_table* is the name of the table that will contain *catchmentID* of the subbasin for each of the pointsources. Default value of this variable is ***(point_sources, ps_catchment)***.
    b. Variables *Field_PS_ID* and *Field_PS_Name* are the field names of the point source integer ID and point source name in *PS_catchment_table*.
    c. Variable *PS_MONTHLY_FILE* contains the name of the monthly point source file in PAICSWAT format. The default value of this variable is **dataDir**+"Pointsources*/monthly.txt"*. The monthly pointsource data consists of two tab delimited text files. Both files should be in the same directory and have the same basename.
        i. Columns of .sta file are: *pointsourceID, Name, X, Y, sourceID* (*pointsourceID* – point source integer ID, *Name* - point source name, *X, Y* - point source coordinates, *sourceID* - text identifier of point source that relates it to the original data). Coordinates must be in a coordinate system defined by the variable *defcoordSystTxt.*
        ii. The file with extension *.txt* contains monthly point source values. Columns of this file are:
            1. *ID* – the point source integer ID as defined by *.sta* file.
            2. *DATE* – date of point source in *yyyy.mm.dd* format. Day for monthly point sources is always 15.
            3. *SS, BOD7, NO3, NO2, NH4, NTOT, PO4, PTOT, NORG, PORG* are the loads of suspended solids, BOD7, N-NO3, N-NO2, N-NH4, total nitrogen, P-PO4, total phosphorus, organic nitrogen, and organic phosphorus in g/day.
            4. *Discharge* – flow of point source in m^3/day
            5. *rho_SS, rho_BOD7, rho_NO3, rho_NO2, rho_NH4, rho_NTOT, rho_PO4 rho_PTOT, rho_NORG, rho_PORG* are the concentrations of suspended solids, BOD7, N-NO3, N-NO2, N-NH4, total nitrogen, P-PO4, total phosphorus, organic nitrogen, and organic phosphorus in mg/l
            6. *year*, *month* are the calendar year and month.

d. Variable *PS_YEARLY_CATCHMENT_TABLE* is a file name of MSAccess database and table name where catchmentIDs of pointsources defined by *PS_YEARLY* will be stored.
e. Variable *PS_YEARLY* contains file name of the table name of yearly pointsource data of those pointsources that are not included in the *PS_MONTHLY_FILE*. This table must contain following fields, the field names are defined in variable *PS_YEARLY_FIELDS*:
    i. *ID* (integer) - unique id.
    ii. *year* (integer) – year of pointsource loading.
    iii. *name (*text) - full name of point source.
    iv. *type (*integer) – type of point source.
    v. *nutrient* (text) – type of substance (either of *SS, BOD7, NO3, NO2, NH4, NTOT, PO4, PTOT).*
    vi. *concentration* (double) – value of concentration of corresponding substance in mg/l.
    vii. *discharge* (double) – value of discharge of pointsource in 1000 m^3/y.
    viii. *X* (double) – x coordinate of point source in coordinates defined by *defcoordSystTxt.*
    ix. *Y* (double) – y coordinate of point source in coordinates defined by *defcoordSystTxt.*

18. Water users related variables.
    a. *WUS_FILE* contains name of file in PAICSWAT format that contains data about water users. The default value of this variable is **dataDir**+"Pointsources*/WaterUseMonthly.txt*".
    b. Water users' data consists of two tab delimited text files (*.sta* and *.txt*). Both files should be in the same directory and have the same basename.
        i. Columns of *.sta* file are: *waterUserID, Name, X, Y, sourceID* (waterUserID – water user integer ID, Name – water user name, x,y – water user coordinates, *sourceID* - text identifier of point source that relates water user to point source). Coordinates must be in a coordinate system defined by a variable *defcoordSystTxt.*
        ii. The file with extension *.txt* contains water usage values. Columns of this file are:
            1. *ID* – water user integer ID as defined by *.sta* file.
            2. *DATE* – date of point source in *yyyy.mm.dd* format.
            3. *WaterUse* – water usage in m^3/day

