i-Tree

HydroPlus

Technical Manual

Updated: October 11, 2021

i-Tree is a cooperative initiative

About

Introduction

This document is intended to facilitate use and development of the HydroPlus environmental model suite – referred to as HydroPlus in reference to its source code – used as the backend for i-Tree Hydro, i-Tree Cool Air, and in the future potentially other models developed by the iTree-ESF collaborative. Some history is included to provide context and institutional knowledge.

Serving as an ad hoc FAQ for the researchers developing and using HydroPlus, this document is expected to evolve and expand over time to include basic information about code architecture and common development practices (e.g. how to add new functions and inputs); explanation and solutions to common problems that come up when working with these complex models; and different options or configurations possible for running this code.

As HydroPlus is part of the i-Tree Research Suite, i-Tree Tools' usual level of technical support and documentation is not available for HydroPlus. This document is intended for readers with a moderate level of expertise in environmental and computer science. Technical support is not available for free, but expert consultation can be arranged. If interested in consultation, reach out to the i-Tree Team at info@itreetools.org.

Disclaimer

The use of trade, firm, or corporation names in this publication is solely for the information and convenience of the reader. Such use does not constitute an official endorsement or approval by the U.S. Department of Agriculture or the Forest Service of any product or service to the exclusion of others that may be suitable. The software distributed under the label "i-Tree Suite" is provided without warranty of any kind. Its use is governed by the End User License Agreement (EULA) to which the user agrees before installation.

Feedback

The i-Tree Development Team actively seeks feedback on any component of the project: the software suite itself, the manuals, or the process of development, dissemination, support, and refinement. Please send comments through any of the means listed on the i-Tree support page: www.itreetools.org/support/.

Acknowledgements

Components of the i-Tree software suite have been developed over the last few decades by the U.S. Forest Service and numerous cooperators. Support for the development and release of the 2019 i-Tree software suite has come from USDA Forest Service Research, State and Private Forestry, and their cooperators through the i-Tree Cooperative Partnership of the Davey Tree Expert Company, Arbor Day Foundation, Society of Municipal Arborists, International Society of Arboriculture, Casey Trees, and State University of New York College of Environmental Science and Forestry.

The i-Tree Cool Air model was originally developed by Drs. Yang Yang, SUNY College of Environmental Science and Forestry (SUNY-ESF), Ted Endreny (SUNY-ESF), and David J. Nowak, USDA Forest Service, Northern Research Station (USFS-NRS). The model code has been improved and integrated within i-Tree based on the work of Thomas Taggart (SUNY-ESF), Shannon Conley (Davey Institute), Robert Coville (Davey Institute), Vamsi Kodali (Syracuse University), Sneha Patil (Syracuse University), Arpit Shah (Syracuse University), and Reza Abdi (SUNY-ESF). i-Tree Cool Air uses i-Tree Hydro as its underlying hydrology model.

The i-Tree Hydro model was originally developed by Drs. Jun Wang SUNY College of Environmental Science and Forestry (SUNY-ESF), Ted Endreny (SUNY-ESF), and David J. Nowak, USDA Forest Service, Northern Research Station (USFS-NRS). The model code has been improved and integrated within i-Tree based on the work of Megan Kerr (Davey Institute), Yang Yang (SUNY-ESF), Sanyam Chaudhary (Syracuse University), Rahul Kumbhar (Syracuse University), Yu Chen (Syracuse University), Thomas Taggart (SUNY-ESF), Shannon Conley (SUNY-ESF), Pallavi Iyengar (Syracuse University), Jeevitha Royapathi (Syracuse University), Robert Coville (Davey Institute), Isira Samarasekera (Syracuse University), Vamsi Kodali (Syracuse University), Sunit Vijayvargiya (Syracuse University), Sneha Patil (Syracuse University), Arpit Shah (Syracuse University), Akshay (Syracuse University), and Reza Abdi (SUNY-ESF). Topographic Index calculations have been improved with algorithms developed for WhiteBox GAT (Lindsay, 2016) with permission by its creator John Lindsay, PhD (University of Guelph).

Many other individuals have contributed to the design, development, testing process, and help text including Scott Maco (Davey Institute), Mike Binkley (Davey Institute), David Ellingsworth (Davey Institute), Dr. Satoshi Hirayabashi (Davey Institute), Dr. Jim Fawcett (Syracuse University), Emily Stephan (SUNY-ESF), Evan Williams (SUNY-ESF), Ryan Morrison (SUNY-ESF), James Kruegler (SUNY-ESF), and Jeremy Hatfield (SUNY-ESF). This Technical Manual was written & designed by Robert Coville (Davey Institute) based on the i-Tree Hydro v6 User Manual and software development notes from Shannon Conley (Davey Institute), Ted Endreny (SUNY-ESF), and Reza Abdi (SUNY-ESF).

Table of Contents

About 3

Introduction 3

Disclaimer 3

Feedback 3

Acknowledgements 4

Table of Contents 5

Getting Started 7

Green Infrastructure 7

A Basic Workflow for Running GI 7

Inputs & Running HydroPlus Models 8

Introduction to Using HydroPlus 8

Required Input Files 8

Config File (HydroPlusConfig.xml) 9

Config File settings applicable to all models 9

Config File settings for Hydro 9

Config File settings for Cool Air 14

Weather data 15

Auto-calibration from a Command Line Interface (CLI) 15

Autocalibration setup files 16

Autocalibration, fresh start, summary 16

Autocalibration, fresh start, in detail 16

Autocalibration, retry with new observations 21

Outputs 21

Hydrology model outputs 21

Green Infrastructure outputs 22

Vertical (Depth) and Final (Volumetric) Water Budgets 22

Hydro Extended Output Header Definitions 22

Temperature model outputs 24

Cell-based Output 24

Time-based Output 25

Tips & Troubleshooting **28**
Local work environment 28
Version Control and Testing **28**
Version Control 28
Locations 28
SVN Concepts and Definitions 29
TortoiseSVN Subversion Control Procedures 29
Git Subversion Control Procedures 30
SOP for Testing Code 31
Procedure for Running Test Script 31
SOP for Developing New Features 32
Development Notes **33**
Visual Studio setup for developing & testing code 33
Config file modifications 34
Batch superficial changes (file only, no code changes) 34
Code changes associated with config file changes 37
Running HydroPlus in sub-hourly timesteps (Jan 21, 2019) 37
Land cover scaling simulations (LCscaling utility) 38
Hydro LCscaling analysis 38
Cool Air LCscaling analysis 39
Conditions for TC+IC != 100% 39
Functors 39
Build settings 40
Feedback on C++ methods for GI dev (Oct 18, 2018) 41
Calculation Classes 41
The DataFolder class 42
Development method and troubleshooting 44
Architecture development Q&A for GI (Oct 26, 2018) 44

Getting Started

As a first step to using HydroPlus for your own projects, we recommend using HydroPlus Test Cases to begin familiarizing yourself with the available models and settings. Test Cases are maintained as part of HydroPlus development, and these Test Cases offer a comprehensive look at what each model requires as inputs and offers as outputs.

To view available Test Cases, open Windows Explorer and navigate to the following directory:
`…\HydroPlus\TestingFilesAndScript\TestCases\`

There you'll find testing inputs and outputs organized in the following folder hierarchy:

- HydroPlus model
    - Input/output configurations available for the model
        - Functional input set
        - Outputs from test simulation
        - Expected output serving as a standard to compare tests against

For a list of what Test Cases are available are how they are used for testing, please explore this manual's section on our SOP for Testing Code. To try running a single Test Case, which is the recommended starting point for new HydroPlus users, please choose a Test Case of interest from the SOP for Testing Code then proceed through the Inputs & Running HydroPlus Models section.

Green Infrastructure

Green infrastructure features are available in the latest StatisticalHydro model mode, designed based on the EPA SWMM model's LID modules. As described above, we recommend starting with TestCases to ensure you can run those validated scenarios before venturing into your own project creation.

Begin with powIR_defaultParams_noTI test case for best comparability with Hydro GUI input/output. Then explore the GI test cases, there is one available for each GI structure type.

A Basic Workflow for Running GI

1. Open master config file(s) from Test Cases for GI of interest
2. Copy dataFolder(s) for GI of interest into your own project config file
3. Adjust parameters in each GI dataFolder. When unsure what value to give a parameter, you can refer to parameter values from your project's BulkArea dataFolder or from the GI's TestCase parameters. For more guidance on GI parameterization, see Config File settings for Hydro > Green Infrastructure.
4. Save config file & run HydroPlus.exe same as without GI (Inputs & Running HydroPlus Models).
5. Access outputs as you would without GI (Outputs).

Inputs & Running HydroPlus Models

Introduction to Using HydroPlus

To run any of the models within HydroPlus, the HydroPlus executable file must be run along with a command line argument giving the path to the input files without any trailing slash.

For example, the following could be run in a Windows command line interface:

```
C:\HydroPlus\x64\Release\HydroPlus.exe C:\HydroPlus\Inputs
```

The program will look in the inputs directory for a configuration file "HydroPlusConfig.xml". The config file determines what model is run and with what settings and parameters. Different input files are needed depending on which model is run. HydroPlus includes the following models:

- StatisticalHydro: the hydrology model used by i-Tree Hydro, operating in a semi-distributed aka statistically-distributed framework.
- GI: a version of StatisticalHydro which includes green infrastructure features based on the EPA SWMM model and the work of Endreny & Abdi in 2019.
- SpatialTemperatureHydro: the air temperature model referred to as i-Tree Cool Air or originally PASATH, operating in a raster-based spatially-distributed framework. This model uses StatisticalHydro's hydrology routines where possible.

More information about the Config File is available later in this section.

Required Input Files

StatisticalHydro

The following input files are expected in the input directory for the Hydro model to run:

- HydroPlusConfig.xml
- dem.dat
    - OR powdecayTI.dat when config file includes <Infiltration>PowerDecay</Infiltration>
        - PowerDecay intended for use with a DEM or a powdecayTI file from HydroPlus; if a DEM is used, Hydro will subsequently produce a powdecayTI.dat file
    - OR expdecayTI.dat when config includes <Infiltration>ExponentialDecay</Infiltration>
        - ExponentialDecay intended for use with the TI files preloaded from Hydro GUI
- weather.dat & Evaporation.dat (Pre-processed files; raw from NOAA NCDC, preprocessed using the Hydro GUI. See Hydro User Manual for more information on weather data.)
- Pollutiondata.dat (Model can run without this but no water quality results will be written)
- Qobs.dat (Only needed when config file does not include <CalibrationTimeStep>NoCalibration</CalibrationTimeStep>)

See Hydro v6 User Manual about StatisticalHydro model inputs, and Abdi 2019 about GI inputs.

SpatialTemperatureHydro

The following input files are expected in the input directory for the Cool Air model to run:

- HydroPlusConfig.xml
- dem.dat
- Weather.dat
- Evaporation.dat (used only for Snow evaporation potentials)
- SolarRadiation.dat
- Blockgroupmap.dat (optional)
- treecover.dat
- imperviousCover.dat
- landcover.dat

Config File (HydroPlusConfig.xml)

Config File settings applicable to all models

OutputDirectory is the full path to the directory which output files will be written to, ending with \

Note that commenting out tags is ineffective: if a tag is written in the file that the code looks for, it will activate and grab values for that tag. Instead, to disable tags, the tags of interest must be wrapped in a <DISABLE></DISABLE> set of tags.

Example Config File with Comments Explaining Parameters - August 23, 2019

The HydroPlusConfig-commented.xml file in /TestingFilesAndScript/TestCases/StatHydro/ is for the semi-spatially distributed Hydro model. It includes in-line comments to clarify XML configuration file options. Parameter names listed here correspond with parameters described in the i-Tree Hydro v6 User Manual's Table of Hydrological Parameters.

For more information about HydroPlus Test Cases and the background for this Test Case see the SOP for Testing Code section.

File Version Info: This example config file has the semi-distributed Hydro model running in power decay infiltration rate mode, with extended outputs disabled and using i-Tree Hydro v6.3 GUI default hydrological parameters. This version is the latest as of August 23, 2019 from code repository revision 427 path …\HydroPlus\TestingFilesAndScript\TestCases\StatHydro\powIR_defaultParams\input\HydroPlusConfig.xml

Config File settings for Hydro

All parameters within the <DataDrawer> section of the config file are used for semi-distributed Hydro simulations. The i-Tree Hydro v6 User Manual has more information about how to inform parameterization of the model. See the Table of Hydrological Parameters for suggested defaults and ranges for parameters. The Example Config File with Comments Explaining Parameters - August 23, 2019 associates config file parameter tags with the parameter names used in the i-Tree Hydro v6 User Manual.

Green Infrastructure

Refer to Getting Started with Green Infrastructure for basic instruction on interacting with GI features. In this section, more technical details are provided as orientation for GI's unique inputs in HydroPlus.

In a config file, GI features can be enabled by adding DataFolders with Type set to a GI structure type. GI can be parameterized as aggregate or individual representations of each GI Type:

- Aggregate representation of GI: Each DataFolder can describe the aggregate & average statistics representing all GI structures of a certain type. This 'lumped' or 'semi-distributed' representation is the approach used for the 'bulk area' of projects run in the Hydro GUI and the StatisticalHydro mode of HydroPlus.
- Individual representation of GI: Each DataFolder can represent an individual GI structure, with multiple DataFolders to represent multiple instances of GI within a DataDrawer for all GI of the same type.