19. Block of the SWAT+ setup related variables (class *SWAT2012DB*).

a. A weather generator input table must contain a field ID, that corresponds to station ID in *WEATHER_DATA* file. Field *WGN_TABLE* provides location of the schema, tablename and fieldname for station.
b. *Usersoil* table provides the soil parameters. It must contain soil name field *SNAM*. Variable *USERSOIL_TABLE* provides location of the table.
c. The crop table provides crop parameters corresponding to the crops and the forest plants in the landuse table. It must ontain the crop abbreviation field *CPNM*. Field *CROP_TABLE* provides a location of the table.
d. Urban table provides the urban parameters corresponding to the urban landuses in the landuse table. It must contain urban landuse abbreviation field *URBNAME*. Field *URBAN_TABLE* provides a table. Variable *URBAN_PLANT_CODE* provides a crop code from the crop table that will be used for HRUs with urban landuse.
e. *cst_user* table stores the information for generating of the weather forecast scenarios (CST). The format of the table is identical to the weather generator input table. The forecast regions correspond to the meteorological stations. Field *CST_TABLE* provides location of the table and fieldname for station.
f. Variable *SimulationPeriod* allows the modification of the simulation start year and date, number of years simulated, and a number of years for the simulation warm-up period.
g. Variable *defaultCoefficients* provides possibility for the user of the script to enter the default global values of SWAT+ variables. This variable is a python list of tuples. Each tuple consisting of four values:
    i. type of variables – either "*V*" if a constant value is entered or "*R*" if a value relative to initial value is entered. Entering of relative values of the soil coefficients (*SOL* table) and *CN2* coefficient is supported.
    ii. Name of SWAT+ table.
    iii. Name of SWAT+ variable.
    iv. Value of SWAT+ variable.

20. Block of variables related to transboundary inflows.
    a. Variable of type Boolean *transboundaryInflow* determines whether the inflow is taken from measured data (if *False*) or from SWAT+ model (if *True*). The default value of this variable is *False*.
    b. Variables *external_XXX* provide the location of the output files generated by SWAT+ model for XXX rivers. Variable *externalInflowDir* contains the folder name of external SWAT+ setup.
    c. Variable *transBoundaryInlets* is the Python dictionary, containing the list of the identifiers of the external inlets as provided by *TransboundaryInletFile*. For each

of the identifiers *stationID* for discharge and water quality stations must be provided. The measured data for these stations from *Q_DATA* and *WQ_DATA* will be used as a transboundary inflow in case if *transboundaryInflow* is *False*.

3.Description of subroutines of script

The following scripts form the modelling system.

3.1. Geospatial data processing

All possible geospatial operations are included in and called from *GISoperations.py*. The specific actions (particular scripts) to be run are specified in *main.py*. There are the following steps of the geospatial data processing:

1. Calculation of the catchment centre and area adds the X and Y coordinate of the centre point in LKS_92 (X and Y) and WGS84 (LON and LAT) coordinate systems and area (AREA) to the polygon feature file created during the spatial division of the model

Used script: catchments.py, function CalculateCatchmentCentroidsAndArea().
Input/output data: *CatchmentFile*

2. Prepare *RiverNodeFile* table and adds the *nodefrom*, *nodeto* columns to *RiverSegmentFile* table.

Used script: catchments.py, function AddSegmentFirstLastPoints ().
Input/output data: *RiverSegmentFile,*
Output data: *RiverNodeFile*

3. Calculation and adding the column segment length (LENGTH) to the table created during the spatial division of the model.

Used script: catchments.py, function CalculateSegmentLength().
Input/output data: *RiverSegmentFile*

4. Calculation of the river depth either by (1) using a raster provided from the Lithuanian FLOOD project (*riverDepthGrid*) and polygon derived from the grid (*riverFPPolygons*) for the territories covered by the FLOOD project or by (2) using a formula depending on the river basin size. Creating a table for each segment (*segmentdepthtable*).

Used script: riverdata.py, function PrepareRiverData().
Input/output data: *riverDepthGrid*, *riverFPPolygons*, *segmentdepthtable* .

5. Calculation of the river width using water polygons (*RiverPolygons*) and creating a table of the river width for each segment (*riverWidth_table*).

Used script: riverdata.py, function PrepareRiverData().
Input/output data: *RiverPolygons*, *riverWidth_table*.

6. Joining catchments (*CatchmentFile*) to the hydrological regions (*regionFC*) and adding it to the catchments. The shape file of the hydrological regions of Lithuania is provided by the Client.

Used script: regionalization.py, function PrepareCatchmentRegions().
Input/output data: *CatchmentFile, regionFC*.

7. Calculation of elevation at the river nodes by extracting DEM (DEMFile) values at the nodal points for the calculation of river slopes (*RiverNodeFile*).

Used script: riverdata.py, function ElevationAtNodes().
Input data: *DEMFile, RiverNodeFile*.
Output data: *nodeElevationFile*

8. Slope calculation from the digital elevation model is done by using script. creating a raster which contains slope values on 5x5 m grid.

Used script: HRUrasterOps.py, function CalculateSlopeRaster.
Input data: *DEMFile*
Output data: *SlopeFile*

9. First step of HRU delineation creates four rasters with the same resolution and spatial extent as the slope raster SlopeFile:

    a. Landuse raster is created using a landuse polygon feature (*LanduseFile*) containing landuse codes and a lookup table (*LUvsSWATTable*) containing a link between the landuse codes and SWAT+ codes found in the SWAT+ crop and urban databases.

    b. Soil raster is created using a soil polygon feature (*SoilFile*) containing soil codes and a lookup table (*SoilvsSWATTable*) which contains a link between the soil codes and SWAT+ codes found in the SWAT+ user_soil database.

    c. Slope-class raster is created using the slope raster (*SlopeFile*, prepared in above step 8) and predefined slope classes (defined in settings.py.SlopeClasses).

    d. Catchment raster is created using catchment polygon feature file (*CatchmentFile*) modified in above step 1.