GI model schematics

The following figures about GI modeling in HydroPlus are provided below, from HydroPlus GI developers at SUNY-ESF:

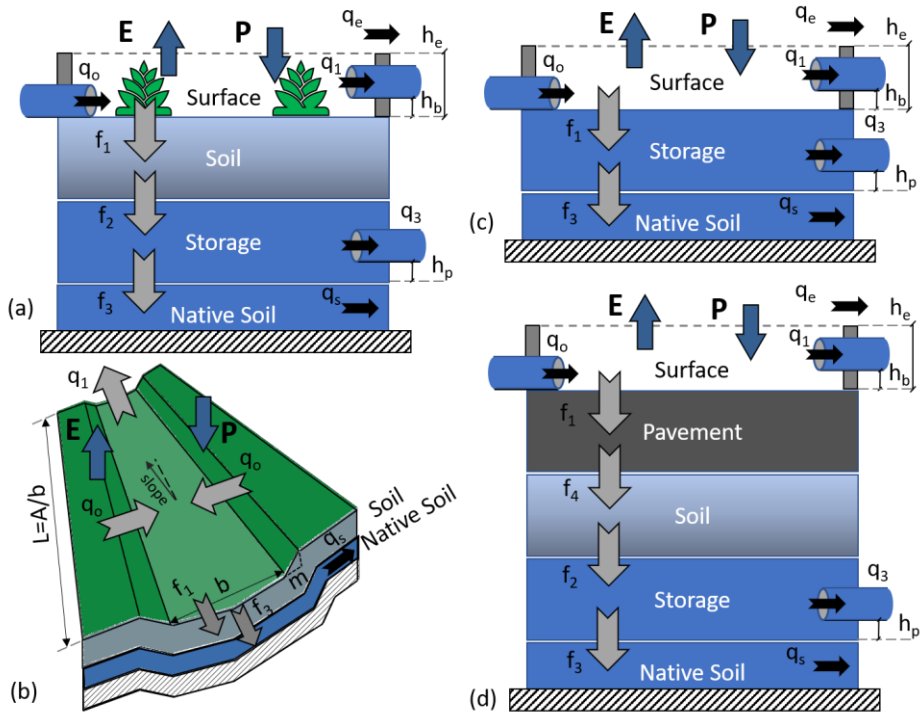


Fig. 1. The schematic design of the bioretention cell (a), vegetation swale (b), infiltration trench (c), and permeable pavement (d). In the figure, the term *q* represents the water flow to and from the GI design with the substrates of $_0$ for the inflow, $_1$ for the surface outflow, $_3$ for the underdrain, $_e$ for the emergency spillway outflow, and $_s$ for the base flow. The term *f* represents the movement of water in the vertical direction with the substrates of $_1$ for the infiltration, $_2$ for the percolation, $_3$ for exfiltration, and $_4$ for percolation through the pavement. The h_b is the outflow pipe height from the surface, h_e is the emergency spillway outflow height, and h_p is the drain offset height. In panel (b), *L* is the length of the swale, *A* is the swale area, *b* is the bottom width, *P* is the rainfall, *E* is the evaporation, and *m* is the side slope.

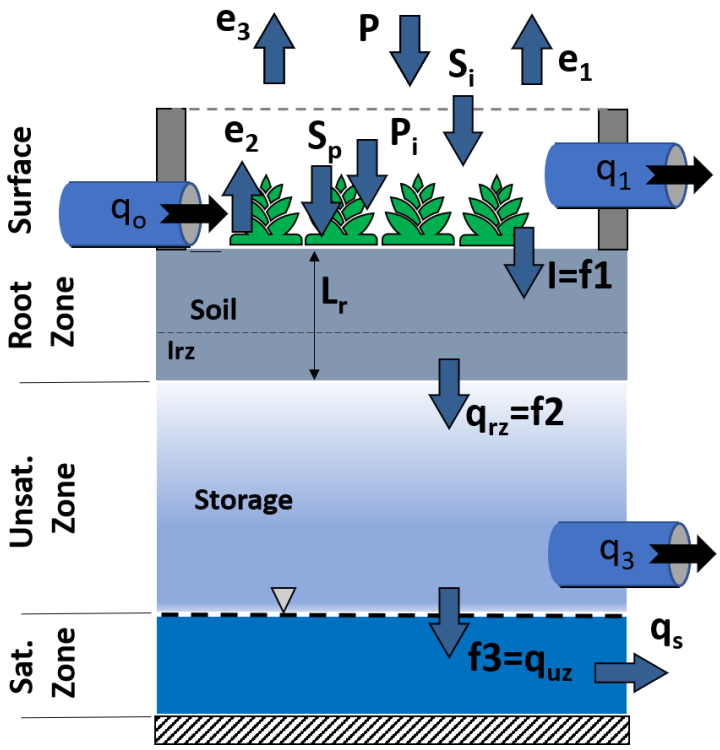


Fig 2. The schematic of the flow variables in i-Tree Hydro and SWMM overlaid with the i-Tree Hydro model structure. In the figure P is the precipitation, P_i is the canopy interception, S_p is the pervious depression storage, S_i is the impervious depression storage e_3 is the vegetation evaporation, q_s is the base flow, I is the infiltration, L_r is the maximum root zone depth, I_{rz} is the root zone storage, q_{rz} is the root zone to unsaturated zone percolation, and q_{uz} is the unsaturated zone to saturated zone percolation. The terms f_1, f_2, and f_3 referred the infiltration, percolation, and exfiltration respectively in SWMM.

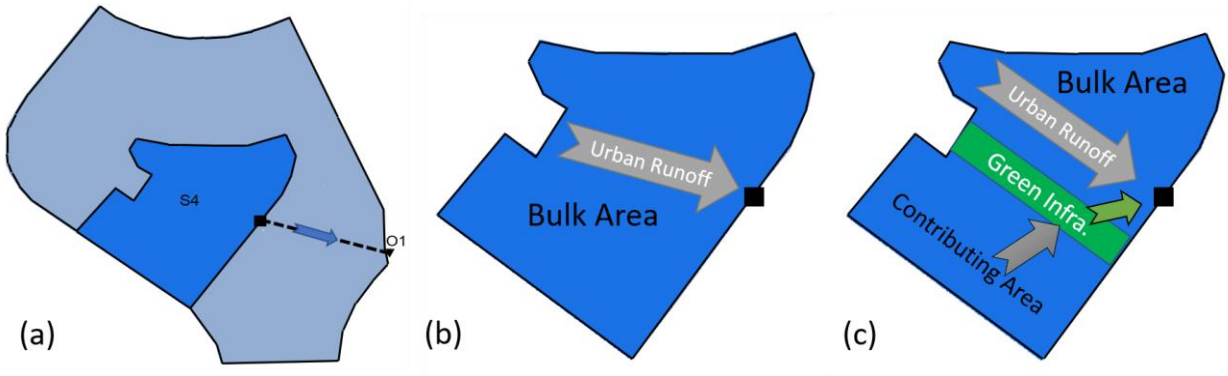
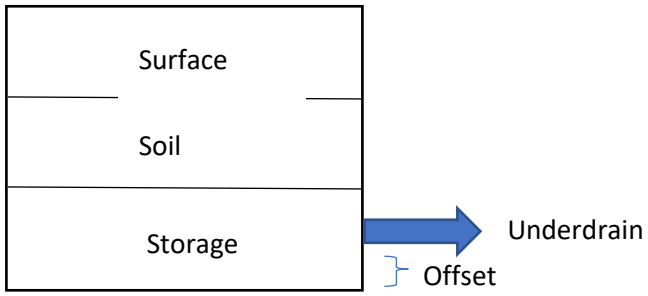


Fig. 3. The sub-catchment S4 (a) has been used in the simulations without applying the GI devices (b) and after adding the GI devices (c).

GI parameter descriptions

Notes about unique GI parameters and relevant figures are provided below, from HydroPlus GI developers at SUNY-ESF:

1. **<soilCondSlope>10.0</soilCondSlope>** <!-- soil conductivity slope: Slope of the curve of log(conductivity) versus soil moisture content (dimensionless). Typical values range from 5 for sands to 15 for silty clay. -->
2. **<drainOffset>0.05</drainOffset>**: from the bottom (to trap sediment).



E.g. The height and volume of the rain barrel/cistern would be determined by commercially available sizes. The drain offset is typically 6 inches from the bottom (to trap sediment).

Once the water height in the storage layer reaches the drain's offset height, any inflow from percolation out of the soil layer will immediately flow out of the underdrain as long as its flow rate is below 15 in/hr (as per Equation 6-8) and the storage volume above the offset height will never be used.

3. **drainOrificeQ_coef and drainOrificeQ_exponent:**

drainOrificeQ_coef is drain.coeff in SWMM code, *C3D* in SWMM manual.

$$C_{3D} = N_{pipe} Q_{full} / A_{LID} \qquad (6\text{-}71)$$

where N_{pipe} is the number of drain pipes in the unit and A_{LID} is the area (ft^2) of the unit.

drainOrificeQ_exponent is drain.expon in SWMM code, η_{3D} in SWMM manual.

```
outflow = folder->ParamDict["drainOrificeQ_coef"] * pow(head,
folder->ParamDict["drainOrificeQ_exponent"])
```

Because the hydraulics of perforated pipe underdrains can be complicated (see van Schilfgaarde 1974) SWMM uses a simple empirical power law to model underdrain outflow q_3:

$$q_3 = C_{3D}(h_3)^{\eta_{3D}} \qquad (6\text{-}9)$$

where

h_3 = hydraulic head seen by underdrain, (ft)
C_{3D} = underdrain discharge coefficient ($ft^{-(\eta_{3D}-1)}/sec$)
η_{3D} = underdrain discharge exponent

4. **<initiallySat_frac>0.0</initiallySat_frac>**

```
folder->ParamDict["initiallySat_frac"] = (1.0 -
(folder->ParamDict["SD0_percent"] / 100.0));
```

SD0_percent: initial upper soil zone saturation percent

Not defined in config file, but in code. May need to be removed.

5. **<stor<u>ClogFactor</u>>0.0</storClogFactor>**

```
storage.clogFactor in SWMM
```

To get the exfiltration (called infiltration in SWMM, Exfiltration in UH)

```
// infiltration rate = storage storage conductivity (seepage rate) reduced by
any clogging
      exfil = folder->ParamDict["seepageRate_mph"] * (1.0 - clogFactor);
```

For more guidance on GI parameterization: explore commented example GI config file (\HydroPlus\TestingFilesAndScript\TestCases\StatHydro\GI\HydroPlusConfig_GI-commented.xml); review Abdi 2019 (section 5.2 describes GI methods, Table 13 & Table 17 list GI TestCases' parameters); SWMM LID documentation; and the Hydro v6 User Manual.

Simulating Multiple Hydrologic Structures

Each <DataFolder> represents a hydrologic structure within the project, and most projects are represented entirely by the BulkArea parameters of a single DataFolder. When simulating Green Infrastructure), multiple DataDrawer or DataFolder tags may be setup to represent different GI structure types and individual GI structures.

Config File settings for Cool Air

The temperature model gathers hydrological parameters from the first DataFolder in the config file. That DataFolder generally represents the Bulk Area, which is the entire project area for most Hydro projects. For parameters included in Cool Air-specific inputs, such as treecover.txt, the Cool Air-specific input option is used instead of inputs from the DataFolder.

TemperatureLocationParams orient the model about the raster inputs. The inputs in the config file should match what would be in the header of raster input files. The required TemperatureLocationParams are nCols, nRows, cellsize, xllcorner, yllcorner. If NODATA_value is defined for raster inputs, it should also be defined with a NODATA tag in this config file parameter group.

TemperatureExecutionParams is the parameter group in the config file that defines the weather station and what outputs are written. RefWeatherLocation is an important input for the model. The conditions associated with the RefWeatherLocation in the gridded spatial inputs should match the conditions at the

observed weather data source. In most cases weather inputs come from airport weather stations with an NLCD LC=21, Impervious Cover<30%, and Tree Cover<5%.

Config file tags for Cool Air output options are listed below. Any row in the example below can be included or left out from your config file, except RefWeatherLocation. Copy config file rows from here and add them to your file to add an output option. For more information about what values are available for these different output options, see the Temperature model outputs section.

Example of all options for TemperatureExecutionParams

```
<TemperatureExecutionParams>
	<RefWeatherLocation>1,42</RefWeatherLocation> <!-- row,col starting at 0 -->
	<PrintBlockGroupRange>All</PrintBlockGroupRange> <!-- All or delete/wrap in
DISABLE tag -->
	<PrintBlockGroupRange0>1,5</PrintBlockGroupRange0> <!-- start,end of range;
additional output ranges can be set, incrementing up # at end of tag name each time -->
	<PrintRowCol>All</PrintRowCol> <!-- All or delete/wrap in DISABLE tag -->
	<PrintRowCol0>row,col</PrintRowCol0> <!-- additional row,cols to print can be set,
incrementing up # at end of tag name each time -->
	<RowColOutputPeriod>All</RowColOutputPeriod> <!-- All or delete/wrap in DISABLE
tag -->
	<RowColOutputPeriod0>YYYYMMDDHH,YYYYMMDDHH</RowColOutputPeriod0> <!-- additional
output ranges can be set, incrementing up # at end of tag name each time -->
	<PrintTimeBasedResults>All</PrintTimeBasedResults> <!-- All or any Time-Based
Variables listed in UH Dev Manual separated by comma -->
	<TimeBasedOutputPeriod>All</TimeBasedOutputPeriod> <!-- All or delete/wrap in
DISABLE tag -->
	<TimeBasedOutputPeriod0>YYYYMMDDHH,YYYYMMDDHH</TimeBasedOutputPeriod0> <!--
additional output ranges can be set, incrementing up # at end of tag name each time -->
</TemperatureExecutionParams>
```

Weather data

Weather data acquisition and preprocessing has changed as of Spring 2020 to accommodate changes from our weather data provider, U.S. NOAA. The latest procedure works for i-Tree Hydro and Cool Air models. It generates three files, two of which are required by Hydro and all three of which are required by Cool Air. That procedure is documented in an i-Tree Support forum FAQ on Weather Data Inputs, under the “Using a raw weather file” section: https://forums.itreetools.org/viewtopic.php?f=35&t=1248

Auto-calibration from a Command Line Interface (CLI)

The original model in HydroPlus, referred to as UFORE-Hydro (Wang et al. 2008), was autocalibrated using the PEST utility (https://pesthomepage.org). That functionality was adapted for the i-Tree Hydro Graphical User Interface (GUI). With HydroPlus, Hydro and other models can be autocalibrated using PEST from a CLI. Autocalibrated from a CLI may not be as convenient as GUI-based autocalibration for

some users, but this approach offers greater transparency, easier code maintenance, greater flexibility, and greater capacity for users to troubleshoot their own projects, which are qualities prioritized for advanced models in the i-Tree Research Suite.