Used script: HRUrasterOps.py, function prepareHRUs ("Landuse", "Soil", "Slopeclasses", "CatchmentID")
Input data: *LanduseFile, LUvsSWATTable, SoilFile, SoilvsSWATTable, SlopeFile, CatchmentFile*.
Output data: *LanduseRasterFile, SoilRasterFile, slopeClassRaster* and *CatchmentIDRaster*.

10. The second step of HRU delineation intersects the created Landuse, Soil, Slope-class and Catchment rasters using the same Python script as in step 9. It creats a HRU raster (*globalHRUraster*) each cell of which contains a combination of landuse, soil, slope-class and catchment as well as a unique value for each of the combinations. This value defines a single HRU. After this the average slope value for each HRU is calculated and stored in a table.

Used script: HRUrasterOps.py, function prepareHRUs ("GlobalHRUs")
Input data: *LanduseRasterFile*, *SoilRasterFile*, *slopeClassRaster*, *CatchmentIDRaster*.
Output data: *globalHRUraster*.

11. Other data which needs to be added to the HRUs are processed creating tables containing the unique HRU identifier and corresponding parameter values:

    a. Melioration data from the melioration polygon feature (*Drainage.TableShapes*) is intersected with every HRU and a value of the fraction meliorated in each HRU is added to a table (*Drainage .HRUDRAINAGE_TABLE*).

    b. County data from the county polygon feature (*countiesFC*) is intersected with every HRU and a value of the county for every HRU added to a table (*countiesHRU*).

    c. Large farm data from the landuse reported data dissolved for farms which are larger than 500ha (*largefarmsFC*) is intersected with every HRU. A cell count of the large farms in each HRU is calculated and a table is created (*hrulargefarms*).

Used scripts: HRUrasterOps.py, function prepareHRUs ("Drainage", "Counties", "Regions"), management.py, function prepareLargeFarms().
Input data: globalHRUraster, *Drainage.TableShapes*, *countiesFC*, *largefarmsFC*.
Output data: *Drainage .HRUDRAINAGE_TABLE* , *countiesHRU*, *hrulargefarms*.

12. The geospatial data which needs to be added at the catchment level is processed:

    a. A polygon feature (*AtmDep.Table_BY_CATCHMENTS*) containing atmospheric deposition values in every catchment is created using the atmospheric deposition data (*AtmDep.TableShapes*) and catchment polygon feature (*CatchmentFile*).

    b. Buffer strip polygon feature *BufferZone.TableShapes* is processed creating a table containing buffer zone statistics in every catchment (*BufferZone.Table_BY_CATCHMENTS*) by intersecting the buffer strip areas with the catchment polygons and calculating the average buffer strip width in the catchment.

c. Parameter depending on fertilization region (*Fertilizer.TableShape*) is added to each catchment and table for each catchment created (*Fertilizer.TableCoeffByCatchments*).

d. Hydrological region data from the river basin polygon feature (*regionFC*) is intersected with every HRU and a value of the region for every catchment is added to a table (*tableCatchmentRegion*).

e. Data about lakes and reservoirs are prepared for each catchment by intersecting the lake and reservoir data (*LakesReserv.TableShapes*) with catchments (*CatchmentFile*) and spatially joined with segments (*RiverSegmentFile*) creating a feature (*lakesreservoirsbycatchments*) which contains the information about the catchment to which the reservoir or pond belongs and if it is on a modelled reach.

f. Average annual catchment precipitation is calculated from the annual precipitation raster (*AnnualPrecipitationRaster*) and a table created with a value for each catchment (*catchmentPrecipitationtable*).

Used scripts: atmosphericdeposition.py, function prepareAtmosphericDepositionLayers(),
bufferstrips.py, function prepareBufferStripLayers(),
lakesReservoirs.py, function prepareLakesByCatchments(),
precipCorrection.py, function calcCatchmentPrecipitation().
Input data: *globalHRUraster*, *CatchmentFile*, *AtmDep.TableShapes* , *BufferZone.TableShapes* , *Fertilizer.TableShape* , *regionFC*, *LakesReserv.TableShapes*, *RiverSegmentFile*, *AnnualPrecipitationRaster*.
Output data: *AtmDep.Table_BY_CATCHMENTS*, *Fertilizer.TableCoeffByCatchments*, *catchmentsRegions*, *lakesreservoirsbycatchments* , *catchmentPrecipitationtable*.

13. Point sources are assigned to the catchments using the point source data file (*PS_MONTHLY_FILE*) and creating a table which contains the information to which catchment each point source belongs (*PS_catchment_table*).

Used scripts: pointsources.py, function pointSourcesbyCatchments()
Input data: *PS_MONTHLY_FILE*
Output data: *PS_catchment_table*

14. Landuse updates are prepared for landuse groups per catchment. We use two CORINE landuse polygons (fields *catchments_1*, *catchments_2* of class *LanduseUpdate*). The polygons must contain the defined landuse groups (*LanduseUpdate.groupfield*). A table containing information for a landuse update (*LanduseUpdate.update_fractions*) is created.

Used scripts: lup.py function prepareLUPbyCatchments()
Input data: *LanduseUpdate.catchments_1*, *LanduseUpdate.catchments_2*

Output data: *LanduseUpdate.update_fractions*

15. The geospatial data processing is finalised by checking whether the Landuse and Soil SWAT+ codes included in the HRU raster (globalHRUraster) are found in the SWAT+ database (class *SWAT2012DB*).

Used scripts: checks.py, functions checkLanduseSWATtable, checkSoilSWATtable.
Input data: *globalHRUraster, SWAT2012DB*

3.2. Creating SWAT+ modeling system

The rest of the operations for the creation of the SWAT+ model is done without using geospatial tools and mostly includes the creation, filling and updating of SWAT+ parameters in the different SWAT+ Watershed databases.

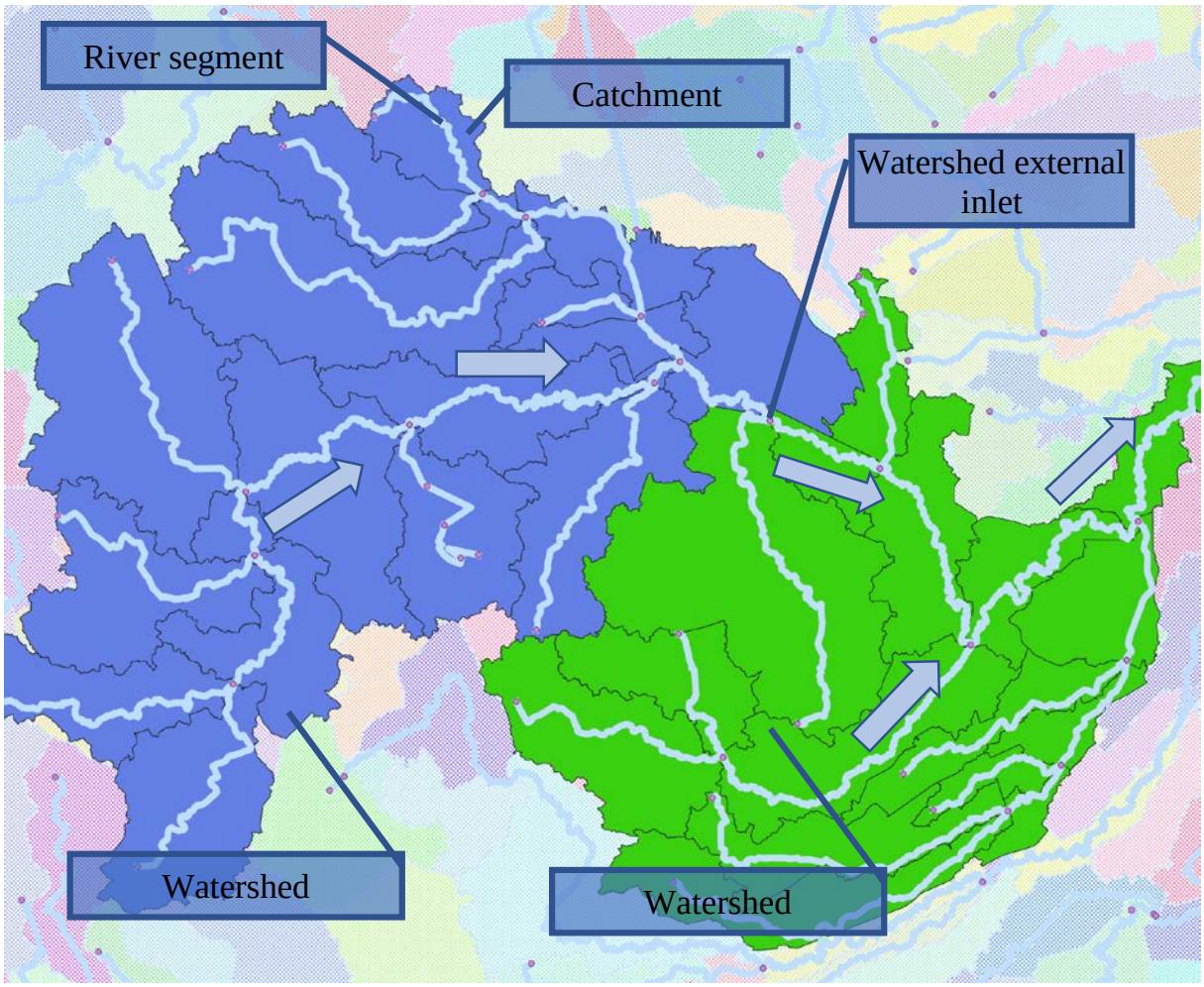


Figure 2. Spatial object mapping to SWAT+ objects and connections.