Use the following steps to autocalibrate a Hydro project from a CLI. These instructions were adapted from the PEST Manual specifically for the HydroPlus StatisticalHydro model. These instructions were tested using Windows 10 Command Prompt, a text editor, and select PEST Suite 17.1 executable files that are version controlled for HydroPlus at https://code.itreetools.org/ESF/Hydro/Autocalibration and are available at https://www.itreetools.org/documents/647/HydroPlusAutocalibrationUtilities.zip. Until autocalibration is built into HydroPlus functionality, the following procedure enables autocalibration without the Hydro GUI.

Autocalibration setup files

These are the files you will need for running autocalibration, prepared through the following steps:

- HydroPlusConfig.TPL
- HydroPlusConfig.PAR
- HydroPlusConfig.PST
- Output.INS
- Output_Obs.OBF

Autocalibration, fresh start, summary

Here is a summary of the current steps to autocalibrate, detailed in the next section:

0. Setup a working directory
1. Run a Hydro simulation as a starting point
2. Create a TPL file
3. Create a PAR file
4. Run Tempchek
5. Create an INS file
6. Run Inschek
7. Update the OBF file
8. Run Pestgen to create a PST file
9. Update the PST file
10. Run Pestchek
11. Run Pest
12. Assess autocalibration results

Autocalibration, fresh start, in detail

0. Setup working directory with Hydro inputs, where you want autocalibration to run and generate outputs. Start a command prompt, with access to HydroPlus.exe and the PEST Suite EXE files from https://www.itreetools.org/documents/647/HydroPlusAutocalibrationUtilities.zip.
1. Run a Hydro simulation using parameters for autocalibration to start from.

a. Autocalibration needs to use output files generated using the appropriate calibration timestep. Calibration timestep means what interval the predicted and observed data sets should be compared in and optimized for.
   By default we recommend using a weekly calibration timestep, which is defined in the config file as <CalibrationTimeStep>Weekly</CalibrationTimeStep>. As an alternative, we recommend trying Monthly. Daily, or especially Hourly, calibration timesteps will extend time required for autocalibration and can lead to over-fitting parameters to finer-resolution fluctuations in observed flow.
   The <CalibrationTimeStep> parameter needs to be defined in the config file to begin with and the same interval must be used consistently throughout each calibration process.

2. Create a .TPL file:
   a. Copy the config file and rename with the .TPL extension.
   b. Add the pest header "ptf #" as the first row in that new TPL file.
   c. For each parameter you want to autocalibrate, replace the parameter value with tags (in the format of #paramname#) for PEST to find and adjust. For example, to autocalibrate the scale parameter of soil transmissivity known as m, replace <m>0.023</m> with <m>#m#</m> in the .TPL file. Each #paramname# tag can be any string 13 characters or less, surrounded by # on either side. At least 3 characters should be in the #paramname# string, as in #m #, to ensure the .TPL and other files of later steps will write a sufficient amount of significant figures in autocalibration-generated config files (as test-generated in step 4 below).
      The Hydro v6 GUI autocalibrated the following parameters:
      i. MSD: Max Depth of Water in Upper Soil Zone (m), aka Maximum root zone storage deficit
      ii. m: scale parameter of soil transmissivity
      iii. T0: Soil Transmissivity at Saturation (m^2/hr)
      iv. PMacro: Soil Macropore fraction
      We recommend these are used for autocalibration by inserting each respective PEST tag for each parameter's value in the .TPL file:

```
#MSD  #
#m    #
#T0   #
#PMacro#
```

3. Create a .PAR file: From a command line, run tempchek.exe on the TPL file (as in command "tempchek HydroPlusConfig.tpl"). That generates a .PMT file if errors are not found. The newly-generated .PMT file needs to be renamed to a .PAR extension (the .PMT version will not be used further for HydroPlus autocalibration). That newly converted .PAR file needs to be edited, as shown below and in excerpt on the right from the PEST Manual figures:
   a. Initial .pmt file contents: *HydroPlusConfig.pmt*

t0
m
msd
pmacro

b. End-of-step contents of that .PMT file, renamed to be a .par file: *HydroPlusConfig.PAR*

single point
t0 0.13 1.0 0.0
m 0.023 1.0 0.0
msd 0.05 1.0 0.0
pmacro 0.1 1.0 0.0

c. Format of generic .PAR file:

<header line: "single point">
<#paramname# from .tpl> <default value> <weighting values "1.0 0.0">

4. Check that .PAR file is working by running tempchek <TPL file> <config file> <PAR file>, which will write a config file using the values given in the PAR file. Confirm the output config is good.

At this point, PEST autocalibration is ready on the input-side of the model. Now PEST needs to be configured for the output-side of autocalibration. This section begins at PEST Manual 18.1.4.

```
s1
s2
y1
xc
```

Figure 18.7 File *in.pmt*.

```
single point
 s1   0.3   1.0 0.0
 s2   0.8   1.0 0.0
 y1   0.4   1.0 0.0
 xc   0.3   1.0 0.0
```

Figure 18.8 File *in.par*.

5. Create a new 'instruction' file with the calibration timestep-based output file's name and INS filetype, as in CaliTotalQQ.INS. (CaliTotalQQ being output based on CalibrationTimeStep setting.) The .INS file guides PEST on parsing model outputs for comparison with observations. The .INS file needs a "pif #" header line, and then a line for each observation with the following format:

   <lower-case L><# of linebreaks down to next row of observations><space><observation name><column of observation start><column of observation end>

   <u>Example (using an observation timeseries starting on line 2 of CaliTotalQQ.dat):</u>

   pif #
   l2 [totalflow_1]40:54
   l1 [totalflow_2]40:54
   l1 [totalflow_3]40:54
   l1 [totalflow_4]40:54
   l1 [totalflow_5]40:54
   …

   Here is a sample Output.INS for hourly calibration for a simulation from 01/01/2015 1st hour to 12/30/2015 24rd hour (not a leap year): https://www.itreetools.org/documents/646/Output.ins

Here is a sample CaliTotalQQ.INS for weekly calibration for a simulation from 01/01/2010 1st hour to 12/29/2010 24th hour (not a leap year):
https://www.itreetools.org/documents/671/CaliTotalQQ.ins

6. Run *INSCHEK <INS file> <Output file>* (as in "inschek CaliTotalQQ.ins CaliTotalQQ.dat") to confirm instruction file works well with the output file and to generate the .OBF file.
7. The .OBF file generated in the previous step was generated using predicted values, but it needs to use observed values instead.
    a. To begin this step, rename the OBF file to reflect it using observed values (e.g. rename to "CaliQobs" or add a "_obs" suffix), to ensure modifications to the OBF are not overwritten by running inschek. Hydro differentiates between CalibrationTimeStep-based files for observations vs predictions, in CaliQobs.dat vs CaliTotalQQ respectively. For this reason, although CaliTotalQQ.INS generated the OBF file, we recommend the OBF file be renamed CaliQobs.OBF and filled using data from CaliQobs.dat.
    b. Next, paste in observations for each row, including a space delimitation between the observation value and the observation ID. For a short amount of observations, the application Notepad++ has an option to use ALT+Click to select columns of data for copy and pasting. For longer amounts of observations, the .OBF file can be imported into a spreadsheet with its data separated into columns, the observation records can be updated using a spreadsheet timeseries of discharge records at the appropriate timestep, and the resulting .OBF data can be copy and pasted from the spreadsheet into a text file.
    c. Note that the .OBF file is space-delimited and can work with only a single space between observation name and observation data.
8. Create PST file: Run *PESTGEN <name PST file to be generated> <name of existing PAR file> <name of existing OBF file>* (as in "pestgen HydroPlusConfig.pst HydroPlusConfig.par CaliQobs.obf") to generate the PST file with the proper parameters and observations.
9. This PST file may need some modification:
    a. Parameter ranges for autocalibration should be updated, and PEST settings for 'singular value decomposition (SVD)' should be added. You can do so by copying the italic text below into your PST file, replacing the original content between the "* parameter groups" and "* observation groups" lines. The following adds SVD features, and assigns the upper and lower bounds of parameters, following the Hydro v6 GUI autocalibration settings. This example also changes parameter data settings from "none relative" to "log factor" as used by the Hydro v6 GUI.

** singular value decomposition*
1
4 5e-7
0
** parameter groups*
t0 relative 0.01 0.0 switch 2.0 parabolic
m relative 0.01 0.0 switch 2.0 parabolic

msd relative 0.01 0.0 switch 2.0 parabolic
pmacro relative 0.01 0.0 switch 2.0 parabolic
** parameter data*
m log factor 0.023 0.01 1.2 m 1.000 0.000 1
T0 log factor 0.13 0.0005 150 t0 1.000 0.000 1
MSD log factor 0.05 0.001 1.0 msd 1.000 0.000 1
PMacro log factor 0.1 0.0000001 0.2 PMacro 1.000 0.000 1
** observation groups*

The following is a legend of variables in the "* parameter data" group of the .PST file. This is from the hydrology.pst-Example.txt written by Hydro v6 GUI autocalibrations.

* parameter data
# parameter name - PARTRANS - PARCHGLIM - PARVALI - PARLBND - PARUBND - PARGP - SCALE - OFFSET - DERCOM
-
PARTRANS - is a character variable which must assume one of four values, viz. "none", "log", "fixed" or "tied".
PARCHGLIM - is the param relative limited or factor limited?
PARVALI - parameter's initial value
PARLBND - parameter lower bound
PARUBDN - parameter upper bound
PARGP - name of parameter groups
SCALE - adjusts parameter before writing to input file
OFFSET - adjusts parameter before writing to input file
DERCOM - set to 1 unless using external derivatives functionality

b. Information below "*model command line" should be updated. The following is an example that applies if PEST is executed from the same directory containing HydroPlus.exe and the input config, .TPL, and .INS files, and when that config file and .TPL file is set to write output to a .\output\ folder in the same directory.

** model command line*
HydroPlus.exe .
** model input/output*
HydroPlusConfig.tpl HydroPlusConfig.xml
.\CaliTotalQQ.ins .\output\CaliQobs.dat
** prior information*

10. Before running PEST, the PEST Manual suggests checking inputs with PESTCHEK <PST file name without extension> (as in "pestchek HydroPlusConfig"). Notices about SVD mode can be safely ignored, though further modifying PEST settings may produce better autocalibrations.
11. Run PEST <PST file name without extension> (as in "pest HydroPlusConfig")
    a. Note that the .PST file can define # of iterations it will work through.
    b. PEST generates various files for output. The .REC and .SEN outputs can be especially useful for information about parameter sensitivity and calibration.

12. After PEST autocalibration is complete, modify your config file to have a CalibrationTimeStep other than NoCalibration if you have not already, then rerun HydroPlus.exe to get an Output.dat that shows goodness-of-fit (CRF) values comparing simulation outputs using autocalibrated parameters versus observed flow.

Autocalibration, retry with new observations

If autocalibration is not achieving desired goodness-of-fit for a project, consider trying a new weather year and observation set before trying an entirely new area of interest. For such 'retries', here are short steps from the above procedure:

- Copy the existing .TPL and .XML file, update as needed for new try (i.e. new simulation period).
- Repeat step 3b from above: grab the existing .PAR file and fill it in with the desired initial values (the file will have been changed by previous autocalibration attempts).
- (Optional) Repeat step 4 to confirm the TPL, PAR, and config XML files work well together.
- Copy Output.ins from the previous autocalibration, unless your simulation period has changed (as in a leap year vs. non-leap year), in which case you must repeat step 5 from above.
- Repeat steps 6-7 using your latest observations to get a .OBF file reflecting your new observation set.
- Repeat steps 8-9 to get a .PST file reflecting your new observation set and ready for use.
- Perform steps 10-12 to autocalibrate with your new observation set.

Outputs

After you've run the simulations you are interested in, find outputs in the OutputDirectory (defined in HydroPlusConfig.xml) for the simulation of interest.

Hydrology model outputs

The following is a list highlighting and explaining some Hydro outputs and extended outputs (enabled in HydroPlusConfig.xml) of interest:

output.dat – A summary of key outputs, many of which are reported within i-Tree Hydro GUI for earlier versions of the HydroPlus model. These values are presented as vertical water fluxes, depths in total millimeters or meters per timestep. More information about vertical, one-dimensional results and final, three-dimensional results available in the Vertical and Final Water Budgets section, including how values in Output.dat should be converted for volumetric results.

VegCanopyFile.csv – Features of the water balance for vegetation processes.

WaterOnPerArea.csv – Features of the water balance for pervious area processes.

WaterOnImpArea.csv – Features of the water balance for impervious area processes.

Evap_ET_Infil.csv – Outputs related to evaporation from vegetation surfaces, infiltration, and evapotranspiration from the root zone.

Green Infrastructure outputs

Each GI structure simulated generates a WaterBalance_<StructureType>.csv file, which including many of the same output variables used by the EPA SWMM model's LID modules. This file is like the Vertical Water Budget Extended Outputs described below. Units are in meters representing depths. Variables in GI WaterBalance output files are described in Abdi 2019; output file headers are associated with labeling diagramed in Figure 30 & Figure 31. Other outputs generated in a GI run represent project's non-GI 'bulk area', as depicted in Abdi 2019 Figure 32.

Vertical (Depth) and Final (Volumetric) Water Budgets

Each of the extended output .csv files comes in two versions marked with either a *_VW or _FW* suffix. The 'VW' versions represent the one-dimensional Vertical Water Balance or Flux results, with outputs computed as depths to be statistically-distributed throughout the project area. The 'FW' versions represent the statistically-distributed 'Final Water' Balance or Flux results, with VWB depths being multiplied by applicable land cover percentages and the applicable project area to produce volumetric results. For example, *Total tree intercepted rain (m^3) = Total tree intercepted rain (m) * (project area (m^2) * % tree canopy)*.

Output.dat and other results are presented in vertical water fluxes as well. In Output.dat, streamflow components TotalQ, BaseQ, PerviousQ, and DCIAQ are proportional to the project area as a whole, and accordingly they can be multiplied by the project area to convert to volumetric results. Other totals in the Output.dat file such as precipitation and interception values are relative to their applicable land cover type, as in the example with *Total tree intercepted rain* in the above paragraph.

Hydro Extended Output Header Definitions

Acronyms and phrases used in column titles of Hydro extended output files are defined below:

General Definitions

W - water
WB - water balance
WF - water flux
SWE - snow water equivalent
SWEB - snow water equivalent balance
Comb - combined water balance from water and snow water equivalent
OIA - open impervious area
TCIA - tree cover over impervious area
TCPA - tree cover over pervious area
TC - total tree cover (over both pervious + impervious)
IA - total impervious cover (open impervious area + impervious area under tree cover)
SV - short vegetation cover
BS - bare soil cover

nonRouted - runoff has not been distributed in time using the 2 parameter surface routing equation

VegCanopyFile Column Definitions

Rain - total rain falling onto the project area
SWE - total snow water equivalent falling onto the project area
Rain_TC - rain falling on the tree canopy
SWE_TC - snow water equivalent falling on the tree canopy
Intercept_TC_W - rain intercepted by the tree canopy
Intercept_TC_SWE - snow water equivalent intercepted by the tree canopy
Evap_TC_W - evaporation of water from tree canopy leaf storage
Evap_TC_SWE - evaporation of snow water equivalent from tree canopy leaf storage
LeafStore_TC_W - water stored in tree canopy leaf storage
LeafStore_TC_SWE - snow water equivalent stored in tree canopy leaf storage
ThroughFall_TC_W - water throughfall through the tree canopy
ThroughFall_TC_SWE - snow water equivalent throughfall through the tree canopy
TreeCanopy_WB - tree canopy water balance (SWE and W combined)
Rain_SV - rain falling on the short veg canopy
SWE_SV - snow water equivalent falling on the short veg canopy
Intercept_SV_W - rain intercepted by the short veg canopy
Intercept_SW_SWE - snow water equivalent intercepted by the short veg canopy
Evap_SV_W - evaporation of water from short veg canopy leaf storage
Evap_SV_SWE - evaporation of snow water equivalent from short veg canopy leaf storage
LeafStore_SV_W - water stored in short veg canopy leaf storage
LeafStore_SV_SWE - snow water equivalent stored in short veg canopy leaf storage
ThroughFall_SV_W - water throughfall through the short veg canopy
ThroughFall_SV_SWE - snow water equivalent throughfall through the short veg canopy
ShortVegCanopy_WB - short veg canopy water balance (SWE and W combined)

WaterOnPerArea Column Definitions

Rain - total rain falling onto the project area
Rain_BS - rain falling onto the bare soil area
ThroughFlow_TCPA_W - throughflow of water onto the pervious area under tree cover
ThroughFlow_SV_W - throughflow of water onto the pervious area under short veg cover
SnowMelt_TCPA - snow melt on the pervious area under tree cover
SnowMelt_SV - snow melt on the pervious area under short veg cover
SnowMelt_BS - snow melt on the pervious area bare soil
Water_PA - total water on the total pervious area
PerArea_WB - water balance for the water on pervious area routines
SWE - total snow water equivalent falling onto the project area
SWE_BareSoil - total snow water equivalent falling onto the bare soil area
ThroughFlow_TCPA_SWE - throughflow of snow water equivalent onto the pervious area under tree cover
ThroughFlow_SV_SWE - throughflow of snow water equivalent onto the pervious area undershort veg cover
SWE_on_TCPA - snow water equivalent on the pervious area under tree cover
SWE_on_SV - snow water equivalent on the pervious area under short veg cover
SWE_on_BS - snow water equivalent on the bare soil area
SnowSub_TCPA - sublimation of snow water equivalent on the pervious area under tree cover
SnowSub_SV - sublimation of snow water equivalent on the pervious area under short veg cover
SnowSub_BS - sublimation of snow water equivalent on the bare soil area
PerArea_SWEB - snow water equivalent balance for the SWE on pervious area routines
PerArea_comb_WB - combined (summed) water balance for W and SWE on the pervious area routines
Water_PA - total water on the total pervious area
Run_on_from_IA - overflow from the depression storage of the total impervious area to pervious areas
Inflow_to_PerDep - total flow into pervious depression storage (summed run-on from IA + total water on PA)
PerDepStor_PA - depression storage of the total pervious area
PerDepEvap_PA - evaporation from depression storage of the total pervious area
PerFlow_To_Infil - overflow from the depression storage of the total pervious area to the infiltration routine
PA_Flow_To_Infil_WB - water balance for the pervious area flow to infiltration routines

WaterOnImpArea Column Definitions

Rain_OIA - rain on the open impervious area
ThroughFlow_TCIA_W - throughflow of water onto the impervious area under tree cover
SnowMelt_OIA - snow melt on the open impervious area
SnowMelt_TCIA - snow melt on the impervious area under tree cover
Water_IA - total water on the total impervious area
ImpArea_WB - water balance for the water on impervious area routines
SWE_OIA - snow water equivalent on the open impervious area
ThroughFlow_TCIA_SWE - throughflow of snow water equivalent onto the impervious area under tree cover
SWE_on_OIA - snow water equivalent on the open impervious area
SWE_on_TCIA - snow water equivalent on the impervious area under tree cover
SnowSub_OIA - sublimation of snow water equivalent on the open impervious area
SnowSub_TCIA - sublimation of snow water equivalent on the impervious area under tree cover
ImpArea_SWEB - snow water equivalent balance for the SWE on impervious area routines
ImpArea_comb_WB - combined (summed) water balance for W and SWE on the impervious area routines
ImpDepStor_IA - depression storage of the total impervious area
ImpDepEvap_IA - evaporation from the depression storage of the total impervious area
ImperFlow_Soil - overflow from the depression storage of the total impervious area to pervious areas
ImperFlow_Outlet_nonRouted - overflow from the depression storage of the total impervious area to the runoff routing routine
ImpArea_Runoff_WB - water balance for the impervious area runoff generation routines

Evap_ET Column Definitions

Evap_TC_W - evaporation of water from tree canopy leaf storage
Evap_TC_SWE - evaporation of snow water equivalent from tree canopy leaf storage
Evap_SV_W - evaporation of water from short veg canopy leaf storage
Evap_SV_SWE - evaporation of snow water equivalent from short veg canopy leaf storage
Infiltration_PA - water infiltrated by the pervious area
rET_VegCover - real (not potential) evapotranspiration

Temperature model outputs

This section lists potential outputs from the Cool Air model. To configure Cool Air outputs, see the Settings for Cool Air section and use the information below as a guide on available output options.

Two main types of output are available from the temperature model:

- Cell-based Output: results specific to a location on the gridded spatial inputs, referred to as a cell. These outputs are a timeseries of results for the specified cell.
- Time-based Output: results specific to a timestep during the simulation period, displayed as an ASCII raster map of outputs for all cells during the specified timestep.

There are also Blockgroup-based Output available but not yet stable, as the output metrics included in those files are unique and still in development. Land Cover Scaling-specific output is written by the Land Cover Scaling C# program.

Cell-based Output

The following gives examples of potential TemperatureExecutionParams options to generate different varieties of Cell-based Output. For a complete list of potential config file output tags, see Example of all options for TemperatureExecutionParams.

```
	Example C:
	Print Row#Col#Heat.txt and Row#Col#Hydro.txt for all locations in the DEM for the
time range specified. June 24, 2015 from the 1st hour/time step to the last hour/timestep
of that day; and June 26-28, 2015.
	<TemperatureExecutionParams>
		<RefWeatherLocation>1,42</RefWeatherLocation>
		<PrintRowCol>All</PrintRowCol>
		<RowColOutputPeriod0>2015062400,2015062423</RowColOutputPeriod0>
		<RowColOutputPeriod1>2015062600,2015062823</RowColOutputPeriod1>
	</TemperatureExecutionParams>

	Example D: Print Row#Col#Heat.txt for specific locations in the DEM for all
timesteps in the simulation (RowColOutputPeriod = All).

	<TemperatureExecutionParams>
		<RefWeatherLocation>1,42</RefWeatherLocation>
		<PrintRowCol0>5,3</PrintRowCol0>
		<PrintRowCol1>2,2</PrintRowCol1>
		<RowColOutputPeriod>All</RowColOutputPeriod>
	</TemperatureExecutionParams>
```

Description of Cell-based Output in Source Code: TemperatureOutputWriter.h

```
2. RowCol Output

*Functions related to the creation of Row#Col#Heat.txt # is a placeholder for DEM
location row number and col number
	ts	Ta	Td	Ea	ImpNR	TreeNR	ShortVegNR	SoilNR	ImpH	ImpLE
TreeH	TreeLE	TreeLEE	TreeLET	ShortH	ShortLE	ShortLEE	ShortLET
SoilH	SoilLE	H	LE
	2015062409	297.025	288.089	0.012781	68.0026	74.0514	60.2795	0	85.4408
0	-171.964	222.164	189.708	32.4559	-180.26	240.489	187.167	53.3219	0	0	35.794
43.7463
	static void createHeatRowColFile(int i, int j, std::string outDirectory);
	-The above function creates the file and adds the header line
	static void writeToHeatRowColFile(int timeStep, int i, int j, DataFolder *folder,
std::string outDirectory);
	-The above function writes output for a specific DEM location for a specific time
step
*Functions related to the creation of Row#Col#Hydro.txt # is a placeholder for DEM
location row number and col number
These are currently not in use!!!
	-static void createHydroRowColFile(int i, int j, std::string outDirectory, Inputs
*input);
	-static void writeToHydroRowColFile(int i, int j, DataFolder *folder, std::string
outDirectory);
*Helper functions for generating row#col# files
	-static void rowColFileBatchProcessor(int timeStep, int organizerLocationIndex,
DataFolder *folder, std::string outDirectory, Inputs *input);
	The above function determines if the data for a location at a specific time step
is written to its
	corresponding heat file
	-static void generateRowColFiles(DataOrganizer *organizer, Inputs *input);
	The above function creates all rowcol files for the simulation run
```

Time-based Output

```
	Example E:
```

For each timestep within 2015062400,2015062423 range, print a file for a specific variable such as AirT#.txt (where # is the timestep#). When PrintTimeBasedResults is set to All, All variable files will be created containing that value for each location in the DEM. (See [below in Time-Based Output Variables Available section, or] function TemperatureOutputWriter::timeStepResultsFileProcessor for complete list of variables.)

```
<TemperatureExecutionParams>
	<RefWeatherLocation>1,42</RefWeatherLocation>
	<PrintTimeBasedResults>All</PrintTimeBasedResults>
	<TimeBasedOutputPeriod0>2015062400,2015062423</TimeBasedOutputPeriod0>
<TemperatureExecutionParams>
```

Example F:
When PrintAllTimeBasedResults is set to AirT, only AirT#.txt files will be printed for the following three time periods.

```
<TemperatureExecutionParams>
	<RefWeatherLocation>1,42</RefWeatherLocation>
	<PrintTimeBasedResults>AirT</PrintTimeBasedResults>
	<TimeBasedOutputPeriod0>2015062400,2015062423</TimeBasedOutputPeriod0>
	<TimeBasedOutputPeriod1>2015062600,2015062723</TimeBasedOutputPeriod1>
	<TimeBasedOutputPeriod2>2015062800,2015063000</TimeBasedOutputPeriod2>
<TemperatureExecutionParams>
```

Example G:
In the below example, all variable files AirT#.txt, TD#.txt, ect. files will be printed for every timestep in the simulation.

```
<TemperatureExecutionParams>
	<RefWeatherLocation>1,42</RefWeatherLocation>
	<PrintTimeBasedResults>All</PrintTimeBasedResults>
	<TimeBasedOutputPeriod>All</TimeBasedOutputPeriod>
<TemperatureExecutionParams>
```

Time-Based Output Variables Available

All equations referenced in the table below are from Yang et al. (2013).

Variable ID used in Config File	Meaning
AirE	Air absolute humidity (kg/m^3); Eq 18
AirT	Air temperature (K)
Td	Dew-point temperature (K)
H	Total sensible heat flux (W/m^2); Eq 1; 12
ImperviousH	Impervious sensible heat flux (W/m^2); Eq 3, 19
ImperviousLE	Impervious latent heat flux (W/m^2); Eq 4; 19
ImpNR	Impervious net radiation (W/m^2); Eq 9, 24
LE	Total latent heat flux (W/m^2); Eq 2; 13
ShortVegH	Short veg sensible heat flux (W/m^2); Eq 5, 22 & 23
ShortVegLE	Short veg latent heat flux (W/m^2); Eq 6, 21
ShortVegLEE	Short veg latent heat flux from evaporation (W/m^2); Eq 22

ShortVegLET	Short veg latent heat flux from transpiration (W/m^2); Eq 23
ShortVegNR	Short veg net radiation (W/m^2); Eq 10, 24
SoilH	Soil sensible heat flux (W/m^2); Eq 7, 20
SoilLE	Soil latent heat flux (W/m^2); Eq 8, 20
SoilNR	Soil net radiation (W/m^2); Eq 11, 24
TreeH	Tree sensible heat flux (W/m^2); Eq 5, 22 & 23
TreeLE	Tree latent heat flux (W/m^2); Eq 6, 21
TreeLEE	Tree latent heat flux from evaporation (W/m^2); Eq 22
TreeLET	Tree latent heat flux from transpiration (W/m^2); Eq 23
TreeNR	Tree net radiation (W/m^2); Eq 10, 24

Description of Time-based Output in Source Code: TemperatureOutputWriter.h

```
3. Time Based output

*Functions related to the creation of timestep files containing the specified variable
value for each location
in the DEM.
	-static void createTimeStepResultFile(std::string outputVarName, std::string date,
Inputs *input);
	The above function creates the file for a specific result and date
	-static void writeToTimeStepResultFile(std::string outputVarName, std::string
date, std::string & outDirectory, double val, bool newLine);
	The above function adds the value for a specific location's variable value to
appropriate file. The value is written
	to the same DEM row/col location.
	-static void timeStepResultsFileProcessor(int timeStep, int
organizerLocationIndex, DataFolder *folder, std::string outDirectory, Inputs *input);
	The above function coordinates the creation of all result files and addition of
each locations results to created files
```

Tips & Troubleshooting

Local work environment

- The **filepath** to your working copy of the code should not have any spaces in it. It's been found that a space in the filepath will cause the testing scripts to fail.
    - OK: C:\Users\coviller**Dev_Files**\SVN\hydrologyv6\branches\HydroPlus
    - Not OK: C:\Users\coviller**Dev Files**\SVN\hydrologyv6\branches\HydroPlus

Version Control and Testing

To maintain working code that can support further development and troubleshooting, we rely on 3 code management strategies.