Spatial objects, their mapping and connections in SWAT+ are as follows, see Figures 2-3 and Bieger et al (2017):

- Watershed is a set of connected river segments or catchments. It is represented as SWAT+ setup. It may depend on other watershed (upstream rivers). Upstream rivers are represented in SWAT+ as point sources (recalls).

- River segment has one catchment. River segments are represented in SWAT+ as channel.

- Each catchment has the SWAT+ aquifer and landscape unit.
- Each watershed has the SWAT+ deep aquifer.
- Each aquifer is connected to its river segment. Additionally, an aquifer is connected to a deep aquifer.
- Catchments are subdivided into hydrological response units HRU. HRU are connected to the river segment by routing unit (RTU). There is one RTU per catchment.
- Each RTU is connected to the river segment and to the aquifer.

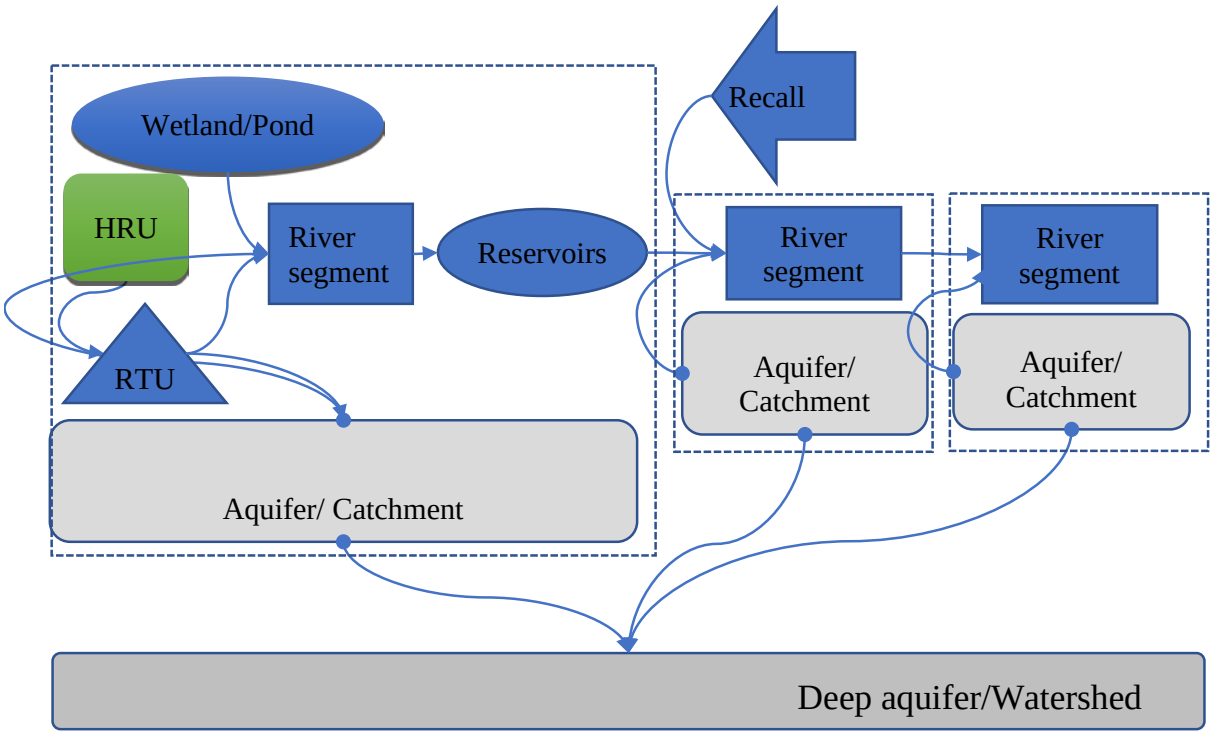


Figure 3. Scheme of SWAT+ objects.

- All routing units within a watershed has the same parameters.
- Ponds and wetlands are represented in SWAT+ catchment as wetlands and connected to a river segment.
- Reservoirs are connected to the first downstream river segment. It is one river segment connected to reservoirs.