- **Version Control**: keeping track of different iterations, variations, and checkpoints of the code.
- **Standard Operating Procedure for Testing Code**: a consistent and efficient way to check of a reliable set of inputs produces the expected set of outputs in various modes and conditions.
- **Standard Operating Procedure for Updating Code**: a consistent and reliable way to build upon stable code with minimal introduction of bugs or divergence from the stable code.

Version Control

The two major systems for version control are SVN and git. i-Tree projects first version controlled in 2018 or later tend to be stored on git at https://code.itreetools.org while older projects are stored on SVN at https://svn.itreetools.org/usvn.

If you are beginning to work on HydroPlus R&D and need access to SVN, please register a new account at https://code.itreetools.org and notify project leader James.Kruegler@davey.com.

Version control is generally organized in a hierarchy with 'repositories' or 'projects' at the top level of each version control tree.

Locations

HydroPlus – the C++ model code for i-Tree Hydro, i-Tree Cool Air, and related models and modes – is version controlled using SVN. Its repository is at https://svn.itreetools.org/svn/hydrologyv6 and https://svn.itreetools.org/svn/hydrologyv6/trunk

LocalizeEMCs – two Python models for extracting event mean concentration (EMC) or export coefficient (EC) values from the White et al. (2015) database, creating output needed by Buffer.py– is version controlled on git at https://code.itreetools.org/ESF/Hydro/LocalizeEMCs_git

Buffer – the Python model for identifying non-point source runoff pollution loading hotspots – is version controlled on git at https://code.itreetools.org/ESF/Hydro/Buffer_git.

Energy – the Python model for mechanistically predicting shade tree benefit for building heating & cooling loads – is version controlled on git at https://code.itreetools.org/ESF/Hydro/Energy_git

SVN Concepts and Definitions

TortoiseSVN is used to interact with the SVN repository for HydroPlus. This TortoiseSVN software should be installed on the local machine, and integrated into the Windows apps and programs so it is available from File Explorer.

The TortoiseSVN manual is useful reference:
https://tortoisesvn.net/docs/release/TortoiseSVN_en/index.html

TortoiseSVN Subversion Control Procedures

TortoiseSVN offers SVN commands, and those most frequently used are discussed below.

1. SVNCheckout (right click on new local folder from File Explorer) is used once on a folder to establish the link between the local folder and the itreetools SVN URL and obtain the files from the server. The SVNCheckout process will default download the head or latest revision of entire solution from the URL to the local folder. An older revision can be selected, and revisions can be explored by using the Show Log features in this command.
2. SVN Update (right click on new local folder from File Explorer) is used each day after establishing a connected folder with SVNCheckout, in order to get the latest revision, presuming someone had made and committed changes. It will automatically update to the latest revision, or head.
3. SVN Commit (right click on new local folder from File Explorer) is used after each local revision has been made and the standard operating procedures of checking test cases. This command provides a text window to enter comments, which should include details on the what, why, how of files that were modified, and a confirmation that the test cases pass. This command also allows for selecting which files should be committed to the SVN URL.
4. TortoiseSVN (right click on new local folder from File Explorer) will open another list of commands in File Explorer, including Show Log, Repo – browser, Merge, Update to Revision, and many others. All are potentially useful for some situations.

When using SVN, it is important to understand the difference between local content versus server-side content. Given the differences between those two storage areas, the following should never be included in version control:

- Build output from project (can always be rebuilt from source)
- Nuget local packages directory (should always be downloaded)
- *.suo and *.user files for Visual Studio repositories (these are only used by VS to retain current development layout.. E.g. which files are open and their position.. committing these will lead to many merge conflicts)

Git Subversion Control Procedures

TortoiseSVN offers git commands and those most frequently used are discussed below. Commonly used Xterm window or DOS git commands are also presented below.

TortoiseSVN

1. Git Clone (right click on new local folder from File Explorer) is used once on a folder to establish the link between the local folder and the itreetools Git URL and obtain the files from the server. The Git Clone process will default download the head or latest revision of entire solution from the URL to the local folder. Open the .git config file to confirm the clone was successful, and there is the proper URL and a "merge" field, with the path to the branch or master.
2. TortoiseGit > Pull (right click on new local folder from File Explorer) is used each day after establishing a connected folder with Git Clone, in order to get the latest revision, presuming someone had made and committed changes. It will automatically update to the latest revision, or head.
3. Git Commit-> <target> (right click on new local folder from File Explorer) will open window to enter comments, add files with changes, and click on Commit & Push button
4. TortoiseGit (right click on new local folder from File Explorer) will open new window within File Explorer with many commands, including Diff, Revert, and Show Log to explore local and URL issues

Xterm DOS

1. git clone <URL> or git clone -b <branch> <URL> is used once on a folder to establish the link between the local folder and the itreetools Git URL and obtain the files from the server. The git clone process will default download the head or latest revision of entire solution from the URL to the local folder. Open the .git config file to confirm the clone was successful, and there is the proper URL and a "merge" field, with the path to the branch or master.
2. git pull, git diff, and git status within the git folder explore local and URL issues
3. To commit, three steps are required:
    a. git add <filename> or git add –a to place one file or all files into staging area
    b. git commit -m "comments on changes" with –m indicating a message in “” follows
    c. git push will move staged files to URL.

Utilities and models related to HydroPlus are developed and stored on git repositories. Below is an example workflow for development of code on one of our git repositories. Other git notes follow.

1. Checkout a local copy of GLRI_AutoRR
2. Modify, delete, add content on local copy
4. Use "git status" to confirm changes are as expected
5. Use "git add ." to stage all changes for commit
6. Use "git commit -m <message in quotes>" to commit changes with brief description of revision's changes. E.g. git commit -m "Changing DEM delineation module to use function from external"
7. Use "git push" to send your local revisions/commits to the server. For this command to work you may

need to setup your git working space. If so, use "git remote get-url" to confirm your remote (server-side) URL and "git remote set-url https://code.itreetools.org/Users/rcoville/GLRI_AutoRR" to (re)set it if needed. Other common issues include other git remote settings or credential commands needed.

Caution: Be careful with big files on version control. Once something is committed to the version control server, it's stored even if deleted in a future revision, as intended by version control tracking old versions. It is difficult or unfeasible to free up storage space from old revisions. As an alternative to committing large files, consider using a placeholder documentation file instead, so users could checkout the program and use the documentation to reproduce/source any large files needed. i-Tree Buffer has an example of this: https://code.itreetools.org/Users/rcoville/Buffer in the Constants folder, which in a fully-functioning working copy is ~180GB but on version control is kept smaller with a descriptive xlsx file to provide guidance about large files that would be included off version control.

A comprehensive list of files to be ignored for git-based version control is available here, which can serve as a reference for SVN use: https://github.com/github/gitignore/blob/master/VisualStudio.gitignore

SOP for Testing Code

A test script with sample inputs and expected outputs are setup to streamline model testing. The standard test procedure will compile a working copy of the HydroPlus code and run it against the tests defined in \HydroPlus\TestingFilesAndScript\ListOfTests.txt.

Procedure for Running Test Script

1. If you have not yet set up Visual Studio 2019 and compiled a new copy of the HydroPlus EXE, follow the instructions in the Development Notes section “Visual Studio setup for developing & testing code” to prepare for running the test script.
2. Go to the /TestingFilesAndScript/ directory (e.g. https://svn.itreetools.org/svn/hydrologyv6/trunk/TestingFilesAndScript/) and open ListOfTests.txt.
3. Within ListOfTests.txt, confirm that the first line of the file defining the Executable location points to the correct location for the compiled HydroPlus.exe to be tested: “Executable,..\HydroPlus\Release\HydroPlus.exe” This should be the case by default so that Steps 1 & 2 require no action. Close ListOfTests.txt.
4. Run the Python 3 script “TestToCompareHPOutputToStandards.py” from a command prompt or your preferred Python IDE.
    a. Nothing in this file or standard inputs should need to be changed; the script will use the HydroPlus.exe in /HydroPlus/Release/ along with the various input and expected output sets to test that EXE against standard results, ensuring all HydroPlusConfig.xml files are set to appropriate input/output paths.

   b. This script will not pause between simulations for users to view terminal output, but a log.txt file will be generated in the same directory as the script for users to review terminal output after all simulations are complete.
5. The script will create a new directory named "Testing_YYYY_MM_DD_hh_mm_ss". In that directory, the "results.txt" file will identify which files in which test cases failed if any. If any tests failed, compare the specified output file to the expected output file to begin troubleshooting. Notepad++ offers a comparison tool for side-by-side difference identification in each output.
6. If outputs are only slightly different that may be acceptable, or if certain expected outputs are not yet reliable it may be OK if the code being tested doesn't match. On the other hand, seemingly minor differences in formatting could cause problems in production, so carefully consider any differences.
7. Once all differences are resolved and the working copy of the code is performing as expected, you can commit it to SVN, and in the SVN commit message summarize the changes that have been made in this revision. Try to use key words that may be searched for later on, making it easier for future development to browse revision history for specific changes.

SOP for Developing New Features

During development, the following procedure should be followed to maintain stable code and minimize divergence from the stable code. Note that this procedure was developed for GI development going on simultaneously with HydroPlus development; a more relaxed procedure might work during less challenging development conditions.

Each day during development:

1. Begin by updating your working copy to the latest revision of the main code, to ensure your work is in sync with that.
2. Develop features as needed. Try to avoid writing redundant code (e.g. if the function you need exists for a different model mode, find a way to use it or use functors to use a variation of it) and try to keep things modular and parsimonious. Importantly: test and troubleshoot as you go; do not try to build upon broken code. More guidance in the Architecture section.
3. Run the standard test scripts to ensure that the day's work has not broken the stable code.
4. If the scripted tests show that the working copy of the code fails to produce expected outputs in stable model modes, then you need to restore the code to working order before committing it each day. (Restoring the code to working order could be as simple as commenting out the newer work for the day, or just commenting out the troublesome parts of it.)
5. Once the code is in working order, commit your work for the day, every day that you work on it. That way a record and backups of work in progress is maintained, keeping new development in sync with the main code and ensuring new code isn't being built on broken code.

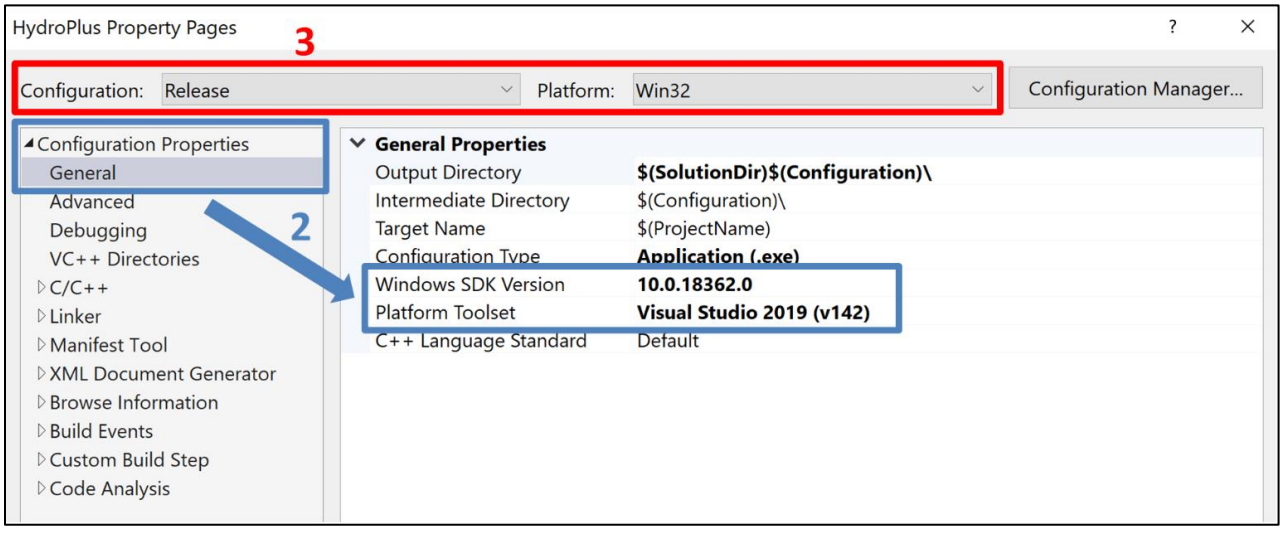
When making comments (at least for temporarily-commented out code), sign with author and latest date. Indicating latest date will help us estimate if 'old' commented out code can be deleted from future revisions, or if it is freshly commented out code we'll know to leave it in case it is work-in-progress.

Development Notes

Visual Studio setup for developing & testing code

The HydroPlus development team uses Visual Studio 2019 to view, edit, and compile the C++ source code. The VS software plays a critical role in how HydroPlus is developed and tested. The "Community Edition" of VS 2019 is free to download and use.

1. Install Visual Studio 2019 Community Edition if you have not already. In the installation you will need certain development components to compile HydroPlus.
    a. To change what components have been installed, open the Visual Studio Installer, go to the 'Installed' tab, then select the 'Modify' button on VS 2019. Select the "Desktop development with C++" workload, or select the following individual components:
        i. MSVC v142 - VS 2019 C++ x64/x86 build tools (v14.28) or later
        ii. Windows 10 SDK (10.0.18362.0) or later
        iii. Just-In-Time Debugger
        iv. C++ profiling tools
2. Open the HydroPlus solution (e.g. https://svn.itreetools.org/svn/hydrologyv6/trunk/HydroPlus/HydroPlus.sln) in Visual Studio.
    a. Ensure that the HydroPlus project is set to compile with the needed configuration properties. To do so, right click on the Hydro project in Visual Studio's Solution Explorer, and select Properties. In the window that appears, click the 'General' option under 'Configuration Properties'. Implement the settings shown below:
        i. Windows SDK Version = 10.0.18362.0 (or later)
        ii. Platform Toolset = Visual Studio 2019 (v142)



3. At the top of the same window, set the Configuration to “Release” mode and Platform to “Win32” (usually it will be in Debug, Win32 mode during development). Close the Properties window, and make sure the same configuration options are selected in the top toolbar.
4. Go to Build > Rebuild Solution to compile a fresh copy of HydroPlus.exe. The Output window in Visual Studio should show compilation processes and will end by stating if the solution compiled successfully or failed.
    a. If the Build fails due to errors related to missing math.h or other basic libraries, consider following advice in StackExchange, using the Visual Studio Installer > Individual components > and under the heading “Compilers, build tools, and runtimes” check the box for “Windows Universal CRT SDK”.
5. Find the /Release/ folder in the same directory as the HydroPlus.sln file, and in that /Release/ folder confirm there is a HydroPlus.exe created as recently as expected.

Config file modifications

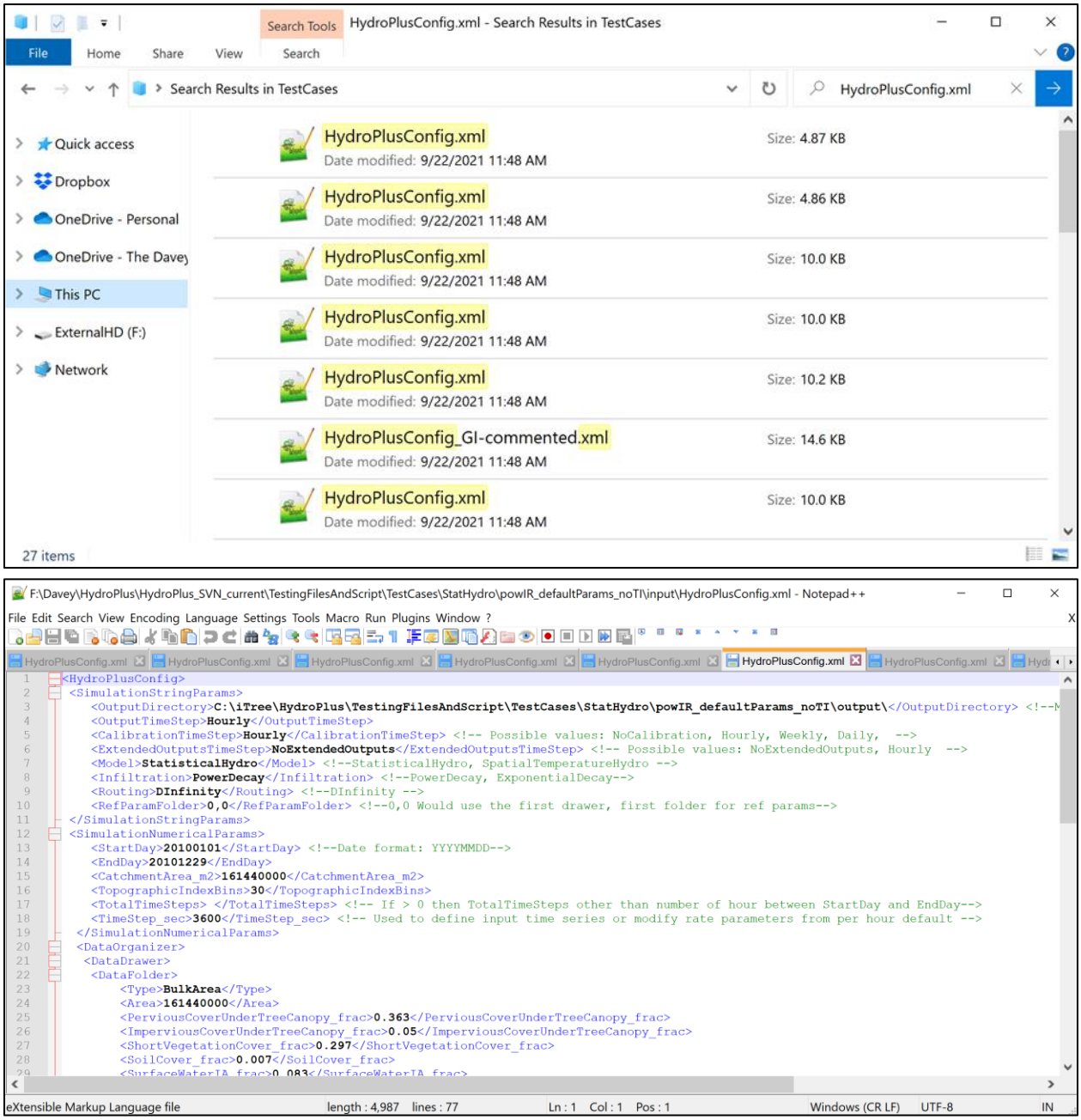
The config files included in /TestingFilesAndScript/TestCases/ serve as ‘master copy’ config files; as we update HydroPlus, users can compare older config files with the latest in TestCases to get their files up-to-speed with the latest config file features.

Because we have numerous TestCases, config file changes need to be made in batch. Superficial changes are those that only affect the config file (e.g. changing comments), as compared with other changes (e.g. changing tag names) that require accompanying code changes.

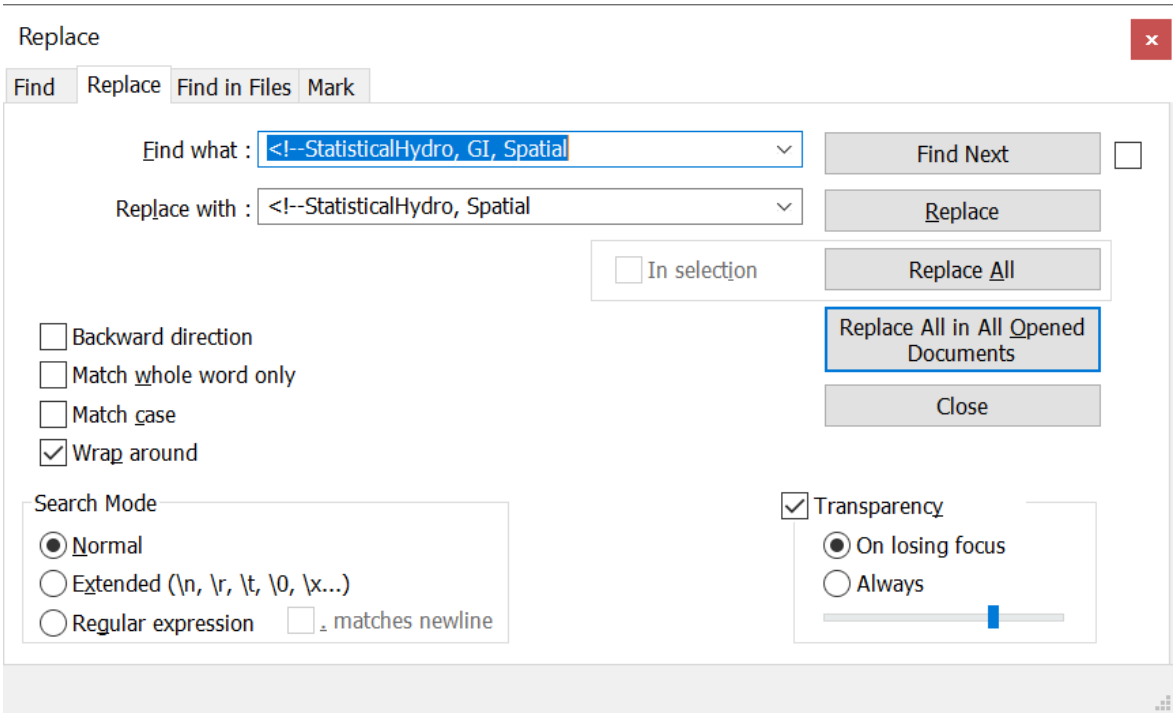
Batch superficial changes (file only, no code changes)

1. Open all HydroPlusConfig.xml files contained in /TestingFilesAndScript/TestCases/

    Use Windows Explorer’s search feature, select all, and open all in Notepad++. To be safe, you may want to close all Notepad++ documents prior to opening all config files, to ensure your batch changes only apply to config files.

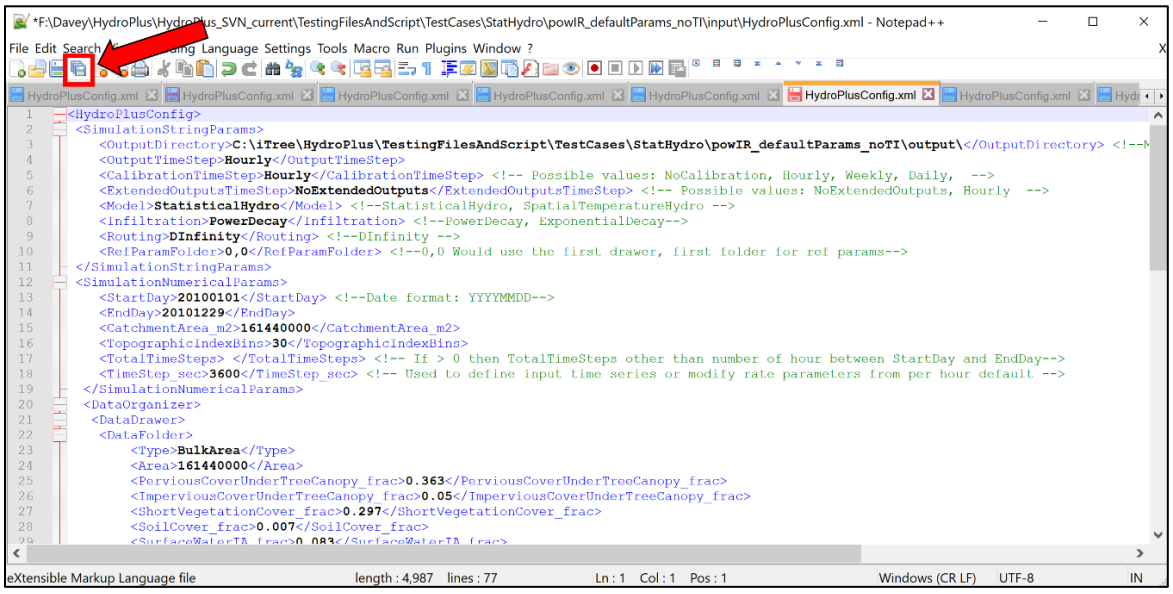


2. Use Find + Replace feature: Find to identify a string to be modified or to anchor a modification; and Replace to change that string or add content before or after it. In the example below, an outdated comment about available Model tag values is being changed to remove GI (since the GI model mode is now built into StatisticalHydro model mode).



Be careful to use a "Find" value detailed enough to ensure you only edit around the string you intend to. For example, if you searched for a string that occurs in multiple different tags, your Replace action might affect unintended tags. To be extra careful, you can use Find Next to identify all strings to be affected before choosing to Replace All in All Opened Documents. To check your work, you can use TortoiseSVN>Revert… or Commit… to view a list of files you've modified in your working copy of the SVN code repository, and you can right-click each file to view each specific difference highlighted. That way you'll see if the differences are what you intended or not before moving forward with development or committing your work.

3. Save changes to all files



4. Follow SOP for Testing Code to confirm your changes have not unintentionally affected results.

Code changes associated with config file changes

Basic workflow:

1. Find and replace all instances of relevant input-parsing code for config file tag being modified. Example: find and replace all *folder->ParamDict["Area"]* with *folder->ParamDict["BAorGIArea_m2"]* in HydroPlus.sln
2. Open all config files as tabs within a single Notepad++ window. Find and replace all, in all open documents. Example: *Area>* with *BAorGIArea_m2>*. That > symbol captures the necessary part of *<Area>* as well as *</Area>.*
3. Save all modified config files. Run test SOP. Parameter name changes should not require expectedoutputs change. If expectedoutputs do need to change, carefully check the new output and then commit that as the expectedoutputs if suitable.

Running HydroPlus in sub-hourly timesteps (Jan 21, 2019)

The parameter which controls all the timesteps related operations is the "`SimulationNumericalParams["TotalTimeSteps"]`". In the UH, we have the following line:

Inputs.cpp
#63, 64, & 65

```
d1 = ToJulianDate(SimulationNumericalParams["StartDay"]);
d2 = ToJulianDate(SimulationNumericalParams["EndDay"])+1;
SimulationNumericalParams["TotalTimeSteps"] = (d2 - d1) * 24;
```

Therefore, by modifying the TotalTimeSteps we can run the model in different timesteps. For example, to run a 5-min interval scenario for 2 hours, manually changing the TotalTimeSteps to 24 ((60/5)*2) in the code works. If we add the TotalTimeSteps parameter to the config file, the users could run the code in finer resolution than hourly based intervals.

yyyymmdd	hh:mm:ss	Precipitation	Total run off	Base flow(m)	Pervious flow	Impervious flow
20120508	12:00:00	0	1.6e-05	2.93245e-32	0	0
20120508	12:05:00	0	3.2094e-17	3.2094e-17	0	0
20120508	12:10:00	0	1.47769e-13	1.47769e-13	0	0
20120508	12:15:00	0	7.30526e-12	7.30526e-12	0	0
20120508	12:20:00	0	7.17038e-11	7.17038e-11	0	0
20120508	12:25:00	0	3.24432e-10	3.24432e-10	0	0
20120508	12:30:00	0	9.51454e-10	9.51454e-10	0	0
20120508	12:35:00	0	2.13408e-09	2.13408e-09	0	0
20120508	12:40:00	0	4.00736e-09	4.00736e-09	0	0
20120508	12:45:00	0	6.64721e-09	6.64721e-09	0	0

20120508	12:50:00	0	1.0076e-08	1.0076e-08	0	0
20120508	12:55:00	0	1.42753e-08	1.42753e-08	0	0
20120508	13:00:00	0	1.91993e-08	1.91993e-08	0	0
20120508	13:05:00	0	2.47856e-08	2.47856e-08	0	0
20120508	13:10:00	0	3.09641e-08	3.09641e-08	0	0
20120508	13:15:00	0	3.76624e-08	3.76624e-08	0	0
20120508	13:20:00	0	4.48097e-08	4.48097e-08	0	0
20120508	13:25:00	0	5.23391e-08	5.23391e-08	0	0
20120508	13:30:00	0	6.01887e-08	6.01887e-08	0	0
20120508	13:35:00	0	6.83024e-08	6.83024e-08	0	0
20120508	13:40:00	0	7.66298e-08	7.66298e-08	0	0
20120508	13:45:00	0	8.51262e-08	8.51262e-08	0	0
20120508	13:50:00	0	9.3752e-08	9.3752e-08	0	0
20120508	13:55:00	0	1.02472e-07	1.02472e-07	0	0

Land cover scaling simulations (LCscaling utility)

Hydro LCscaling analysis

HydroPlus was modified in early 2019 to interface with the "LCscaling" utility, which batch runs UH through increments of tree cover and impervious cover to assess a gradient of land cover change effects. This 'land cover scaling' analysis has been used since before 2015 with the semi-distributed Hydro model, using a tool called 'AutoRun'. The 'new and improved' LCscaling utility is designed to work with the semi-distributed Hydro model as well as the fully-distributed Cool Air model.