The sequential steps of automated model generation are as follows:

1. To start building the databases the model must be split into watersheds and the catchments and rivers must be assigned to them creating the model geometry. For this we need to read all the information required for the creation of the networks. The script *Catchments.py.ReadAll(CatchmentInfo)* is used to read the following data prepared for the further processing:

    a. River segments prepared as a line feature containing information of connection between the river segments and the catchment they belong to as well as the ID to which river belongs (*RiverSegmentFile*) is read.

    b. Nodes are read as point features which correspond to river endpoints (*RiverNodeFile*).

    c. Table containing rivers and their IDs (*RiverTableFile*) for connecting the river names to river segments is read.

    d. Table containing the definition of watersheds, their name and ID (*WatershedTableFile*) is read.

    e. Table containing the connection between catchments and Watersheds including all catchments and the ID of the corresponding Watershed (*CatchmentsByWatershedTableFile*) is read.

    f. Table containing all the outlets of river segments that flow out of the modelling system which includes a unique identifier corresponding to the FlowTo field in the river segment file (*OutletTableFile*) is read.

    g. Table containing all the inlets into the system from transboundary sources containing an ID of the inlet as well as the ID of the river segment it flows into (*TransboundaryInletFile*) is read.

    h. Table containing all the catchments which have a part of their area outside of Lithuania containing the ID of the catchments and area outside of the Lithuanian territory (*TransboundaryCatchmentFile*) is read.

    i. Table containing all special water transfer points containing the outletID from which the water is transferred, the river segment to which the water is added and a coefficient by which the amount of water transferred must be multiplied is read.

Used scripts: Catchments.py function ReadAll(CatchmentInfo)

Input data: *RiverSegmentFile*, *RiverNodeFile*, *RiverTableFile*, *WatershedTableFile*, *CatchmentsByWatershedTableFile*, *OutletTableFile*, *TransboundaryInletFile*, *TransboundaryCatchmentFile*.

2. All other information needed for the updating of SWAT+ databases is read:

    a. Table containing the weather stations for the model including the coordinates (LKS 92) and names of the stations .sta file (*WEATHER_DATA*) prepared during weather data processing is read.

    b. File containing meteorological observations .txt file (*WEATHER_DATA*) prepared during weather data processing is read.

    c. Table of the raster containing the information from HRU delineation (*globalHRUraster*) including Landuse codes, Soil code, cell size of the raster is read. During this procedure processing of the HRUs is performed.

        i. The HRUs that do not belong to any of catchments are excluded.
        ii. The HRU with the largest size is found in each catchment.
        iii. All HRUs smaller than 1 ha are removed, except if there are no HRUs left after this, than the biggest is added.
        iv. HRU are appended to catchments and the local number in catchment is assigned for each of the HRUs left.
        v. Calculation of total area of the HRUs left is done.
        vi. Closest weather station is assigned to each catchment.

    d. Table containing drainage fraction for each HRU (*Drainage .HRUDRAINAGE_TABLE*) created during the processing of geospatial drainage data is read.

    e. Table containing atmospheric deposition in catchments created during the processing of the geospatial data on atmospheric deposition (*AtmDep.BY_CATCHMENTS*) is read.

    f. Table containing the width of the bufferzones in catchments (*BufferZone.Table_STATISTICS_BY_CATCHMENTS*) created during processing of geospatial data on buffer strips is read.

Used scripts: Catchments.py function ReadAll ("HRU","METEO")
Input data: *WEATHER_DATA*, *globalHRUraster*, *Drainage .HRUDRAINAGE_TABLE* , *AtmDep.BY_CATCHMENTS* , *BufferZone.Table_STATISTICS_BY_CATCHMENTS* .

3. All Watershed databases are created together with the respective SWAT+ file structure. To do this the following sequential steps are taken using the script sqliteSetup.py.

a. File structure is created containing the name of the Watershed as the project folder.

b. An empty database is created for each Watershed.

c. Table structure is copied to empty database from the template database (*swatplus_datasets*).

d. insert_routing_units() – routing units associated with HRU and LSU are created.

e. FillSolParameters() - fills the soil parameters from *usersoil* table containing the default parameters for every soil class. River segments are added.

f. insert_channels() - updates channel parameters.

g. insert_reservoirs() - updates reservoir parameters.

h. insert_recall() - monitoring point with the point source type is filled for each river end point (applies monthly point source data to the point sources added to the catchments) and inlets for the Watershed are filled (updates transboundary inlets into SWAT+ setups; the inflows from transboundary rivers can be chosen from either the observation data or from SWAT+ models).

i. insert_hrus() - HRU tables are filled with the relevant HRU data.

j. insert_aquifers() - aquifer data inserted.

k. Outflow(connections) of RTU, channel, aquifers and reservoirs are created. insert_connections()

l. insert_lsus() - landscape units are created.

m. SetSimulationPeriod() updates the simulation period using the simulation period defined in the settings.py.

n. FillWeatherTables() updates the tables related to weather data.

o. FillWgn() updates the weather generator table (Section 2.3) with all the parameters and stations included in the model.