To use that LCscaling tool:

1. Check it out from https://code.itreetools.org/svn/hydro/branches/LCscaling
2. Place a HydroPlus.exe (freshly compiled in Release mode from https://code.itreetools.org/svn/hydrologyv6/trunk) in …\LCscaling\AutoRun\HydroBin\
3. Compile AutoRun.sln and use interface to interact with LCscaling analysis. This runs HydroPlus.exe from Step 2 in config file setting <Model>LCScalingTempSpatial</Model>.

History

This utility was developed by Thomas Taggart (SUNY-ESF) and Pallavi Iyengar (Syracuse University) beginning June 2014 (or potentially earlier with Yang Yang (SUNY-ESF)), to produce results for EPA EnviroAtlas datasets, collaborating with David Nowak (USFS) and Theodore Endreny (SUNY-ESF) on utility design and I/O goals. Shannon Conley (Davey Institute) worked with Robert Coville (Davey Institute) and Theodore Endreny (SUNY-ESF) to update AutoRun for use with HydroPlus.exe, which began April 2019 with the initiation of the LCscaling branch of the Hydro SVN repository (referenced above). Nakul Sahayata (Syracuse University) began to assist with LCscaling development in March 2020.

This utility continues to be developed for use in GLRI Forest Planning & NUCFAC Temperature-Health R&D projects.

Cool Air LCscaling analysis

Note that for Cool Air, an alternative to the LCscaling utility was developed. That uses LCscaling input grids in a single program execution instead, and uses blockgroup hourly outputs and a new <LCScalingOutput>Yes</LCScalingOutput> option in Cool Air config file's <TemperatureExecutionParams> section, to execute a batch interpolation feature for results from all 1% increments of TC & IC from 0-100%.

For example, Cool Air will run LCscaling interpolation using 2x2 input grids of edge case conditions and the following config file settings:

```
<TemperatureExecutionParams>
        <RefWeatherLocation>1,1</RefWeatherLocation>
        <PrintBlockGroupDailyAndHourly>True</PrintBlockGroupDailyAndHourly>
        <PrintBlockGroupRange0>1,4</PrintBlockGroupRange0>
        <LCScalingOutput>Yes</LCScalingOutput>
</TemperatureExecutionParams>
```

Conditions for TC+IC != 100%

In the Statistical Hydro model, all land cover parameters are explicitly defined and are expected to sum to 100%.

In the Spatial Temperature model, land cover for each dataFolder (a.k.a. cell, pixel in the area of interest raster grid) is defined by a raster layer for tree cover % (treecover.dat), impervious cover % (imperviousCover.dat), and NLCD Land Cover Class (landcover.dat). NLCD Land Cover (LC) classes are associated with different parameters for model subroutines, and these LC classes inform what other land cover is present when TC+IC<100%.

The conditions for TC+IC>100% or TC+IC<100% are defined (as of revision 473) in HydroPlus's LandCoverPCal.cpp file, in the LandCoverPCal::CalLandCover function. Comments in this section describe methods. In short, the excess when TC+IC>100% is assumed to be TreeOnImp (tree canopy covering impervious cover). The deficiency when TC+IC<100% is defined according to LC class, where special LC classes dictate what is filled in (e.g. in a 'open water' cell, water is filled in) or a standard method is used (as for LC21, filling the gap with ShortVeg and/or Soil).

Functors

Functors are used instead of complicated switches to plug in certain functions depending on a given condition. For example, whether the semi-distributed hydrology model is run or the spatially-distributed temperature model is run, most of the same functions will be called but certain processes will require

different functions depending on the specific model. Functors can control which functions are run depending on which model is being run.

In `RootZoneETCalc::calculate` a functor is called:

```
void RootZoneETCalc::calculate(Inputs *input, DataFolder *folder, int timeStep, std::map<std::string, void(*)(Inputs *input, DataFolder *folder, int timeStep)> &functors)
{
    // Resetting the SumET for each timestep/folder - run against V5 to make sure it works the same in semi-distributed mode (11/5/14) - it does ! 11/9/14
    folder->VarDict["sumET"] = 0.0;

    // rzMSD = Maximum root zone storage deficit - set in the param file
    double rzMSD = input->SoilParams["MSD_MaximumRootZoneStorageDeficit_meters"];

    // This for loop runs for each member of the Topographic Index Distribution - typically 30 values
    for(int ia=0; ia< input->IndexAreaSize; ia++)
    {
        if (timeStep == 0)
        {
            folder->VarVecDict["rET_TI"].push_back(0.0);
        }
        // Setting the rootzone ET equal to 0, for this index value
        folder->VarVecDict["rET_TI"][ia] = 0.0;
        input->RepoDict["iA"] = ia;
        functors["calculateRootzoneET"](input, folder, timeStep);
        // Total ET - for the whole run period? - was (incorrectly) but isn't now due to resetting the value at the beginning of this function
        // Total ET - For this timestep is previous ET value + the amount of ET from this index value * the fraction of the watershed area for this index value
        folder->VarDict["sumET"] += folder->VarVecDict["rET_TI"][ia] * input->IndexArea[ia];             // calculate total ET from the subcatchment
        // In affect this appears to be averaging the ET across the watershed (by Index areas - when in sem-dist mode) while summing it to total ET
    }
```

To find what this functor is doing, I searched “functor” in the entire solution and looked for where many appear to be defined. That’s in TestSimulation.cpp within the definition of `runBulkAreaCalculations`

```
void TestSimulation::runBulkAreaCalculations(DataFolder *folder, Inputs *input, int timeStep, std::vector<DataFolder*> &drawer)
{
    std::map<std::string, void(*)(Inputs *input, DataFolder *folder, int timeStep)> functors;
    if (input->InputOutputParams["Model"] == "StatisticalHydro")
    {
        functors.emplace("calculateTreeEvaporation", TreeEvaporationCalc::calculateTreeEvaporationStatistical);
        functors.emplace("calculateShortVegEvaporation", ShortVegEvaporationCalc::calculateShortVegEvaporationStatistical);
        functors.emplace("calculatePerDepEvap", PerviousDepressionStorageCalc::calculatePerDepEvapStatistical);
        functors.emplace("calculateImpEvap", ImperviousDepressionStorageCalc::calculateImpEvapStatistical);
        functors.emplace("calculateRootzoneET", RootZoneETCalc::calculateRootzoneETStatistical);
    }
    if (input->InputOutputParams["Model"] == "SpatialTemperatureHydro")
    {
        functors.emplace("calculateTreeEvaporation", TreeEvaporationCalc::calculateTreeEvaporationTemperature);
        functors.emplace("calculateShortVegEvaporation", ShortVegEvaporationCalc::calculateShortVegEvaporationTemperature);
        functors.emplace("calculatePerDepEvap", PerviousDepressionStorageCalc::calculatePerDepEvapTemperature);
        functors.emplace("calculateImpEvap", ImperviousDepressionStorageCalc::calculateImpEvapTemperature);
        functors.emplace("calculateRootzoneET", RootZoneETCalc::calculateRootzoneETTemperature);
    }
    if (input->InputOutputParams["Infiltration"] == "PowerDecay")
    {
        functors.emplace("calculateInfiltration", SoilInfiltrationPowDecayCalc::calculate);
        functors.emplace("calculateUnsatZoneSoilMoistureDeficit", UnsatZoneSoilMoistureDeficitPowDecayCalc::calculate);
```

What we see here is that, within the runBulkAreaCalculations function, there is basically as switch board saying what a functor should do depending on a certain input or condition. In the calculateRootzoneET functor highlighted above, depending on what model is being run, a different actual function (either calculateRootzoneETStatistical or calculateRootzoneETTemperature) is assigned to that functor. That way, back in RootZonETCalc::calculate, the functor calculateRootzoneET can be called and it will automatically activate the appropriate function depending on which model is running.

Build settings

In August 2020 it was observed that the Visual Studio setting for Whole Program Optimization was resulting in build failures for some developers. Whole Program Optimization is disabled moving forward for two reasons: 1) disabling that feature resolves the build errors some developers faced, and 2) this feature is not recommended for development cases like ours, where libraries may be shared across different versions of Visual Studio, resulting in unexpected behavior when this feature is enabled. More information about this feature is available at https://devblogs.microsoft.com/cppblog/quick-tips-on-using-whole-program-optimization/.

Feedback on C++ methods for GI dev (Oct 18, 2018)

The content below was shared by Shannon Conley in Oct '18 as feedback for Reza Abdi as rev279 had many compilation errors. Shannon resolved many of those errors and provided this feedback in rev280. Reza was simultaneously working on resolving many of those errors with feedback Robbie Coville provided, and Reza committed his changes to rev281. Content below provided as reference, eventually to be cleaned up & integrated into the rest of this manual.

Calculation Classes

These classes should contain only static function members. Highlighted in yellow is a static function declaration. Marked in red is not-static. A static function cannot call a non-static function. This results in a compilation error. Also, static functions cannot access data members. such as inflowFromGI_mm. This is by design. All data should be read from or stored in either the *input or *folder. If data is specific to a GI unit or bulk area it should be stored in one of the *folder dictionaries (will explain more below). If data is read from an input file or related to more than one unit then it would be stored in *input.

```
#include "../Inputs/Inputs.h"
#include "../DataFolder.h"

class WaterOnImperviousAreaCalc
{
    public:
        static void calculate(Inputs *input, DataFolder *folder, int timeStep);
        static void storeValues(Inputs *input, DataFolder *folder, int timeStep);
        double inflowFromGI_Imp();

    private:

        double inflowFromGI_mm = 0;

};
```

A static member function is used without creating an instance of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::. Below are some examples.

```
TreeInterceptionCalc::calculate(input, folder, timeStep);
ShortVegInterceptionCalc::calculate(input, folder, timeStep);
functors["calculateTreeEvaporation"](input, folder, timeStep);
functors["calculateShortVegEvaporation"](input, folder, timeStep);
PerviousAreaThroughFlowCalc::calculate(input, folder, timeStep);
SnowMeltOpenAreaCalc::calculate(input, folder, timeStep);
SnowMeltUnderTreeCalc::calculate(input, folder, timeStep);
SnowMeltUnderShortVegCalc::calculate(input, folder, timeStep);
WaterOnImperviousAreaCalc::calculate(input, folder, timeStep);
ImperviousDepressionStorageCalc::calculate(input, folder, timeStep, functors);
WaterOnPerviousAreaCalc::calculate(input, folder, timeStep, drawer);
PerviousDepressionStorageCalc::calculate(input, folder, timeStep, functors);
SemiAveSMDCalc::calculate(input, folder, timeStep);
functors["calculateInfiltration"](input, folder, timeStep);
```

This is a nice link explaining static concepts in C++.
https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm

The DataFolder class

Originally, this class was called Cell, but it seems to be a loaded term. A DataFolder can be created for each GIUnit. For instance, BulkArea will have its own instance of a DataFolder class. Only info specific to this unit should be stored here. There are four options for storing data.

1. `std::map<std::string, double> ParamDict;` This stores input data specific to the unit with the double datatype. So in a calculation, you can access PerDepStorage as highlighted below.

```
- <DataOrganizer>
  - <DataDrawer>
    - <DataFolder>
        <Type>BulkArea</Type>
        <AreaPer>1.0000</AreaPer>
        <Area>161440000</Area>
        <TreePerviousCover>0.363</TreePerviousCover>
        <TreeImperviousCover>0.05</TreeImperviousCover>
        <ShortVegCover>0.297</ShortVegCover>
        <SoilCover>0.007</SoilCover>
        <ImperviousCover>0.283</ImperviousCover>
        <DCIA>0.2685</DCIA>
        <ImpDepStorage>2.5</ImpDepStorage>
        <PerDepStorage>1.0</PerDepStorage>
      </DataFolder>
```

```
void PerviousDepressionStorageCalc::calculate(Inputs *input, DataFolder *folder, int timeStep, std::map<std::string,
{

    // perviousStorageMax = param.dat pervious storage maximum value (in mm) * 0.001 -> to convert to meters
    folder->VarDict["perDepStorMax"] = 0.001 * folder->ParamDict["PerDepStorage"];
    // perDepStorPrev = impDepStor  (Shifts the current pervious depression storage to the temp variable)
    double perDepStorPrev =  folder->VarDict["perDepStor"];
    double perPrecip =  folder->VarDict["waterToPerDep"];
    // increment in pervious Storage
```

2. std::map<std::string, std::string> ParamStringDict; This is the same as above, but stores data in string format.

3. This is used to store data created/used by calculations. As opposed to creating class data member. All data specific to a unit such as BulkArea should be stored here. You can create a new entry/start using it without explicitly declaring/defining it. By default, all values will be zero.

```
std::map<std::string, double> VarDict;
```

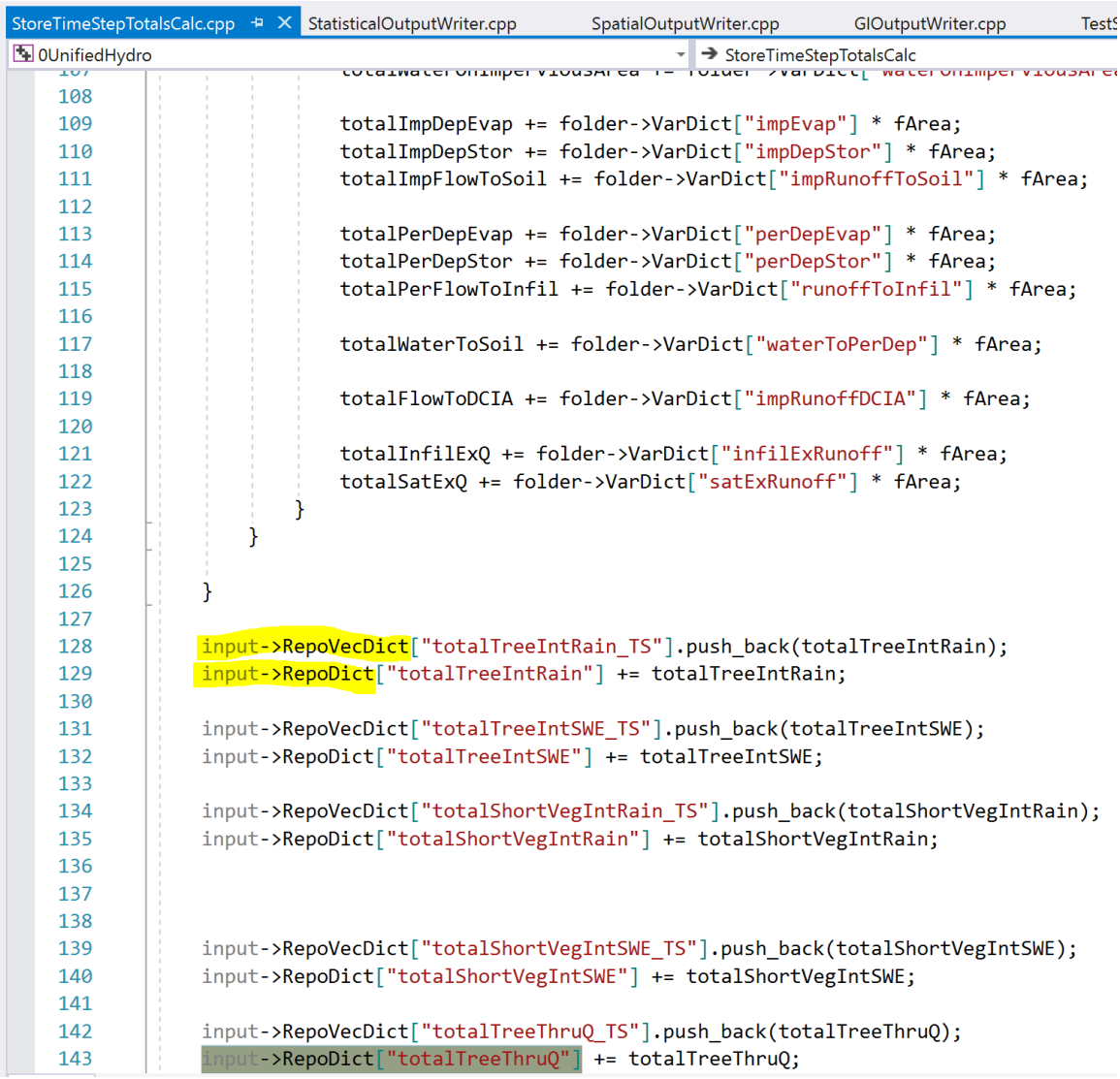
```
void PerviousDepressionStorageCalc::calculate(Inputs *input, DataFolder *folder, int timeStep, std::map<std:
{

    // perviousStorageMax = param.dat pervious storage maximum value (in mm) * 0.001 -> to convert to meters
    folder->VarDict["perDepStorMax"] = 0.001 * folder->ParamDict["PerDepStorage"];
```

4. std::map<std::string, std::vector<double> > VarVecDict; These store vectors unique to a unit. Use sparingly/can grow quite large/cause memory issues. Most should be deleted/legacy for writing extended output.

```
void PerviousDepressionStorageCalc::storeValues(Inputs *input, DataFolder *folder, int timeStep)
{
    folder->VarDict["totalperDepEvap"] += folder->VarDict["perDepEvap"];
    folder->VarDict["totalrunoffToInfil"] += folder->VarDict["runoffToInfil"];
    if (input->InputOutputParams["Model"]=="StatisticalHydro")
    {
        folder->VarVecDict["perDepEvap"].emplace_back(folder->VarDict["perDepEvap"]);
        folder->VarVecDict["perDepStor"].emplace_back(folder->VarDict["perDepStor"]);
        folder->VarVecDict["runoffToInfil"].emplace_back(folder->VarDict["runoffToInfil"]);
    }
}
```

If calculation data needs to be shared/passed between units, store it in input->RepoDict or input->RepoVecDict (Use RepoVectDict sparingly same issues as VarVecDict.)



Development method and troubleshooting

If a large amount of code has been added and is not compiling, strongly consider starting over from the most recent stable checkpoint on the HydroPlus webpage. First create functions that print the desired inputs to the console. Program a single-line calculation/compile/run/repeat. Afterwards, remove or comment out the print statements. Always make sure the code is working. Do not program with errors. If code cannot compile, you cannot debug. Even if the code compiles, this does not mean it is behaving as expected.

Architecture development Q&A for GI (Oct 26, 2018)

Questions from Reza Abdi & Robbie Coville, answers from Shannon Conley

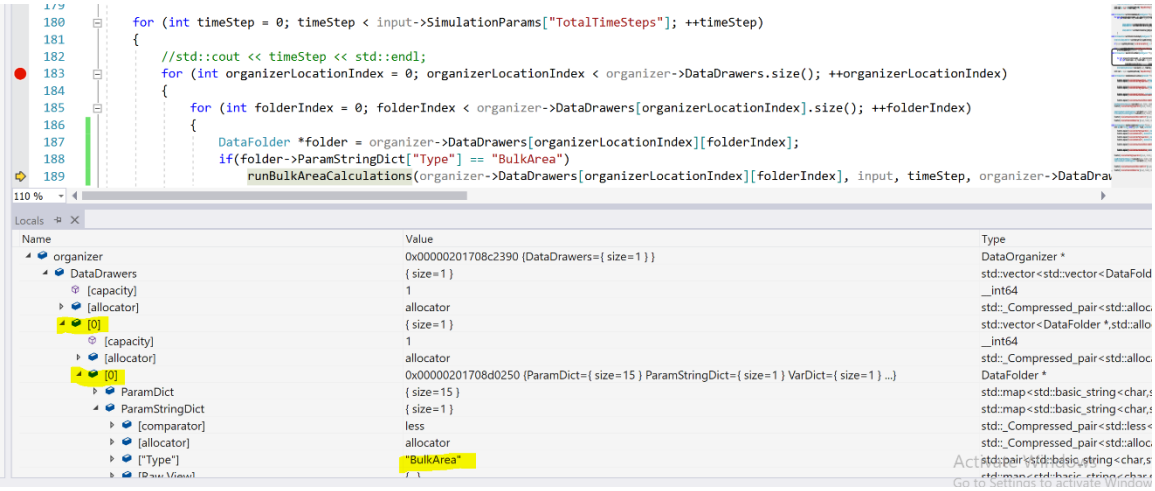
1. Where in the code do we begin running GI calcs? A potentially simple answer: in TestSimulation::RunStatisticalModel, after calling runBulkAreaCalculations, we call runGICalculations. The rationale for having a separate set of calculations to run for bulk area vs. GI is that, though most of

the calcs will be the same, GI may want to call a few extra processes; maybe this can be handled all within runBulkAreaCalculations using a functor based on the condition of what type of structure the code is currently running (Bulk Area vs. a GI type). That raises the next question (#2).

We use functors when the calculation has sections of code that differ. If the calculation is completely different, then we do not need a functor. Can just use 'if else'. We should rename runBulkAreaCalculations though. I have mapped this out in the GI code over the summer and need more time to explain it in detail.

2) What tells the code to grab inputs from the GI structure in the config file rather than bulk area, and how does the code keep track of which structure it is running? For example, bulk area vs. rain garden vs. rain barrel. We can start with just bulk area & rain garden only, but we should keep in mind the need for multiple GI types and even multiple instances of each structure type.

The 'type' tag determines how the code will treat the structure. All data related to a structure is stored within a data folder. To get the type, folder->ParamStringDict["Type"] (see code below). We can easily have multiple types and more than one of each. In the config file, DataOrganizer corresponds to an instance of the `DataOrganizer *organizer` class. This class contains a container called DataDrawers. For each DataDrawer tag in the config, a new vector of folders is added to this container. The size of each vector corresponds to the number of DataFolder xml elements enclosed. In the below xml example, there is only one DataDrawer with one DataFolder of type BulkArea. We keep track of each folder by its position within the vector of vectors. So we can think of this BulkArea folder as in the first position in the first drawer. Think of a filing cabinet where you open up the first drawer and pull out the first filing folder. If you want to add RainGarden you can place it in the same drawer or a different one. If you place in the same drawer, then the size of DataDrawers would remain 1 and the size of the vector at position 0 would increase to 2.



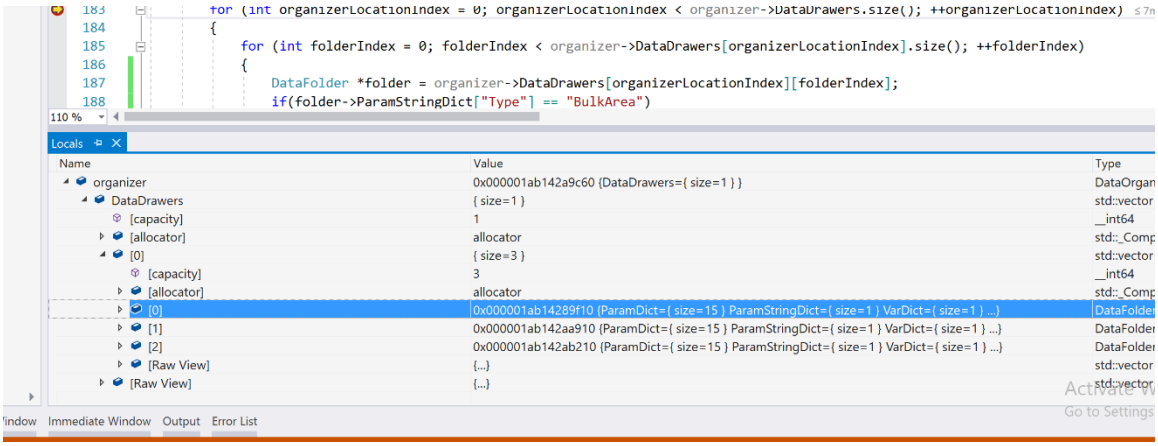
```
<DataOrganizer>
<DataDrawer>
<DataFolder>
    <Type>BulkArea</Type>
    <AreaPer>1.0000</AreaPer>
    <Area>161440000</Area>
    <TreePerviousCover>0.363</TreePerviousCover>
    <TreeImperviousCover>0.05</TreeImperviousCover>
    <ShortVegCover>0.297</ShortVegCover>
    <SoilCover>0.007</SoilCover>
    <ImperviousCover>0.283</ImperviousCover>
    <DCIA>0.2685</DCIA>
    <ImpDepStorage>2.5</ImpDepStorage>
    <PerDepStorage>1.0</PerDepStorage>
  </DataFolder>
</DataDrawer>
</DataOrganizer>
```

If you want to add two more BulkArea structures you can place it in the same drawer or a different one. If you place in the same drawer, then the size of DataDrawers would remain 1 and the size of the vector at position 0 would increase to 3. You distinguish between the three

```
organizer->DataDrawers[0][0]

organizer->DataDrawers[0][1]

organizer->DataDrawers[0][2]
```




```
<DataOrganizer>
 <DataDrawer>
 <DataFolder>
   <Type>BulkArea</Type>
   <AreaPer>1.0000</AreaPer>
   <Area>161440000</Area>
   <TreePerviousCover>0.363</TreePerviousCover>
   <TreeImperviousCover>0.05</TreeImperviousCover>
   <ShortVegCover>0.297</ShortVegCover>
```

```
    <SoilCover>0.007</SoilCover>
    <ImperviousCover>0.283</ImperviousCover>
    <DCIA>0.2685</DCIA>
    <ImpDepStorage>2.5</ImpDepStorage>
    <PerDepStorage>1.0</PerDepStorage>
   </DataFolder>
<DataFolder>
    <Type>BulkArea</Type>
    <AreaPer>1.0000</AreaPer>
    <Area>161440000</Area>
    <TreePerviousCover>0.363</TreePerviousCover>
    <TreeImperviousCover>0.05</TreeImperviousCover>
    <ShortVegCover>0.297</ShortVegCover>
    <SoilCover>0.007</SoilCover>
    <ImperviousCover>0.283</ImperviousCover>
    <DCIA>0.2685</DCIA>
    <ImpDepStorage>2.5</ImpDepStorage>
    <PerDepStorage>1.0</PerDepStorage>
   </DataFolder>
   <DataFolder>
    <Type>BulkArea</Type>
    <AreaPer>1.0000</AreaPer>
    <Area>161440000</Area>
    <TreePerviousCover>0.363</TreePerviousCover>
    <TreeImperviousCover>0.05</TreeImperviousCover>
    <ShortVegCover>0.297</ShortVegCover>
    <SoilCover>0.007</SoilCover>
    <ImperviousCover>0.283</ImperviousCover>
    <DCIA>0.2685</DCIA>
    <ImpDepStorage>2.5</ImpDepStorage>
    <PerDepStorage>1.0</PerDepStorage>
   </DataFolder>
  </DataDrawer>
 </DataOrganizer>
```

In the main branch of the Unified code in BuildDataOrganizer class, it not possible to add a folder with any other type than BulkArea (Oct 2018). This stop is in place because over the summer how to handler land cover hadn't yet been ironed out.

```
void BuildDataOrganizer::BuildStatisticalDataOrganizer(DataOrganizer *organizer, Inputs *input)
{
    int index = 0;
    for (int j = 0; j < input->DataDrawers.size(); ++j)
    {
        std::vector<DataFolder*> dataDrawer;
        for (int i = 0; i < input->DataDrawers[j].size(); ++i)
        {
            std::map<std::string, double> folderInfo = input->DataDrawers[j][i];
            if (input->StringDataDrawers[j][i]["Type"] == "BulkArea")
            {
                DataFolder *folder = new DataFolder;
                folder->ParamDict = folderInfo;
                folder->ParamStringDict = input->StringDataDrawers[j][i];
                folder->VarDict["aveSMD"] = input->AveSMD;
                AddBulkAreaStatisticalCoverData(folder);
                dataDrawer.push_back(folder);
            }
            ++index;
        }
        organizer->DataDrawers.push_back(dataDrawer);
    }

}
```

You can refer to BuildGrid.cpp in the GI code to see how I was thinking this might be straightened out.