p. FillMGTParameters() fills default values of mgt1.cn2 from usersoiltable, sets the plant identifier plant_id dependent on the landuse and urban tables, sets urban landuse (IURBAN and URBLU) flag for identifying urban teritories.

q. updateManagement() fills values for management practices and fertilization.

r. updatePrecipCorrection() fills parameters RFINC fields in the SUB table by comparing the annual long term precipitation of catchment by the corresponding annual precipitation form the station assigned.

s. updateDrainage() updates drainage parameters depending on the fraction of HRU meliorated, calculates and updates the parameters (DDRAIN, GDRAIN, and TDRAIN).

t. updateAtmosphericDeposition() updates atmospheric deposition parameters according to table (ATMDEP_BY_CATCHMENTS) with parameters (RAMMO_SUB, RCN_SUB, DRYDEP_NH4, DRYDEP_NO3).

u. updateBufferStrips() updates buffer strip parameter in catchments using the table (*BuferZone.Table_STAT_BY_CATCHMENTS*).

v. SetDefaultCoefficients() updates coefficient values specified in the settings.py files (defaultCoefficients).

w. updateRegionalCoefficients() applies the model coefficients by hydrological regions using the table specified in (*REGIONALCOEFFICIENT_TABLE*).

x. updateWUS() updates and applies data for water usage.

y. updateLup() fills land use update coefficients.

z. WriteFiles() writes information from the Watershed database into SWAT+ setup files by calling SWAT+ Editor API (api/actions/write_files.py).

Used script: sqliteSetup.py, functions fill_and_write_SWAT_Plus_setups
Input: *swatplus_datasets*
Output: ready-to-run system of SWAT+ setups.

4. Batch file for the parallelization of the model calculation (sequence of Watershed-by-watershed runs) is created to be allow the run the modeling system in correct order depending on the hierarchy level using the script *prepareRuns.py*.

4.Structure of electronic deliverables

4.1. Modelling system

The water quality modeling system is electronically delivered as a data folder (**MAINPATH)**. The water quality modeling system is generated automatically by executing the script ***main.py*** from folders **projectDir** */Scripts*.

4.2. River network and catchment data

The feature datasets of the prepared river network and catchment data are located under PostGre schema *catchmentdata*. Following data tables are included:

River network segment dataset for fine resolution – *segments*. The fields are:
id – the identifier of segment
riverto – the identifier of segment to which particular segment flows. Special values: -1 – external outlow, -4 – internal outflow.
skip_catchment. Whether this segment has corresponding catchment
kadastrid – river cadaster id, that corresponds to field *kadastrid* in supplied river cadaster dataset
kadastrid_lake – lake cadaster id, that corresponds to field *kadastrid* in supplied lake cadaster dataset
length – the length of segment in meters
geometry – the PostGIS geometry column, representing segment. All segments are of geometry type *ST_LineString*

Catchment polygon dataset for fine resolution – *catchments_fine*. The fields are:
id – identifier of the catchment
type – type of catchment. Possible values are:
1 – catcment having segment without the lake
2 – catcment having segment and the lake
3 – catchment having only standalone lake
4 – catchment without segment or lake
segmentid – identifier of segment for catchments of types 1 and 2, otherwise blank. If supplied is equal to *id*.
lakegid – identifier of lake for catchments of types 2 and 3, otherwise blank. It corresponds to attribute *gid* of lake cadaster dataset.
kadastrid – river cadaster id, that corresponds to field *kadastrid* in supplied river cadaster dataset
kadastrid_lake – lake cadaster id, that corresponds to field *kadastrid* in supplied lake cadaster dataset (*2020-08-17_plotiniai.shp*)
flowto – identifier of catchment to which outflow is directed. -1 for outlets
segmentto – corresponds to *riverto* of segment, for types 1 and 2m otherwise blank

outletid – identifier of outlet for catchments flowin into outlets, including water transfers
area – area of catchment in m^2, excluding area outside of the modelling area
addedarea – area of catchment that lies outside of the modelling area, except for catchments having inflow from big transboundary rivers
inflowarea - area of catchment that lies outside of the modelling area, for catchments having inflow from big transboundary rivers
geometry – the PostGIS geometry column, representing catchment. They have PostGIS geometry type *ST_Polygon* or *ST_MultiPolygon*

Outlet dataset – *outlets*. Fields are:
id – identifier of the outlet
outlettype – type of outlet. 3 – outlet is water transfer, other values – external outlet
outletname – name of the outlet

Water transfer dataset – *transfers*. Fields are:
id - identifier of transfer table entry
outletid – identifier of outlet
name - name of water transfer entry
flowto – catchment receiving outflow from this water transfer entry
multiplier – outflow to inflow ratio for this water transfer entry

Table of watersheds – *watersheds*. Fields are:
watershedid – identifier of watershed
basinname – name of watershed basin
watershedname – name of watershed
composite – composite

4.3. Land use

Following layers are transferred to PostGIS tables under the Postgre schema landuse to be used by the modelling system scripts:

landuse – contains landuse polygons. Fields are:
objectid – identifier of landuse polygon
lucode - generalized landuse code as described in point 4 of this chapter
shape_area – area of polygon in m^2
shape_length – perimeter of polygon in m
shape – PostGIS geometry columns
globallookup – table for connection of *landuse.lucode* with SWAT+ crop/urban codes. Fields are:
globalcode – generalized landuse code corresponding to field *lucode* of table *landuse*
swatcode – SWAT+ crop or urban code

The following additional electronic data is supplied:
lucode_statistics – summary areas of *union* polygons grouped by *LUCODE*

crops_statistics – summary areas of *crop* polygons
forest_statistics – summary areas of *forest* polygons
gdr_statistics – summary areas of *GDR* polygons.

4.4. Other input data

For soil data, atmospheric deposition, meteorological data, wateruse and point source data, ferilisation (incl livestock) data included in the electronic deliverables see the respective sections of the report PAIC (2020).

4.5. Versioning system

The modelling system is delivered via subversion repository system (SVN). Its structure is given in the Figure 4 below.

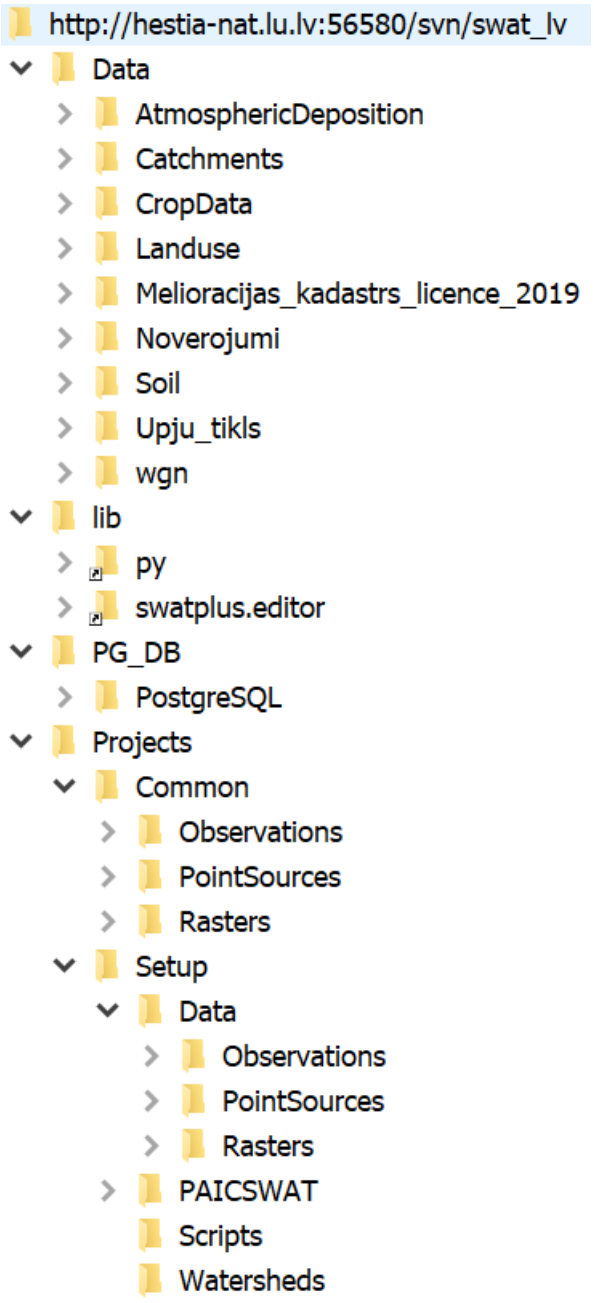


Figure 4: Structure of SVN repository.

Repository access:
Adress: http://hestia-nat.lu.lv:56580/svn/swat_lv
Read only mode is allowed for the partners.
Username: swat_lv
Password: swat_lv

Repository can be accesed by browsers, but recomended tool for the repository access is TortoiseSVN[7].

[7] https://tortoisesvn.net/downloads.html

References

Bieger, Katrin, Arnold, Jeffrey G., Rathjens, Hendrik, White, Michael J., Bosch, David D., Allen, Peter M., Volk, Martin, and Srinivasan, Raghavan, 2017. Introduction to SWAT+, a Completely Restructured Version of the Soil and Water Assessment Tool. Journal of the American Water Resources Association (JAWRA) 53(1): 115– 130. doi.org/10.1111/1752-1688.12482

PAIC, 2020. DATA BASE FOR MODEL IMPLEMENTATION. LIFE GoodWater IP deliverable R1, Rīga, 31p.